



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ – 7^ο ΕΞΑΜΗΝΟ

Δημητρίου Αγγελική – ΑΜ: 03117106

Τζομάκα Αφροδίτη – ΑΜ: 03117107

Ομάδα 9

3^η Εργαστηριακή Άσκηση

Κρυπτογραφική συσκευή VirtIO για QEMU-KVM

1. ΕΙΣΑΓΩΓΗ – ΣΤΟΧΟΣ ΤΗΣ ΑΣΚΗΣΗΣ:

Στην εργαστηριακή άσκηση αυτή, μας ζητήθηκε να αναπτύξουμε εικονική κρυπτογραφική συσκευή στο περιβάλλον εικονικοποίησης QEMU-KVM, η οποία θα επιτρέπει σε διεργασίες που εκτελούνται μέσα στην εικονική μηχανή να έχουν πρόσβαση σε πραγματική κρυπτογραφική συσκευή του host, τύπου cryptodev-linux, με χρήση τεχνικής παρα-εικονικοποίησης (paravirtualization). Για την υλοποίηση αυτή τροποποιήσαμε και τον κώδικα του QEMU-KVM ώστε να επικοινωνεί ο guest μέσω του πρότυπου VirtIO, με το υλικό του host. Μέρος της άσκησης ήταν επίσης και η υλοποίηση ενός εργαλείου για κρυπτογραφημένη επικοινωνία (chat) πάνω από TCP/IP sockets, με την οποία θα επαληθεύσουμε τη σωστή λειτουργία του συστήματος.

2. Z1: ΕΡΓΑΛΕΙΟ CHAT ΠΑΝΩ ΑΠΟ TCP/IP SOCKETS

Η υλοποίηση του ζητήματος αυτού βασίζεται στην εξής παραδοχή. Θεωρούμε ότι έχουμε έναν server ο οποίος περιμένει σύνδεση από έναν client και έτσι για την υλοποίηση της ιδέας του chat λειτουργεί και αυτός κατά κάποιον τρόπο ως client. Οι δύο χρήστες αυτοί (Aggeliki – Client και Afroditi – Server για προφανείς λόγους) μπορούν να στέλνουν ελεύθερα μηνύματα ο ένας στον άλλον μέσω δύο διαφορετικών terminal. Ταυτόχρονα, έχουν οριστεί 2 δεσμευμένες λέξεις, help και exit, που υλοποιούν

την βοήθεια σχετικά με την λειτουργία του chat και την έξοδο από αυτό αντίστοιχα. Όσον αφορά στην έξοδο αυτή, είναι σημαντικό να αναφέρουμε μία ακόμα παραδοχή. Πιο συγκεκριμένα, προκειμένου να υλοποιηθεί η δυνατότητα απόλυσης της σύνδεσης και από την πλευρά του server πρέπει σε αυτήν την περίπτωση να ενημερωθεί κατάλληλα ο client για να κλείσει εκείνος την σύνδεση. Αυτό το πετυχαίνουμε στέλνοντας κάθε φορά που ο server δέχεται exit μια κωδική λέξη (tyn) στο socket την οποία ο client έχει ρυθμιστεί να την αντιλαμβάνεται ως exit.

Αρχικά, ο server και ο client, για να μπορέσουν να επικοινωνήσουν πάνω από TCP/IP sockets, δημιουργούν ένα socket, μέσω της κλήσης `socket(PF_INET, SOCK_STREAM, 0)`. Έπειτα, ο server κάνει bind, δεσμεύοντας έτσι το socket με μία συγκεκριμένη διεύθυνση (την διεύθυνση του μηχανήματος μας) και πόρτα, μέσω της κλήσης `bind(sd, (struct sockaddr *)&sa, sizeof(sa))`. Έπειτα, κάνει listen, ορίζοντας τις παραμέτρους με τις οποίες από δω και πέρα αρχίζει μπορεί να δέχεται συνδέσεις, μέσω της κλήσης `listen(sd, TCP_BACKLOG)`.

Από την άλλη μεριά ο client, συνδέεται σε μία συγκεκριμένη διεύθυνση και πόρτα (αυτές που έχει ορίσει ο server μας), μέσω της κλήσης `connect(sd, (struct sockaddr *)&sa, sizeof(sa))`. Ταυτόχρονα, ο server τρέχει σε μια αέναη λούπα, περιμένοντας να κάνει connect κάποιος client, μέσω της κλήσης `accept(sd, (struct sockaddr *)&sa, &len)` από την οποία επιστρέφεται ένας νέος file descriptor, όπου σε αυτόν πλέον θα αναφέρεται ο server για την συγκεκριμένη σύνδεση με αυτόν τον client.

Για να μπορέσουν να επικοινωνούν και οι δύο, δηλαδή να γράφει ο ένας στο socket και να μπορεί ο άλλος να διαβάζει μετά, χρειαζόμαστε την `select()` και την δομή δεδομένων `fd_set`, η οποία περιμένει μέχρι κάποιος file descriptor, να είναι έτοιμος για κάποιο I/O operation, όπως read και write στην δικιά μας περίπτωση. Όταν λοιπόν ένας από τους δύο file descriptor είναι έτοιμος για read/write δεδομένων, τότε η `select(newsd + 1, &rdset, NULL, NULL, NULL)` επιστρέφει ανάλογα τον file descriptor που είναι έτοιμος και από κει και πέρα στο πρόγραμμα μας με ένα if statement, αποφασίζουμε αν θα διαβάσουμε από το standard input ή θα διαβάσουμε από το socket.

Αφού γίνει αυτή η διαδικασία η κάθε πλευρά ελέγχει αν έχει γράψει exit, help ή ελεύθερο μήνυμα ο χρήστης (STDIN) ή υπάρχει μήνυμα από την άλλη πλευρά (socket). Το help όπως είπαμε και πριν εμφανίζει στο STDOUT πληροφορίες για τον τρόπο λειτουργίας ενώ το exit αν πρόκειται για τον client απολύει την σύνδεση κάνοντας shutdown, break και close το socket. Με αυτό τον τρόπο ο server διαβάζει 0 bytes και καταλαβαίνει ότι ο client έχει αποχωρήσει (Aggeliki went away). Για τη σωστή λειτουργία του exit από την πλευρά του server όπως εξηγήθηκε παραπάνω, ο client έχει ρυθμιστεί να αναγνωρίζει στην περίπτωση `ready_fd = sd` και μια ακόμα κωδική λέξη, την tyn και να δράσει όπως και στην περίπτωση του exit. Παρακάτω φαίνεται και ο κώδικας του server και του client, που περιέχει αναλυτικά όσα περιγράψαμε.

socket-client.c

```
/*
 * socket-client.c
 * Simple TCP/IP communication using sockets
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 */

#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <netdb.h>

#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>

#include <arpa/inet.h>
#include <netinet/in.h>

#include "socket-common.h"

int is_int(char const *p) //checks if string is equal to the string given by its own conver
sion to int
{
    int c = strlen(p);
    char buffer[c];
    sprintf(buffer, "%d", atoi(p));
    return strcmp(buffer, p) == 0;
}

/* Insist until all of the data has been written */
ssize_t insist_write(int fd, const void *buf, size_t cnt)
{
    ssize_t ret;
    size_t orig_cnt = cnt;

    while (cnt > 0)
    {
        ret = write(fd, buf, cnt);
        if (ret < 0)
            return ret;
        buf += ret;
        cnt -= ret;
    }

    return orig_cnt;
}

int main(int argc, char *argv[])
{
    int sd, port;
```

```

char *hostname;
struct hostent *hp;
struct sockaddr_in sa;

if (argc != 3)
{
    fprintf(stderr, "Usage: %s hostname port\n", argv[0]);
    exit(1);
}
hostname = argv[1];
if (!is_int(argv[2]))
{
    fprintf(stderr, "Port is an integer!!\n");
    exit(1);
}
port = atoi(argv[2]);

/* Create TCP/IP socket, used as main chat channel */
if ((sd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("socket");
    exit(1);
}
fprintf(stderr, "Created TCP socket\n");

/* Look up remote hostname on DNS */
if (!(hp = gethostbyname(hostname))) //performs nslookup and puts ipv4 addr in hp-
>h_addr
{
    printf("DNS lookup failed for host %s\n", hostname);
    exit(1);
}

/* Connect to remote TCP port */
sa.sin_family = AF_INET;
sa.sin_port = htons(port);
memcpy(&sa.sin_addr.s_addr, hp->h_addr, sizeof(struct in_addr));
fprintf(stderr, "Connecting to remote host... ");
fflush(stderr);
if (connect(sd, (struct sockaddr *)&sa, sizeof(sa)) < 0)
{
    perror("connect");
    exit(1);
}
fprintf(stderr, "Connected.\n");

for (;;)
{
    fd_set inset;

    FD_ZERO(&inset);           // we must initialize before each call to select
    FD_SET(STDIN_FILENO, &inset); // select will check for input from stdin
    FD_SET(sd, &inset);         // select will check for input from socket

    // select only considers file descriptors that are smaller than maxfd
    int maxfd = MAX(STDIN_FILENO, sd) + 1; //+1 is needed, select checks fds < 1st argum
ent

```

```

// wait until any of the input file descriptors are ready to receive
int ready_fds = select(maxfd, &inset, NULL, NULL, NULL);

if (ready_fds <= 0)
{
    perror("select");
    continue; // just try again
}

// user has typed something, we can read from stdin without blocking
if (FD_ISSET(STDIN_FILENO, &inset)) //returns if fd is in inset=ready
{
    char buffer[101];

    int n_read = read(STDIN_FILENO, buffer, 100); // at most 100
    if (n_read == -1)
    {
        perror("read");
        exit(-1);
    }
    buffer[n_read] = '\0';

    if (n_read >= 4 && strncmp(buffer, "exit", 4) == 0) //User typed exit
    {
        if (shutdown(sd, SHUT_WR) < 0) //necessary in order to read 0 and not block
        {
            perror("shutdown");
            exit(1);
        }
        break;
    }
    else if (n_read >= 4 && strncmp(buffer, "help", 4) == 0) //User typed help
    {
        printf(YELLOW "Type a message for Afroditi or 'exit' to leave chat." WHITE "\n");
    }
    else
    {
        fprintf(stdout, BLUE"Aggeliki: " WHITE);
        fflush(stdout);
        if (insist_write(1, buffer, n_read) != n_read)
        {
            perror("write");
            exit(1);
        }
        if (insist_write(sd, buffer, n_read) != n_read)
        {
            perror("write");
            exit(1);
        }
    }
}
else
{
    char buffer[101];

```

```

    int n = read(sd, buffer, sizeof(buffer));

    if (n <= 0)
    {
        if (n < 0)
            perror("read");
        else
            fprintf(stderr, RED "\nAfroditi went away\n" WHITE); //will never happen
        break;
    }
    if (n >= 3 && strncmp(buffer, "tyn", 3) == 0) //server wants us to exit
    {
        if (shutdown(sd, SHUT_WR) < 0)
        {
            perror("shutdown");
            exit(1);
        }
        break;
    }
    fprintf(stdout, CYAN"Afroditi: " WHITE);
    fflush(stdout);
    if (insist_write(1, buffer, n) != n)
    {
        perror("write");
        exit(1);
    }
}

fprintf(stderr, GREEN"\nDone.\n"WHITE);
if (close(sd) < 0)
    perror("close");
return 0;
}

```

socket-server.c

```

/*
 * socket-server.c
 * Simple TCP/IP communication using sockets
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 */

#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <netdb.h>

```

```

#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>

#include <arpa/inet.h>
#include <netinet/in.h>

#include "socket-common.h"

/* Insist until all of the data has been written */
ssize_t insist_write(int fd, const void *buf, size_t cnt)
{
    ssize_t ret;
    size_t orig_cnt = cnt;

    while (cnt > 0)
    {
        ret = write(fd, buf, cnt);
        if (ret < 0)
            return ret;
        buf += ret;
        cnt -= ret;
    }

    return orig_cnt;
}

int main(void)
{
    char addrstr[INET_ADDRSTRLEN];
    int sd, newsd;
    socklen_t len;
    struct sockaddr_in sa;

    /* Make sure a broken connection doesn't kill us */
    signal(SIGPIPE, SIG_IGN); //when all read ends are closed, if we try to write -
> raise EPIPE

    /* Create TCP/IP socket, used as main chat channel */
    if ((sd = socket(PF_INET, SOCK_STREAM, 0)) < 0) //domain,type,protocol = def
    {
        perror("socket");
        exit(1);
    }
    fprintf(stderr, "Created TCP socket\n");

    /* Bind to a well-known port */
    memset(&sa, 0, sizeof(sa)); //memory allocation & init
    sa.sin_family = AF_INET; //internet IPv4
    sa.sin_port = htons(TCP_PORT);
    sa.sin_addr.s_addr = htonl(INADDR_ANY); //listen from anyone
    if (bind(sd, (struct sockaddr *)&sa, sizeof(sa)) < 0)
    {
        perror("bind");
        exit(1);
    }
}

```

```

fprintf(stderr, "Bound TCP socket to port %d\n", TCP_PORT);

/* Listen for incoming connections */
if (listen(sd, TCP_BACKLOG) < 0) //The backlog argument defines the maximum length to which the
                                //queue of pending connections for sockfd may grow
{
    perror("listen");
    exit(1);
}

/* Loop forever, accept()ing connections */
for (;;)
{
    fprintf(stderr, "Waiting for an incoming connection...\n");

    /* Accept an incoming connection */
    len = sizeof(struct sockaddr_in);
    if ((newsd = accept(sd, (struct sockaddr *)&sa, &len)) < 0)
    {
        perror("accept");
        exit(1);
    }
    if (!inet_ntop(AF_INET, &sa.sin_addr, addrstr, sizeof(addrstr))) //convert IPv4 and
    IPv6 addresses from binary to text form
    {
        perror("could not format IP address");
        exit(1);
    }
    fprintf(stderr, "Incoming connection from %s:%d\n",
            addrstr, ntohs(sa.sin_port));

    /* We break out of the loop when the remote peer goes away */
    for (;;)
    {
        fd_set inset;

        FD_ZERO(&inset);           // we must initialize before each call to select
        FD_SET(STDIN_FILENO, &inset); // select will check for input from stdin
        FD_SET(newsd, &inset);       // select will check for input from socket

        // select only considers file descriptors that are smaller than maxfd
        int maxfd = MAX(STDIN_FILENO, newsd) + 1; //+1 is needed, select checks fds < 1st
        argument

        // wait until any of the input file descriptors are ready to receive
        int ready_fds = select(maxfd, &inset, NULL, NULL, NULL);

        if (ready_fds <= 0)
        {
            perror("select");
            continue; // just try again
        }

        // user has typed something, we can read from stdin without blocking
        if (FD_ISSET(STDIN_FILENO, &inset)) //returns if fd is in inset=ready
        {

```



```

char buffer[101];

int n_read = read(STDIN_FILENO, buffer, 100); // at most 100
if (n_read == -1)
{
    perror("read");
    exit(-1);
}
buffer[n_read] = '\0';

if (n_read >= 4 && strcmp(buffer, "exit", 4) == 0) //User typed exit
{
    char buf[3] = "tyn";
    if (insist_write(newsd, buf, sizeof(buf)) != sizeof(buf))
    {
        perror("write");
        exit(1);
    }
}
else if (n_read >= 4 && strcmp(buffer, "help", 4) == 0) //User typed help
{
    printf(YELLOW "Type a message for Aggeliki or 'exit' to kick her out." W
HITE "\n");
}
else
{
    fprintf(stdout, CYAN "Afroditi:" WHITE);
    fflush(stdout);
    //echo in stdout of server the message they sent
    if (insist_write(1, buffer, n_read) != n_read)
    {
        perror("write");
        exit(1);
    }
    //and put it in socket for client to read
    if (insist_write(newsd, buffer, n_read) != n_read)
    {
        perror("write");
        exit(1);
    }
}
}
else
{
    char buffer[101];
    int n = read(newsd, buffer, sizeof(buffer));

    if (n <= 0)
    {
        if (n < 0)
            perror("read");
        else
            fprintf(stderr, RED"\nAggeliki went away\n" WHITE);
        break;
    }
    fprintf(stdout, BLUE"Aggeliki: " WHITE);
    fflush(stdout);

```

```

        if (insist_write(1, buffer, n) != n)
        {
            perror("write");
            exit(1);
        }
    }
}
/* Make sure we don't leak open files */
if (close(newsd) < 0)
    perror("close");
}

/* This will never happen */
return 1;
}

```

socket-common.c

```

/*
 * socket-common.h
 *
 * Simple TCP/IP communication using sockets
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 */

#ifndef _SOCKET_COMMON_H
#define _SOCKET_COMMON_H

/* Compile-time options */
#define TCP_PORT 35001
#define TCP_BACKLOG 5

// #define HELLO_THERE "Hello there!"

#define MAX(a, b) ((a) > (b) ? (a) : (b))
#define DEFAULT "\033[30;1m"
#define RED "\033[31m"
#define GREEN "\033[32m"
#define YELLOW "\033[33m"
#define BLUE "\033[34m"
#define MAGENTA "\033[35m"
#define CYAN "\033[36m"
#define WHITE "\033[37m"
#define GRAY "\033[38;1m"

#endif /* _SOCKET_COMMON_H */

```

3. Z2: ΚΡΥΠΤΟΓΡΑΦΗΜΕΝΟ CHAT ΠΑΝΩ ΑΠΟ TCP/IP SOCKETS

Έχοντας φτιάξει το εργαλείο για chat πάνω από TCP/IP sockets για το προηγούμενο ζητούμενο, θα το βελτιώσουμε ώστε τα μηνύματα που στέλνονται να είναι κρυπτογραφημένα. Αυτό θα γίνει χρησιμοποιώντας την κρυπτογραφική συσκευή `/dev/crypto` μέσω του `cryptodev-linux` API από το πρόγραμμα μας, ώστε να μπορούμε να κρυπτογραφούμε κάθε μήνυμα που στέλνουμε και να αποκρυπτογραφούμε κάθε μήνυμα που λαμβάνουμε. Παρακάτω θα αναλύσουμε μόνο αυτά που προσθέσαμε στο πρόγραμμά μας, καθώς το κομμάτι της επικοινωνίας του chat έχει παραμείνει ακριβώς το ίδιο. Επίσης για λόγους ευκολίας ο server και ο client μοιράζονται κοινό κλειδί το οποίο είναι hard-coded στο πεδίο των include των αρχείων και είναι απλά μια τυχαία συμβολοσειρά 16 χαρακτήρων, όπως και το initialization vector.

Αρχικά, ο server και ο client ανοίγουν το αρχείο `/dev/crypto`, μέσω της κλήσης `open("/dev/crypto", O_RDWR)` ώστε να πάρουν τον file descriptor, για χρήση του σε όλες τις διαδικασίες σχετιζόμενες με την κρυπτογράφηση. Έχοντας ορίσει μια μεταβλητή της μορφής `struct session_op sess` η οποία είναι μία δομή που περιέχει πληροφορίες σχετικά με το session που θα δημιουργηθεί για μια κρυπτογράφηση/αποκρυπτογράφηση. Πιο συγκεκριμένα, εμείς ορίζουμε τον αλγόριθμο συμμετρικής κρυπτογράφησης που θα χρησιμοποιηθεί, το μέγεθος κλειδιού (16) και το ίδιο το κλειδί `sess.cipher = CRYPTO_AES_CBC, sess.keylen = KEY_SIZE, sess.key = key`.

Έπειτα μέσω μιας `ioctl` κλήσης στο ανοιχτό αρχείο του `/dev/crypto` με παραμέτρους την δομή `session` που ορίσαμε προηγουμένως και την εντολή `CIOCGSESSION`, θα αρχίσει ένα session με τη συσκευή. Από εκεί και πέρα, κάθε κλήση χρειάζεται να αναφερθεί σε αυτό το session, που περιέχει όλες τις πληροφορίες για τις κλήσεις κρυπτογράφησης/αποκρυπτογράφησης `ioctl(cfd, CIOCGSESSION, &sess)`.

Αφού, ξεκινήσει το session, ο server και ο client ξεκινάνε την επικοινωνία και ανάλογα με τον file descriptor, αν πρόκειται να γράψει στο socket τότε θα πρέπει να κρυπτογραφήσει τα δεδομένα μέσω της συνάρτησης `encrypt_decrypt()` με όρισμα `mode = COP_ENCRYPT` που έχουμε φτιάξει. Αντίστοιχα, αν πρόκειται να διαβάσει από το socket τότε θα πρέπει να αποκρυπτογραφήσει τα δεδομένα μέσω της ίδιας συνάρτησης `encrypt_decrypt()` με όρισμα `mode = COP_DECRYPT`.

Όσον αφορά στη συνάρτηση αυτή, αρχικά ορίζουμε μια μεταβλητή της μορφής `struct crypt_op cryp` η οποία είναι μία δομή που περιέχει πληροφορίες σχετικά με το session που χρησιμοποιείται αλλά και τα δεδομένα που θα κρυπτογραφηθούν/αποκρυπτογραφηθούν. Επίσης έχουμε ορίσει ένα struct που περιέχει ένα buffer για τα επεξεργασμένα δεδομένα με όνομα `dencrypted`. Στη συνέχεια, εμείς ορίζουμε το session που χρησιμοποιείται, το μέγεθος των δεδομένων, τον buffer της πηγής ο οποίος είναι ο global buffer που περιέχει τα δεδομένα προς επεξεργασία καθώς και τον buffer για τα επεξεργασμένα πλέον δεδομένα (`dencrypted`). Επίσης ορίζουμε το καθορισμένο

initialization vector και προφανώς το mode (κρυπτογράφηση ή αποκρυπτογράφηση). Μετά καλούμε την `ioctl()`, με όρισμα τον file descriptor που έχει επιστραφεί από το άνοιγμα του αρχείου `/dev/crypto`, με όρισμα `CIOCCRYPT` για κρυπτογράφηση ή για αποκρυπτογράφηση και την δομή `struct crypt_op cryp` που ορίσαμε προηγουμένως. Τέλος, αντιγράφουμε στον buffer τα κρυπτογραφημένα (ή τα αποκρυπτογραφημένα) δεδομένα από τον `decrypted` μεγέθους `DATA_SIZE` που έχουμε ορίσει ίσο με 256bytes.

Παρακάτω δίνονται οι τροποποιημένοι κώδικες:

socket-client.c

```
/*
 * socket-client.c
 * Simple TCP/IP communication using sockets
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 */

#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <netdb.h>
#include <fcntl.h>

#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>

#include <arpa/inet.h>
#include <netinet/in.h>

#include <crypto/cryptodev.h>
#include "socket-common.h"

unsigned char buffer[DATA_SIZE];
unsigned char key[] = "patatesthganites";
unsigned char iv[] = "xontrolyphenopossum"; //initialization vector used along with secret
key for encryption

int is_int(char const *p) //checks if string is equal to the string given by its own conver
sion to int
{
    int c = strlen(p);
    char buffer[c];
    sprintf(buffer, "%d", atoi(p));
    return strcmp(buffer, p) == 0;
}

/* Insist until all of the data has been written */
ssize_t insist_write(int fd, const void *buf, size_t cnt)
{
    ssize_t ret;
    size_t orig_cnt = cnt;

    while (cnt > 0)
```

```

    {
        ret = write(fd, buf, cnt);
        if (ret < 0)
            return ret;
        buf += ret;
        cnt -= ret;
    }

    return orig_cnt;
}

int encrypt_decrypt(int cfd, struct session_op sess, int mode)
{
    struct crypt_op cryp;
    struct
    {
        unsigned char decrypted[DATA_SIZE];
    } data;
    memset(&cryp, 0, sizeof(cryp));

    cryp.ses = sess.ses;
    cryp.len = (unsigned)256;
    cryp.src = buffer;
    cryp.dst = data.decrypted;
    cryp.iv = iv;
    cryp.op = mode;

    if (ioctl(cfd, CIOCCRYPT, &cryp)) //encrypt or decrypt data using crypto device
    {
        perror("ioctl(CIOCCRYPT)");
        return -1;
    }
    memset(buffer, '\0', 256);
    for (int i = 0; i < DATA_SIZE; i++) //copy to our own global buffer
    {
        buffer[i] = data.decrypted[i];
    }

    return 0;
}

int main(int argc, char *argv[])
{
    int sd, port;
    char *hostname;
    struct hostent *hp;
    struct sockaddr_in sa;
    struct session_op sess;

    int cfd = open("/dev/crypto", O_RDWR); //open crypto device
    if (cfd < 0)
    {
        perror("open(/dev/crypto)");
        return 1;
    }

    memset(&sess, 0, sizeof(sess)); //initialize session

    if (argc != 3)
    {
        fprintf(stderr, "Usage: %s hostname port\n", argv[0]);
        exit(1);
    }
    hostname = argv[1];

```

```

if (!is_int(argv[2]))
{
    fprintf(stderr, "Port is an integer!!\n");
    exit(1);
}
port = atoi(argv[2]);

/* Create TCP/IP socket, used as main chat channel */
if ((sd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("socket");
    exit(1);
}
fprintf(stderr, "Created TCP socket\n");

/* Look up remote hostname on DNS */
if (!(hp = gethostbyname(hostname)))
{
    printf("DNS lookup failed for host %s\n", hostname);
    exit(1);
}

/* Connect to remote TCP port */
sa.sin_family = AF_INET;
sa.sin_port = htons(port);
memcpy(&sa.sin_addr.s_addr, hp->h_addr, sizeof(struct in_addr));
fprintf(stderr, "Connecting to remote host... ");
fflush(stderr);
if (connect(sd, (struct sockaddr *)&sa, sizeof(sa)) < 0)
{
    perror("connect");
    exit(1);
}
fprintf(stderr, "Connected.\n");

sess.cipher = CRYPTO_AES_CBC;
sess.keylen = KEY_SIZE;
sess.key = key;

if (ioctl(cfd, CIOCGSESSION, &sess))
{
    perror("ioctl(CIOCGSESSION)");
    return 1;
}

/* Read answer and write it to standard output */
for (;;)
{
    fd_set inset;

    FD_ZERO(&inset);
    FD_SET(STDIN_FILENO, &inset); // we must initialize before each call to select
    FD_SET(sd, &inset); // select will check for input from stdin
    // select will check for input from socket

    // select only considers file descriptors that are smaller than maxfd
    int maxfd = MAX(STDIN_FILENO, sd) + 1; //+1 is needed, select checks fds < 1st argum
ent

    // wait until any of the input file descriptors are ready to receive
    int ready_fds = select(maxfd, &inset, NULL, NULL, NULL);

    if (ready_fds <= 0)
    {
        perror("select");
    }
}

```

```

        continue; // just try again
    }

    // user has typed something, we can read from stdin without blocking
    if (FD_ISSET(STDIN_FILENO, &inset)) //returns if fd is in inset=ready
    {
        memset(buffer, '\0', 256); //ensures buffer has no leftovers from previous operations
        int n_read = read(STDIN_FILENO, buffer, 256);
        if (n_read == -1)
        {
            perror("read");
            exit(-1);
        }
        buffer[n_read] = '\0';

        if (n_read >= 4 && strcmp(buffer, "exit", 4) == 0) //User typed exit
        {
            if (shutdown(sd, SHUT_WR) < 0) //necessary in order to read 0 and not block
            {
                perror("shutdown");
                exit(1);
            }
            break;
        }
        else if (n_read >= 4 && strcmp(buffer, "help", 4) == 0) //User typed help
        {
            printf(YELLOW "Type a message for Afroditi or 'exit' to leave chat." WHITE "\n");
        }
        else
        {
            fprintf(stdout, BLUE"Aggeliki: " WHITE);
            fflush(stdout);

            if (insist_write(1, buffer, n_read) != n_read)
            {
                perror("write");
                exit(1);
            }

            if (encrypt_decrypt(cfd, sess, COP_ENCRYPT) < 0)
            {
                return 1;
            }

            if (insist_write(sd, buffer, sizeof(buffer)) != sizeof(buffer))
            {
                perror("write");
                exit(1);
            }
        }
    }
    else
    {
        memset(buffer, '\0', 256);
        int n = read(sd, buffer, sizeof(buffer));

        if (n <= 0)
        {
            if (n < 0)
                perror("read");
        }
    }
}

```

```

        else
            fprintf(stderr, RED "\nAfroditi went away\n" WHITE);
            break;
    }
    fprintf(stdout, CYAN"Afroditi: " WHITE);
    fflush(stdout);

    if (encrypt_decrypt(cfd, sess, COP_DECRYPT) < 0)
    {
        return 1;
    }
    if (n >= 3 && strncmp(buffer, "tyn", 3) == 0) //User typed exit
    {
        if (shutdown(sd, SHUT_WR) < 0)
        {
            perror("shutdown");
            exit(1);
        }
        break;
    }
    if (insist_write(1, buffer, sizeof(buffer)) != sizeof(buffer))
    {
        perror("write");
        exit(1);
    }
}

if (ioctl(cfd, CIOCFSESSION, &sess.ses))
{
    printf("it was me!\n");
    perror("ioctl(CIOCFSESSION)");
    return 1;
}

if (close(cfd) < 0)
{
    perror("close");
    return 1;
}

fprintf(stderr, GREEN"\nDone.\n" WHITE);
if (close(sd) < 0)
{
    perror("close");
    return 1;
}
return 0;
}

```

socket-server.c

```

/*
 * socket-server.c
 * Simple TCP/IP communication using sockets
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 */

#include <stdio.h>

```



```

#include <errno.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <netdb.h>
#include <fcntl.h>

#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>

#include <arpa/inet.h>
#include <netinet/in.h>

#include <crypto/cryptodev.h>
#include "socket-common.h"

unsigned char buffer[DATA_SIZE];
unsigned char key[] = "patatesthganites";
unsigned char iv[] = "xontrolyphmenopossum"; //initialization vector used along with secret key for encryption

int encrypt_decrypt(int cfd, struct session_op sess, int mode)
{
    struct crypt_op cryp;
    struct
    {
        unsigned char dencrypted[DATA_SIZE];
    } data;
    memset(&cryp, 0, sizeof(cryp));

    cryp.ses = sess.ses;
    cryp.len = sizeof(buffer);
    cryp.src = buffer;
    cryp.dst = data.dencrypted;
    cryp.iv = iv;
    cryp.op = mode;

    if (ioctl(cfd, CIOCCRYPT, &cryp)) //encrypt or decrypt data using crypto device
    {
        perror("ioctl(CIOCCRYPT)");
        return -1;
    }
    memset(buffer, '\0', sizeof(buffer));
    for (int i = 0; i < DATA_SIZE; i++) //copy to our own global buffer
    {
        buffer[i] = data.dencrypted[i];
    }

    return 0;
}

/* Insist until all of the data has been written */
ssize_t insist_write(int fd, const void *buf, size_t cnt)
{
    ssize_t ret;
    size_t orig_cnt = cnt;

    while (cnt > 0)

```

```

    {
        ret = write(fd, buf, cnt);
        if (ret < 0)
            return ret;
        buf += ret;
        cnt -= ret;
    }

    return orig_cnt;
}

int main(void)
{
    char addrstr[INET_ADDRSTRLEN];
    int sd, newsd;
    socklen_t len;
    struct sockaddr_in sa;

    /* Make sure a broken connection doesn't kill us */
    signal(SIGPIPE, SIG_IGN);

    /* Create TCP/IP socket, used as main chat channel */
    if ((sd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket");
        exit(1);
    }
    fprintf(stderr, "Created TCP socket\n");

    /* Bind to a well-known port */
    memset(&sa, 0, sizeof(sa));
    sa.sin_family = AF_INET;
    sa.sin_port = htons(TCP_PORT);
    sa.sin_addr.s_addr = htonl(INADDR_ANY);
    if (bind(sd, (struct sockaddr *)&sa, sizeof(sa)) < 0)
    {
        perror("bind");
        exit(1);
    }
    fprintf(stderr, "Bound TCP socket to port %d\n", TCP_PORT);

    /* Listen for incoming connections */
    if (listen(sd, TCP_BACKLOG) < 0)
    {
        perror("listen");
        exit(1);
    }

    /* Loop forever, accept()ing connections */
    for (;;)
    {
        struct session_op sess;

        int cfd = open("/dev/crypto", O_RDWR); //open crypto device
        if (cfd < 0)
        {
            perror("open(/dev/crypto)");
            return 1;
        }

        memset(&sess, 0, sizeof(sess)); //initialize session
        fprintf(stderr, "Waiting for an incoming connection...\n");

        /* Accept an incoming connection */
    }
}

```

```

len = sizeof(struct sockaddr_in);
if ((newsd = accept(sd, (struct sockaddr *)&sa, &len)) < 0)
{
    perror("accept");
    exit(1);
}
if (!inet_ntop(AF_INET, &sa.sin_addr, addrstr, sizeof(addrstr)))
{
    perror("could not format IP address");
    exit(1);
}
fprintf(stderr, "Incoming connection from %s:%d\n",
        addrstr, ntohs(sa.sin_port));

sess.cipher = CRYPTO_AES_CBC;
sess.keylen = KEY_SIZE;
sess.key = key;

if (ioctl(cfd, CIOCGSESSION, &sess)) //session starts
{
    perror("ioctl(CIOCGSESSION)");
    return 1;
}
//from here on every call must refer to this session
//contains all info for encrypt - decrypt operations

/* We break out of the loop when the remote peer goes away */
for (;;)
{
    fd_set inset;

    FD_ZERO(&inset);           // we must initialize before each call to select
    FD_SET(STDIN_FILENO, &inset); // select will check for input from stdin
    FD_SET(newsd, &inset);      // select will check for input from socket

    // select only considers file descriptors that are smaller than maxfd
    int maxfd = MAX(STDIN_FILENO, newsd) + 1; //+1 is needed, select checks fds < 1s
t argument

    // wait until any of the input file descriptors are ready to receive
    int ready_fds = select(maxfd, &inset, NULL, NULL, NULL);

    if (ready_fds <= 0)
    {
        perror("select");
        continue; // just try again
    }

    // user has typed something, we can read from stdin without blocking
    if (FD_ISSET(STDIN_FILENO, &inset)) //returns if fd is in inset-ready
    {
        memset(buffer, '\0', 256); //ensures buffer has no leftovers from previous o
perations

        int n_read = read(STDIN_FILENO, buffer, 256);
        if (n_read == -1)
        {
            perror("read");
            exit(-1);
        }
        buffer[n_read] = '\0';

        if (n_read >= 4 && strncmp(buffer, "exit", 4) == 0) //User typed exit
        {
            //buffer is now global

```

```

        buffer[0] = 't';
        buffer[1] = 'y';
        buffer[2] = 'n';
        if (encrypt_decrypt(cfd, sess, COP_ENCRYPT) < 0)
        {
            return 1;
        }
        if (insist_write(newsd, buffer, sizeof(buffer)) != sizeof(buffer))
        {
            perror("write");
            exit(1);
        }
    }
    else if (n_read >= 4 && strcmp(buffer, "help", 4) == 0) //User typed help
    {
        printf(YELLOW "Type a message for Aggeliki or 'exit' to kick her out." W
HITE "\n");
    }
    else
    {
        fprintf(stdout, CYAN "Afroditi: " WHITE);
        fflush(stdout);
        //echo in stdout of server the message they sent
        if (insist_write(1, buffer, n_read) != n_read)
        {
            perror("write");
            exit(1);
        }

        if (encrypt_decrypt(cfd, sess, COP_ENCRYPT) < 0)
        {
            return 1;
        }

        if (insist_write(newsd, buffer, sizeof(buffer)) != sizeof(buffer))
        {
            perror("write");
            exit(1);
        }
    }
}
else
{
    memset(buffer, '\0', 256); //ensures buffer has no leftovers from prev
ious operations
    int n = read(newsd, buffer, sizeof(buffer));

    if (n <= 0)
    {
        if (n < 0)
            perror("read");
        else
            fprintf(stderr, RED"\nAggeliki went away\n" WHITE);
        break;
    }
    fprintf(stdout, BLUE"Aggeliki: " WHITE);
    fflush(stdout);

    if (encrypt_decrypt(cfd, sess, COP_DECRYPT) < 0)
    {
        return 1;
    }

    if (insist_write(1, buffer, sizeof(buffer)) != sizeof(buffer))

```

```

        {
            perror("write");
            exit(1);
        }
    }
}
/* Make sure we don't leak open files */
if (ioctl(cfd, CIOCFSESSION, &sess.ses)) //finish session because client left
{
    perror("ioctl(CIOCFSESSION)");
    return 1;
}

if (close(cfd) < 0)
{
    perror("close");
    return 1;
}
if (close(newsd) < 0)
    perror("close");
}

/* This will never happen */
return 1;
}

```

socket-common.c

```

/*
 * socket-common.h
 *
 * Simple TCP/IP communication using sockets
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 */

#ifndef _SOCKET_COMMON_H
#define _SOCKET_COMMON_H

/* Compile-time options */
#define TCP_PORT 35001
#define TCP_BACKLOG 5

// #define HELLO_THERE "Hello there!"

#define MAX(a, b) ((a) > (b) ? (a) : (b))
#define DEFAULT "\033[30;1m"
#define RED "\033[31m"
#define GREEN "\033[32m"
#define YELLOW "\033[33m"
#define BLUE "\033[34m"
#define MAGENTA "\033[35m"
#define CYAN "\033[36m"
#define WHITE "\033[37m"
#define GRAY "\033[38;1m"

#define DATA_SIZE 256
#define BLOCK_SIZE 16
#define KEY_SIZE 16

#endif /* _SOCKET_COMMON_H */

```

Για να επιβεβαιώσουμε την ορθή λειτουργία της διαδικασίας της (από)κρυπτογράφησης εκτός του απλού τρεξίματος του κώδικα εκτελέσαμε και την εντολή tcpdump με τα ακόλουθα αποτελέσματα:

Γενική λειτουργία chat:

```
aggeliki@aggeliki-inspiron: ~/js-lab/sockets
File Edit View Search Terminal Help
aggeliki@aggeliki-inspiron:~/js-lab/sockets$ ./socket-server
Created TCP socket
Bound TCP socket to port 35001
Waiting for an incoming connection...
Incoming connection from 127.0.0.1:46500
hello
Afrditti:hello
Aggeliki: hi
hello
Afrditti:hello
Aggeliki: heyyy
[]

aggeliki@aggeliki-inspiron: ~/js-lab/sockets
File Edit View Search Terminal Help
aggeliki@aggeliki-inspiron:~/js-lab/sockets$ ./socket-client localhost 35001
Created TCP socket
Connecting to remote host... Connected.
Afrditti: hello
hi
Aggeliki: hi
Afrditti: hello
heyyy
Aggeliki: heyyy
[]
```

Μη κρυπτογραφημένο chat:

```
aggeliki@aggeliki-inspiron: ~/js-lab/sockets
File Edit View Search Terminal Help
aggeliki@aggeliki-inspiron:~/js-lab/sockets$ sudo tcpdump -nt lo -vvv -XXX
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
17:48:02.924888 IP (tos 0x0, ttl 64, id 38577, offset 0, flags [DF], proto TCP (6), length 58)
127.0.0.1.35001 > 127.0.0.1.46500: Flags [P.], cksum 0xf62e (incorrect -> 0x4929), seq 1191914761:1191914767, ack 1189630975, win 512, options [nop,nop,TS
val 3233981498 ecr 3233980805], length 6
0x0000: 0000 0000 0000 0000 0000 0000 0000 4500 .....E.
0x0010: 003a 30b1 4000 4000 a00a 7f00 0001 7f00 ...:~@.....
0x0020: 0001 88b9 b5a4 470b 2d09 46e8 53ff 8018 .....G...F.S...
0x0030: 0200 fe2e 0000 0101 080a c0c2 a43a c0c2 .....v.hello.
0x0040: 043a .....:
17:48:02.924901 IP (tos 0x0, ttl 64, id 12348, offset 0, flags [DF], proto TCP (6), length 52)
127.0.0.1.46500 > 127.0.0.1.35001: Flags [P.], cksum 0xf62e (incorrect -> 0x5f60), seq 1, ack 6, win 512, options [nop,nop,TS val 3233981498 ecr 3233981498],
length 0
0x0000: 0000 0000 0000 0000 0000 0000 0000 4500 .....E.
0x0010: 003a 303c 4000 4000 0c86 7f00 0001 7f00 ...:~@.....
0x0020: 0001 b5a4 88b9 46e8 53ff 470b 2d0f 8018 .....F.S.G....
0x0030: 0200 fe28 0000 0101 080a c0c2 a43a c0c2 .....(.....
0x0040: 043a .....:heyyy.
17:48:06.093969 IP (tos 0x0, ttl 64, id 12349, offset 0, flags [DF], proto TCP (6), length 58)
127.0.0.1.46500 > 127.0.0.1.35001: Flags [P.], cksum 0xf62e (incorrect -> 0xf807), seq 1:7, ack 6, win 512, options [nop,nop,TS val 3233984667 ecr 32339814
98], length 6
0x0000: 0000 0000 0000 0000 0000 0000 0000 4500 .....E.
0x0010: 003a 303d 4000 4000 0c7f 7f00 0001 7f00 ...:~@.....
0x0020: 0001 b5a4 88b9 46e8 53ff 470b 2d0f 8018 .....F.S.G....
0x0030: 0200 fe2e 0000 0101 080a c0c2 b0b9 c0c2 .....(.....
0x0040: 043a .....:heyyy.
17:48:06.093990 IP (tos 0x0, ttl 64, id 38578, offset 0, flags [DF], proto TCP (6), length 52)
127.0.0.1.35001 > 127.0.0.1.46500: Flags [P.], cksum 0xf62e (incorrect -> 0x4698), seq 6, ack 7, win 512, options [nop,nop,TS val 3233984667 ecr 3233984667],
length 0
0x0000: 0000 0000 0000 0000 0000 0000 0000 4500 .....E.
0x0010: 003a 96b2 4000 4000 a60f 7f00 0001 7f00 ...:~@.....
0x0020: 0001 88b9 b5a4 470b 2d0f 46e8 5405 8018 .....G...F.T...
0x0030: 0200 fe28 0000 0101 080a c0c2 b0b9 c0c2 .....(.....
0x0040: b0b9 .....:
[]
```

Κρυπτογραφημένο chat:

```
aggeliki@aggeliki-inspiron: ~/js-lab/sockets
File Edit View Search Terminal Help
aggeliki@aggeliki-inspiron:~/js-lab/sockets$ sudo tcpdump -nt lo -vvv -XXX
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
17:52:30.320720 IP (tos 0x0, ttl 64, id 10485, offset 0, flags [DF], proto TCP (6), length 308)
127.0.0.1.35001 > 127.0.0.1.46504: Flags [P.], cksum 0xf728 (incorrect -> 0xbae3), seq 1062088450:1092088786, ack 2440450804, win 512, options [nop,nop,TS val 3234248894 ecr 3234232130], length 256
0x0000: 0000 0000 0000 0000 0000 0000 0000 4500 .....E.
0x0010: 013a 28a5 4000 4000 131d 7f00 0001 7f00 ...:~@.....
0x0020: 0001 88b9 b5a4 470b 2d0f 46e8 53ff 8018 .....G...F.S...
0x0030: 0200 fe28 0000 0101 080a c0c2 b0b9 c0c2 .....(.....
0x0040: 043a .....:heyyy.
0x0050: 7f00 c0c1 521f d7af e90a 0000 29d1 dc17 wh.....:).
0x0060: 14e0 05fc ba0d b0ed a73d 70af 2a03 c078 ...:~@.....:~@.....
0x0070: 02f0 7770 c0fe 0002 b15c b034 7304 1803 b.....[14...
0x0080: 810c c5f8 f225 53a9 156f 6c0e c13 d070 ...:~@.....:~@.....
0x0090: 9170 7778 532a 85fc 1d0f ad45 ab71 b043 ...:~@.....:~@.....
0x00a0: b15d 5616 000c 0002 a007 0021 2203 8078 ...:~@.....:~@.....
0x00b0: a750 4c63 4b0c f1fe 153c 707d 7208 0445 ...:~@.....:~@.....
0x00c0: 4373 3c05 736c 74ca 1f89 1a0c ac0e 791a ...:~@.....:~@.....
0x00d0: 0202 8a03 0000 1830 1801 5e1a 0000 b045 ...:~@.....:~@.....
0x00e0: 3750 4214 70af 71a8 ad09 f33b 2aff f5e9 W.....[q.....:~@.....
0x00f0: 000a 0709 1402 3e09 a05e 4a7e 1033 070e ...:~@.....:~@.....
0x0100: 1159 7a00 170e 88fc 3518 9205 0e58 47a3 ...:~@.....:~@.....
0x0110: 5c03 000f 07f1 280c 07e4 afad 0a79 0000 ...:~@.....:~@.....
0x0120: 3a2a c71c b0e1 0a03 0a2d 2fa9 0a07 5c35 ...:~@.....:~@.....
0x0130: b0c4 0a04 2210 7306 7209 912c 1552 0145 ...:~@.....:~@.....
0x0140: f0a0 e032 30ca 143e c70c c70a 1400 c000 ...:~@.....:~@.....
0x0150: c70a .....:~@.....
17:52:30.320740 IP (tos 0x0, ttl 64, id 58869, offset 0, flags [DF], proto TCP (6), length 52)
127.0.0.1.46504 > 127.0.0.1.35001: Flags [P.], cksum 0xf628 (incorrect -> 0xb2f1), seq 1, ack 256, win 510, options [nop,nop,TS val 3234248894 ecr 3234248894], length 0
0x0000: 0000 0000 0000 0000 0000 0000 0000 4500 .....E.
0x0010: 013a e018 4000 4000 1508 7f00 0001 7f00 ...:~@.....
0x0020: 0001 b15c 8809 9176 1330 4117 f382 8018 .....v58A....
0x0030: 01fe f028 0000 0101 080a c0c2 b0b9 c0c2 .....(.....
0x0040: b0b9 .....:
17:52:30.320765 IP (tos 0x0, ttl 64, id 58869, offset 0, flags [DF], proto TCP (6), length 308)
127.0.0.1.46504 > 127.0.0.1.35001: Flags [P.], cksum 0xf728 (incorrect -> 0xb05e), seq 1:257, ack 256, win 512, options [nop,nop,TS val 3234252349 ecr 3234248894], length 256
0x0000: 0000 0000 0000 0000 0000 0000 0000 4500 .....E.
0x0010: 013a e019 4000 4000 1508 7f00 0001 7f00 ...:~@.....
0x0020: 0001 b15c 8809 9176 1330 4117 f382 8018 .....v58A....
0x0030: 0200 fe28 0000 0101 080a c0c2 b0b9 c0c2 .....(.....
0x0040: b0b9 e02c 0709 173e 0a39 c312 17e5 43fe ...:~@.....:~@.....
0x0050: 558a d03a 4a0d b5ad 13df 2a09 c039 3122 ...:~@.....:~@.....
0x0060: c70a 0001 b0ed 0a05 09a2 c333 7723 9a30 ...:~@.....:~@.....
0x0070: c70f f0a8 c50c b2af 1807 5729 79c1 1a05 ...:~@.....:~@.....
0x0080: b0d9 0307 030c c0b3 b059 c0b0 0309 b7ad ...:~@.....:~@.....
0x0090: e120 c73d 0705 c7c4 799c b04d 030e b104 ...:~@.....:~@.....
0x00a0: 2a70 0000 0208 0c3a 1915 0703 c013 c02a ...:~@.....:~@.....
0x00b0: 090c 01ab 0a79 0001 b03d 0a07 5a14 0222 ...:~@.....:~@.....
0x00c0: 199a b0e4 1a0c 5000 dec0 880a 0a70 0a70 ...:~@.....:~@.....
0x00d0: f0f0 0116 020c 250a c632 0a04 0a35 4129 ...:~@.....:~@.....
0x00e0: d100 0705 1a08 3a08 1f0b 440c 1a75 18af ...:~@.....:~@.....
```

4. Z3: ΥΛΟΠΟΙΗΣΗ ΣΥΣΚΕΥΗΣ CRYPTODEV ME VIRTIO

Σκοπός αυτού του ζητήματος είναι η υλοποίηση της κρυπτογραφικής συσκευής VirtIO για το QEMU και του αντίστοιχου οδηγού συσκευής για τον guest πυρήνα Linux, μέσα στην εικονική μηχανή. Η συσκευή VirtIO ουσιαστικά θα λαμβάνει τις κλήσεις του guest στο cryptodev και θα τις αποστέλλει για επεξεργασία στην συσκευή cryptodev του host μέσω του μηχανισμού επικοινωνίας VirtIO. Η συσκευή μας υλοποιείται σε δύο μέρη, το frontend (μέσα στο VM) και το backend (μέρος του QEMU). Παρακάτω ακολουθεί ανάλυση για τα μέρη αυτά ξεχωριστά.

Πρωτού ξεκινήσουμε με το frontend, εφαρμόσαμε το patch στον κώδικα του QEMU προκειμένου να αντλήσουμε τον κώδικα του backend και να τεστάρουμε την επικοινωνία με τα δοθέντα test files ("Hello" messages).

Δεν ξεχνάμε κάθε φορά που συνδεόμαστε στο QEMU να εκτελούμε την εντολή:

```
./utopia.sh -device virtio-cryptodev-pci
```

προκειμένου να προσθέσουμε και την συσκευή virtio-cryptodev στον κώδικά του.

Έχοντας πλέον έρθει σε επαφή με το βασικό σκελετό του βοηθητικού κώδικα τόσο στο frontend όσο και στο backend είμαστε έτοιμες για την κατάλληλη τροποποίηση του.

4.1. FRONTEND

Ξεκινήσαμε την υλοποίηση της επικοινωνίας αυτής από την μεριά του frontend. Παρατηρούμε ότι τα αρχεία που χρειάζονται τροποποιήσεις είναι τα **crypto-chrdev.c**, **crypto-module.c**, **crypto.h**.

4.1.1. **crypto-chrdev.c:**

Αποφασίζουμε να ξεκινήσουμε με το αρχείο **crypto-chrdev.c** και συγκεκριμένα την υλοποίηση των συναρτήσεων/μεθόδων **crypto_chrdev_open()**, **crypto_chrdev_release()**, **crypto_chrdev_ioctl()** για τις αντίστοιχες κλήσεις συστήματος που ορίζονται στο **struct file_operations**.

Είναι πολύ βασικό να μιλήσουμε λιγάκι για τις λίστες scatter-gather οι οποίες θα χρησιμοποιηθούν κατά κόρον στον αρχείο αυτό. Γενικά η χρήση τους στον πυρήνα του linux συνίσταται σε DMA μεταφορές δεδομένων από συσκευές I/O. Σε αυτές θα περάσουμε τους pointers των δομών δεδομένων καθώς και pointers για όλα τα πεδία τους (deep copy) ώστε οι διευθύνσεις μνήμης που θα περαστούν από τον kernel του VM στον host να έχουν νόημα.

Επισημαίνεται ότι χρησιμοποιήθηκε η προτεινόμενη δομή της Virtqueue που χρησιμοποιεί η συσκευή virtio-cryptodev-pci για τις κλήσεις συστήματος. Η επικοινωνία μεταξύ frontend και backend στηρίζεται στις λειτουργίες των virtqueues οι οποίες μεταφέρουν τις κλήσεις συστήματος από τον guest στον hypervisor ώστε αυτός να καλέσει τις αντίστοιχες κλήσεις στην πραγματική συσκευή.

➤ **crypto_chrdev_open():**

Η κλήση συστήματος open στο κομμάτι του frontend είναι εκείνη που κάθε φορά που καλείται από τον κώδικα του chat, αναλαμβάνει την ειδοποίηση του backend για άνοιγμα της πραγματικής συσκευής /dev/crypto που υπάρχει στο host μηχανήμα.

Λαμβάνοντας υπόψη τη δομή της συνάρτησης virtqueue_add_sgs που χρησιμοποιεί ένα struct με scatterlists καταλαβαίνουμε ότι πρέπει να δηλώσουμε, αρχικοποιήσουμε και γεμίσουμε σωστά μία σειρά από scatter-gather lists για την αποστολή και λήψη των κατάλληλων δεδομένων προς και από τον host. Αρχικά, βλέπουμε τα ορίσματα out_sgs και in_sgs που αναφέρονται στις scattergather. Γενικά ισχύει ότι οι λίστες αυτές τοποθετούνται με τη σειρά στο προαναφερθέν struct, πρώτα εκείνες που προορίζονται για ανάγνωση και ακολουθούν εκείνες που προορίζονται για εγγραφή. Σύμφωνα με αυτόν τον κανόνα έχουν. Τα out_sgs, in_sgs λοιπόν, ορίζουν τον αριθμό των readable και writable λιστών αντίστοιχα.

Όπως υποδεικνύεται, εδώ θα χρειαστούμε 2 sg lists, μία για την αποστολή του syscall_type και μία για την λήψη του host_fd που προκύπτει με το άνοιγμα του /dev/crypto. Επομένως, αφού κάνουμε το απαιτούμενο memory allocation για τις 2 παραπάνω παραμέτρους, ακολουθούμε τη λογική που αναπτύχθηκε παραπάνω για την ορθή τοποθέτησή τους στο struct sgs.

Σε δεύτερη φάση, τα δεδομένα πρέπει να τοποθετηθούν στο virtqueue, να ενημερωθεί το backend για την ύπαρξή τους και εν συνεχεία να ανανεώσει τους buffers με τα δεδομένα του. Οι τρεις αυτές ενέργειες πρέπει να γίνουν μαζί, αδιαίρετα, η μία μετά την άλλη. Αυτό, σε συνδυασμό με τα hints που υπάρχουν στα άλλα αρχεία, μας οδηγεί στην ανάγκη ορισμού ενός σημαφόρου στο struct crypto_device. Για να πάρουμε λοιπόν αυτόν τον σημαφόρο (και την virtqueue) ανακτούμε το device μέσω του minor number που παίρνουμε από το inode του. «Κλειδωνόμαστε» λοιπόν έως ότου η συνάρτηση virtqueue_get_buf επιστρέψει τους επεξεργασμένους buffers.

Τελικά, όταν αυτό συμβεί, ελέγχουμε αν επέτυχε το open του host μέσω του host_fd που επιστράφηκε και ύστερα το περνάμε στο struct crypto_open_file crof. Για το struct αυτό είχαμε αρχικά δεσμεύσει την απαιτούμενη μνήμη, αντιστοιχίσει το πεδίο crdev του με το crdev που χρησιμοποιούμε, αρχικοποιήσει το crof->host_fd σε -1 και αποθηκεύσει όλο το struct στα private data του filp ώστε να μπορούμε να έχουμε πρόσβαση σε αυτό σε κάθε κλήση συστήματος που αφορά αυτό το ανοιχτό αρχείο.

Δεν ξεχνάμε, τέλος, να αποδεσμεύσουμε την μνήμη που είχαμε δεσμεύσει αρχικά για τις παραμέτρους syscall_type και host_fd.

```
static int crypto_chrdev_open(struct inode *inode, struct file *filp)
{
    int ret = 0;
    int err;
    unsigned int len;
    struct crypto_open_file *crof;
    struct crypto_device *crdev;
    struct virtqueue* vq;
    struct scatterlist sg_sys_type, sg_fd_host, *sgs[2]; //Declaration for scattergather lists
    unsigned int *syscall_type;
    int *host_fd;

    debug("Entering");

    syscall_type = kzalloc(sizeof(*syscall_type), GFP_KERNEL);
    *syscall_type = VIRTIO_CRYPTODEV_SYSCALL_OPEN;
    host_fd = kzalloc(sizeof(*host_fd), GFP_KERNEL);
    *host_fd = -1;

    ret = -ENODEV;
    if ((ret = nonseekable_open(inode, filp)) < 0)
        goto fail;

    /* Associate this open file with the relevant crypto device. */
    crdev = get_crypto_dev_by_minor(iminor(inode));
    if (!crdev)
    {
        debug("Could not find crypto device with %u minor",
            iminor(inode));
        ret = -ENODEV;
        goto fail;
    }
    vq = crdev->vq;

    crof = kzalloc(sizeof(*crof), GFP_KERNEL);
    if (!crof)
    {
        ret = -ENOMEM;
        goto fail;
    }
    crof->crdev = crdev;
    crof->host_fd = -1;
    filp->private_data = crof;

    /**
     * We need two sg lists, one for syscall_type and one to get the
     * file descriptor from the host.
     */
    /* ?? */
    sg_init_one(&sg_sys_type, syscall_type, sizeof(*syscall_type));
    sgs[0] = &sg_sys_type;
    sg_init_one(&sg_fd_host, host_fd, sizeof(*host_fd));
    sgs[1] = &sg_fd_host;

    /**
     * Wait for the host to process our data.
     */
    /* ?? */
    down(&crdev->vq_sem);
    err = virtqueue_add_sgs(vq, sgs, 1, 1, &sg_sys_type, GFP_ATOMIC);
```

```

virtqueue_kick(vq);
while (virtqueue_get_buf(vq, &len) == NULL)
    ; //Wait until host processes the out buff (fd_host)
up(&crdev->vq_sem);

/* If host failed to open() return -ENODEV. */
/* ?? */
if (*host_fd < 0)
{
    ret = -ENODEV;
    goto fail;
}

crof->host_fd = *host_fd;

fail:
kfree(syscall_type); //Must free
kfree(host_fd);
debug("Leaving");
return ret;
}

```

➤ **crypto_chrdev_release():**

Η κλήση συστήματος release στο κομμάτι του frontend είναι εκείνη που κάθε φορά που καλείται από τον κώδικα του chat, ειδοποιεί το backend για το κλείσιμο της πραγματικής συσκευής /dev/crypto που υπάρχει στο host μηχανήμα.

Γενικά, χρησιμοποιούμε την λογική που αναπτύχθηκε παραπάνω για την Open. Εκείνα που αλλάζουν αυτήν τη φορά είναι πρώτον το syscall_type (προφανώς) και η αντιμετώπιση του host_fd ως readable sg list εφόσον θέλουμε απλά να το επιστρέψουμε στο backend. Αυτό σημαίνει ότι για τις αντίστοιχες παραμέτρους της virtqueue_add_sgs θα ισχύει out_sgs = 2, in_sgs = 0. Το host_fd ανακτάται από το struct crof που βρίσκεται στα private data του ανοιχτού αρχείου. Επιπλέον, μετά το επιτυχές κλείσιμο, εδώ πρέπει να απελευθερώσουμε και τη δομή crof, μιας και το αρχείο έχει κλείσει οπότε η μνήμη που διέθετε τις εν λόγω πληροφορίες δεν μας είναι πλέον χρήσιμη.

```

static int crypto_chrdev_release(struct inode *inode, struct file *filp)
{
    int err;
    int ret = 0;
    unsigned int len;
    struct scatterlist sg_sys_type, sg_fd_host, *sgs[2]; //Declaration for scattergather lists
    struct crypto_open_file *crof = filp->private_data;
    struct crypto_device *crdev = crof->crdev;
    struct virtqueue* vq = crdev->vq;
    unsigned int *syscall_type;
    int *host_fd;

    debug("Entering");

    syscall_type = kzalloc(sizeof(*syscall_type), GFP_KERNEL);
    *syscall_type = VIRTIO_CRYPTODEV_SYSCALL_CLOSE;
    host_fd = kzalloc(sizeof(*host_fd), GFP_KERNEL); //Retrieve host_fd from private data
}

```

```

*host_fd = crof->host_fd;

/**
 * Send data to the host.
 */
/* ?? */
sg_init_one(&sg_sys_type, syscall_type, sizeof(*syscall_type));
sgs[0] = &sg_sys_type;
sg_init_one(&sg_fd_host, host_fd, sizeof(*host_fd));
sgs[1] = &sg_fd_host;

/**
 * Wait for the host to process our data.
 */
/* ?? */
down(&crdev->vq_sem);
err = virtqueue_add_sgs(vq, sgs, 2, 0, &sg_sys_type, GFP_ATOMIC);
virtqueue_kick(vq);
while (virtqueue_get_buf(vq, &len) == NULL)
    ; //Wait until host processes the out buff (fd_host)
up(&crdev->vq_sem);

kfree(syscall_type);
kfree(host_fd);
kfree(crof);
debug("Leaving");
return ret;
}

crof->host_fd = *host_fd;

fail:
kfree(syscall_type); //Must free
kfree(host_fd);
debug("Leaving");
return ret;
}

```

➤ crypto_chrdev_ioctl():

Η κλήση συστήματος `ioctl` στο κομμάτι του frontend είναι εκείνη που κάθε φορά που καλείται από τον κώδικα του chat, ειδοποιεί το backend για την εκτέλεση της κατάλληλης `ioctl` (`mode = cmd = {CIOCGSESSION, CIOCFSESSION, CIOCCRYPT}`) κλήσης της πραγματικής συσκευής `/dev/crypto` που υπάρχει στο host μηχανήμα.

Αρχικά, όπως και πριν, δηλώνουμε τις παραμέτρους που είναι κοινές για όλα τα modes της `ioctl` και δεσμεύουμε την απαραίτητη μνήμη για αυτές (`syscall_type = VIRTIO_CRYPTODEV_SYSCALL_IOCTL`, `host_fd = crof->host_fd`, `ioctl_cmd = cmd`, `host_return_val`). Όλες οι προηγούμενες, πλην της `host_return_val` (`writable` - τιμή επιστροφής της `ioctl` του host), είναι `readable` επομένως εισάγονται εξαρχής στο struct `sgs` που περιέχει τις `sg` lists. Επίσης εξαιτίας του πλήθους των παραμέτρων για το 'μέτρημά' τους χρησιμοποιήθηκαν δύο μεταβλητές/counters (`num_out` - `readable`, `num_in` - `writable`) για διευκόλυνση.

Στη συνέχεια, ανάλογα με το cmd, ακολουθώντας την προτεινόμενη δομή θα ορίσουμε τα αντίστοιχα πεδία της. Για τον ορισμό αυτό, χρειαζόμαστε τις πληροφορίες του session που όρισε ο χρήστης στον κώδικα του chat. Αυτές τις παίρνουμε μέσω copy_from_user κλήσεων εφόσον ανήκουν σε user space και εμείς θέλουμε να τις αποθηκεύσουμε σε μεταβλητές του kernel.

- **CIOCGSESSION:** Παίρνουμε από τον χρήστη τον pointer στο struct session_op και το αντίστοιχο session_key. Προσθέτουμε στο struct των sgs το key ως readable παράμετρο, ενώ τα session_op και host_return_val ως writeable. Όσον αφορά το session_op προφανώς και εκείνο ανήκει στις writable παραμέτρους μιας και περιέχει πληροφορίες για το session που άνοιξε στον host και οφείλει να τις γνωρίζει και το frontend (θα φανεί αργότερα).
- **CIOCFSESSION:** Με την ίδια λογική, παίρνουμε από τον χρήστη το id του session που πρέπει να τερματιστεί και το τοποθετούμε ως readable παράμετρο στο sgs μαζί με το host_return_val.
- **CIOCCRYPT:** Εδώ θα χρειαστούμε από το χρήστη τον pointer στο struct crypt_op και τα πεδία src (δεδομένα προς κρυπτογράφηση /αποκρυπτογράφηση) και iv. Τα πεδία αυτά θα προστεθούν στις readable παραμέτρους του sgs ενώ ως writeable θα καταχωρηθούν το host_return_val (προφανώς) και το dst (καινούρια μεταβλητή που θα περαστεί αργότερα στο πεδίο dst του crypt_op).

Ύστερα από την επιτυχή (ελπίζουμε) εκτέλεση της ioctl στον host και της απόκτησης των επεξεργασμένων δεδομένων πρέπει να φροντίσουμε και την επιστροφή τους (όπου χρειάζεται) στα structs του user. Για το λόγο αυτό θα χρησιμοποιηθεί η συνάρτηση copy_to_user.

- **CIOCGSESSION:** Το πεδίο session_op στο *arg.
- **CIOCFSESSION:** Τίποτα.
- **CIOCCRYPT:** Το πεδίο dst στο *arg->dst (δεδομένα από κρυπτογράφηση /αποκρυπτογράφηση).

Τέλος, όπως πάντα, αποδεσμεύουμε την μνήμη χρησιμοποιώντας και εδώ ένα switch ανάλογα με τις παραμέτρους που χρησιμοποιήθηκαν σε κάθε cmd.

```
static long crypto_chrdev_ioctl(struct file *filp, unsigned int cmd,
                               unsigned long arg)
{
    long ret = 0;
    int err;
    struct crypto_open_file *crof = filp->private_data;
    struct crypto_device *crdev = crof->crdev;
    struct virtqueue *vq = crdev->vq;
    struct scatterlist syscall_type_sg,
        sg_fd_host, sg_ioctl_cmd, sg_host_return_val,
        sg_session_key, sg_session_op,
        sg_ses_id, sg_crypt_op, sg_src, sg_iv, sg_dst,
        *sgs[8];
    unsigned int num_out, num_in, len;
```

```

unsigned int *syscall_type, *ioctl_cmd;
int *host_fd, *host_return_val;
unsigned char *session_key = NULL, *src = NULL, *iv = NULL, *dst = NULL;
struct session_op *session_op = NULL;
struct crypt_op *crypt_op = NULL;
__u32 *ses_id = NULL;

debug("Entering");

/**
 * Allocate all data that will be sent to the host.
 */
syscall_type = kzalloc(sizeof(*syscall_type), GFP_KERNEL);
*syscall_type = VIRTIO_CRYPTODEV_SYSCALL_IOCTL;
host_fd = kzalloc(sizeof(*host_fd), GFP_KERNEL); //Retrieve host_fd from private data
*host_fd = crof->host_fd;
ioctl_cmd = kzalloc(sizeof(*ioctl_cmd), GFP_KERNEL);
*ioctl_cmd = cmd;
host_return_val = kzalloc(sizeof(*host_return_val), GFP_KERNEL);

num_out = 0;
num_in = 0;

/**
 * These are common to all ioctl commands.
 */
sg_init_one(&syscall_type_sg, syscall_type, sizeof(*syscall_type));
sgs[num_out++] = &syscall_type_sg;
/* ?? */
sg_init_one(&sg_fd_host, host_fd, sizeof(*host_fd));
sgs[num_out++] = &sg_fd_host;
sg_init_one(&sg_ioctl_cmd, ioctl_cmd, sizeof(*ioctl_cmd));
sgs[num_out++] = &sg_ioctl_cmd;

/**
 * Add all the cmd specific sg lists.
 */
switch (cmd)
{
case CIOCGSESSION:
    debug("CIOCGSESSION");

    session_op = kzalloc(sizeof(*session_op), GFP_KERNEL);
    if (copy_from_user(session_op, (void __user *)arg, sizeof(*session_op)))
    {
        ret = -EFAULT;
        goto fail;
    }

    session_key = kzalloc(session_op->keylen * sizeof(*session_key), GFP_KERNEL);
    if (copy_from_user(session_key, session_op->key, session_op->keylen * sizeof(*session_key)))
    {
        ret = -EFAULT;
        goto fail;
    }

    sg_init_one(&sg_session_key, session_key, session_op->keylen * sizeof(*session_key));
    sgs[num_out++] = &sg_session_key;
    sg_init_one(&sg_session_op, session_op, sizeof(*session_op));
    sgs[num_out + num_in++] = &sg_session_op;
    sg_init_one(&sg_host_return_val, host_return_val, sizeof(*host_return_val));
    sgs[num_out + num_in++] = &sg_host_return_val;

    break;

```

```

case CIOCFSESSION:
    debug("CIOCFSESSION");

    ses_id = kzalloc(sizeof(*ses_id), GFP_KERNEL);
    if (copy_from_user(ses_id, (void __user *)arg, sizeof(*ses_id)))
    {
        ret = -EFAULT;
        goto fail;
    }

    sg_init_one(&sg_ses_id, ses_id, sizeof(*ses_id));
    sgs[num_out++] = &sg_ses_id;
    sg_init_one(&sg_host_return_val, host_return_val, sizeof(*host_return_val));
    sgs[num_out + num_in++] = &sg_host_return_val;

    break;

case CIOCCRYPT:
    debug("CIOCCRYPT");

    crypt_op = kzalloc(sizeof(*crypt_op), GFP_KERNEL);
    if (copy_from_user(crypt_op, (void __user *)arg, sizeof(*crypt_op)))
    {
        ret = -EFAULT;
        goto fail;
    }
    src = kzalloc(crypt_op->len * sizeof(*src), GFP_KERNEL);
    if (copy_from_user(src, crypt_op->src, crypt_op->len * sizeof(*src)))
    {
        ret = -EFAULT;
        goto fail;
    }
    iv = kzalloc(VIRTIO_CRYPTODEV_BLOCK_SIZE * sizeof(*iv), GFP_KERNEL);
    if (copy_from_user(iv, crypt_op->iv, VIRTIO_CRYPTODEV_BLOCK_SIZE * sizeof(*iv)))
    {
        ret = -EFAULT;
        goto fail;
    }
    dst = kzalloc(crypt_op->len * sizeof(*dst), GFP_KERNEL);

    sg_init_one(&sg_crypt_op, crypt_op, sizeof(*crypt_op));
    sgs[num_out++] = &sg_crypt_op;
    sg_init_one(&sg_src, src, crypt_op->len * sizeof(*src));
    sgs[num_out++] = &sg_src;
    sg_init_one(&sg_iv, iv, VIRTIO_CRYPTODEV_BLOCK_SIZE * sizeof(*iv));
    sgs[num_out++] = &sg_iv;
    sg_init_one(&sg_dst, dst, crypt_op->len * sizeof(*dst));
    sgs[num_out + num_in++] = &sg_dst;
    sg_init_one(&sg_host_return_val, host_return_val, sizeof(*host_return_val));
    sgs[num_out + num_in++] = &sg_host_return_val;

    break;

default:
    debug("Unsupported ioctl command");

    break;
}

/**
 * Wait for the host to process our data.
 */

```

```

/* ?? */
/* ?? Lock ?? */
down(&crdev->vq_sem);
err = virtqueue_add_sgs(vq, sgs, num_out, num_in,
                        &syscall_type_sg, GFP_ATOMIC);

virtqueue_kick(vq);
while (virtqueue_get_buf(vq, &len) == NULL)
    /* do nothing */;
up(&crdev->vq_sem);

ret = *host_return_val;

switch (cmd)
{
case CIOCGSESSION:
    if (copy_to_user((void __user *)arg, session_op, sizeof(*session_op)))
    {
        ret = -EFAULT;
        goto fail;
    }
    break;
case CIOCFSESSION:
    break;
case CIOCCRYPT:
    if (copy_to_user(((struct crypt_op *)arg)->dst, dst, crypt_op->len * sizeof(*dst)))
    {
        ret = -EFAULT;
        goto fail;
    }
    break;
default:
    debug("Unsupported ioctl command");
    break;
}

fail:

kfree(syscall_type);
kfree(host_fd);
kfree(ioctl_cmd);
kfree(host_return_val);

switch (cmd)
{
case CIOCGSESSION:
    kfree(session_op);
    kfree(session_key);
    break;
case CIOCFSESSION:
    kfree(ses_id);
    break;
case CIOCCRYPT:
    kfree(crypt_op);
    kfree(src);
    kfree(iv);
    kfree(dst);
    break;
default:
    debug("Unsupported ioctl command");
    break;
}

debug("Leaving");

```

```
    return ret;
}
```

4.1.2. crypto.h:

Λόγω της ανάγκης χρήσης κάποιου lock στην μέθοδο open και του hint `/* ?? Lock ??` `*/` καταλαβαίνουμε ότι πρέπει να ορίσουμε έναν σημαφόρο (σημαφόρο γιατί σε αντίθεση με το spinlock δεν χρησιμοποιεί busy wait και μπορεί να διατηρηθεί για μεγαλύτερες χρονικές περιόδους).

```
/**
 * Device info.
 */
struct crypto_device {
    /* Next crypto device in the list, head is in the crdrvdata struct */
    struct list_head list;

    /* The virtio device we are associated with. */
    struct virtio_device *vdev;

    struct virtqueue *vq;
    /* ?? Lock ?? */
    struct semaphore vq_sem;
    /* The minor number of the device. */
    unsigned int minor;
};
```

4.1.3. crypto-module.c:

Μετά την δήλωση του σημαφόρου πρέπει να συμβεί και η αρχικοποίηση.

```
/**
 * This function is called each time the kernel finds a virtio device
 * that we are associated with.
 */
static int virtcons_probe(struct virtio_device *vdev)
{
    int ret = 0;
    struct crypto_device *crdev;

    debug("Entering");

    crdev = kzalloc(sizeof(*crdev), GFP_KERNEL);
    if (!crdev) {
        ret = -ENOMEM;
        goto out;
    }

    crdev->vdev = vdev;
    vdev->priv = crdev;

    crdev->vq = find_vq(vdev);
```



```

if (!(crdev->vq)) {
    ret = -ENXIO;
    goto out;
}

/* Other initializations. */
/* ?? */
sema_init(&crdev->vq_sema, 1);

/**
 * Grab the next minor number and put the device in the driver's list.
 */
spin_lock_irq(&crdrvdata.lock);
crdev->minor = crdrvdata.next_minor++;
list_add_tail(&crdev->list, &crdrvdata.devs);
spin_unlock_irq(&crdrvdata.lock);
debug("Got minor = %u", crdev->minor);

debug("Leaving");

out:
return ret;
}

```

4.2. BACKEND

Το backend από τη μεριά του έχει ως στόχο την λήψη δεδομένων από τον VirtIO ring buffer όταν ειδοποιείται από το frontend ότι υπάρχουν διαθέσιμα δεδομένα στην Virtqueue και στη συνέχεια να εκτελεί τις κατάλληλες ενέργειες.

Παρακάτω φαίνεται η συνάρτηση `vq_handle_output` που είναι η μοναδική που χρειάστηκε να συμπληρώσουμε από την μεριά του backend που προέκυψε από το patching.

Η συνάρτηση αυτή, αφού καταφέρει να πάρει κάποιο στοιχείο (`elem`) από τη Virtqueue με τη `virtqueue_pop` ξεκινά αντλώντας από αυτό είδος του syscall που μεταφέρει και το οποίο βρίσκεται πάντα στη πρώτη θέση των `out_sgs`. Ύστερα ανάλογα με το syscall type έχουμε τα εξής:

- **open:** Σύνδεση της τοπικής μεταβλητής `host_fd` με την `host_fd` του frontend που βρίσκεται στην πρώτη θέση του `in_sgs`. Εκτελεί την `open` στον host και αναθέτει το fd που προκύπτει στο `host_fd`.
- **close:** Σύνδεση της τοπικής μεταβλητής `host_fd` με την `host_fd` του frontend που βρίσκεται στη δεύτερη θέση του `out_sgs` αυτήν την φορά. Εκτελεί το `close` για το αρχείο που υποδεικνύει το συγκεκριμένο fd.
- **ioctl:** Σύνδεση των τοπικών μεταβλητών `host_fd` και `ioctl_cmd` με τις `host_fd` και `ioctl_cmd` του frontend που βρίσκονται στην δεύτερη και τρίτη θέση του `out_sgs` αντίστοιχα. Επίσης συνδέεται η τοπική μεταβλητή `host_return_val` με τη `host_return_val` του frontend που βρίσκεται στο `in_sgs`. Ανάλογα με το `ioctl_cmd`

κάνουμε την ίδια διαδικασία για τα επιμέρους πεδία κάθε command mode, τρέχουμε το αντίστοιχο ioctl και περνάμε το host_return_val.

Τέλος, το backend μέσω της virtqueue_push κάνει push το στοιχείο που έλαβε αρχικά πίσω στην Virtqueue ώστε να ολοκληρωθεί το busy wait στην πλευρά του frontend.

Παρατηρούμε ωστόσο, ότι στον κώδικα που μας δόθηκε αμέσως μετά την συνάρτηση virtqueue_push, καλείται και η συνάρτηση virtio_notify. Η συνάρτηση αυτή ειδοποιεί το frontend σχετικά με την ολοκλήρωση των εργασιών του backend, με χρήση ενός interrupt. Η λειτουργία αυτή επιτρέπει στο frontend να αντικαταστήσει το busy wait με αναμονή με sleep έως ότου έρθει αυτό το interrupt. Εφόσον όμως στον αρχικό κώδικα που μας δόθηκε, το frontend χρησιμοποιούσε αποκλειστικά busy wait, κατά την συγγραφή του κώδικα ακολουθήσαμε το ίδιο σκεπτικό, με αποτέλεσμα η εντολή virtio_notify να είναι πλέον αχρείαστη.

```
static void vq_handle_output(VirtIODevice *vdev, VirtQueue *vq)
{
    VirtQueueElement *elem;
    unsigned int *syscall_type, *ioctl_cmd, *ses_id;
    int *host_fd, *host_return_val;
    struct session_op *session_op;
    struct crypt_op *crypt_op;

    DEBUG_IN();

    elem = virtqueue_pop(vq, sizeof(VirtQueueElement));
    if (!elem) {
        DEBUG("No item to pop from VQ :(");
        return;
    }

    DEBUG("I have got an item from VQ :)");

    syscall_type = elem->out_sg[0].iov_base;
    switch (*syscall_type) {
    case VIRTIO_CRYPTODEV_SYSCALL_TYPE_OPEN:
        DEBUG("VIRTIO_CRYPTODEV_SYSCALL_TYPE_OPEN");
        /*??*/
        host_fd = elem -> in_sg[0].iov_base;
        *host_fd = open("/dev/crypto", O_RDWR);
        if(*host_fd<0) {
            DEBUG("Unable to open /dev/crypto");
            perror("open");
            return;
        }

        break;

    case VIRTIO_CRYPTODEV_SYSCALL_TYPE_CLOSE:
        DEBUG("VIRTIO_CRYPTODEV_SYSCALL_TYPE_CLOSE");
        /* ?? */
        host_fd = elem -> out_sg[1].iov_base;
        close(*host_fd);
        if(*host_fd<0) {
```

```

        DEBUG("Unable to close");
        perror("close");
        return;
    }

    break;

case VIRTIO_CRYPTODEV_SYSCALL_TYPE_IOCTL:
    DEBUG("VIRTIO_CRYPTODEV_SYSCALL_TYPE_IOCTL");
    /* ?? */
    host_fd = elem->out_sg[1].iov_base;
    ioctl_cmd = elem->out_sg[2].iov_base;

    switch (*ioctl_cmd) {
    case CIOCGSESSION:
        DEBUG("CIOCGSESSION");
        session_op = elem->in_sg[0].iov_base;
        session_op->key = elem->out_sg[3].iov_base;
        host_return_val = elem->in_sg[1].iov_base;
        *host_return_val = ioctl(*host_fd, CIOCGSESSION, session_op);
        if(*host_return_val) {
            DEBUG("Error in CIOCGSESSION");
            perror("ioctl");
        }

        break;

    case CIOCFSESSION:
        DEBUG("CIOCFSESSION");
        ses_id = elem->out_sg[3].iov_base;
        host_return_val = elem->in_sg[0].iov_base;
        *host_return_val = ioctl(*host_fd, CIOCFSESSION, ses_id);
        if(*host_return_val) {
            DEBUG("Error in CIOCFSESSION");
            perror("ioctl");
        }

        break;

    case CIOCCRYPT:
        DEBUG("CIOCCRYPT");
        crypt_op = elem->out_sg[3].iov_base;
        crypt_op->src = elem->out_sg[4].iov_base;
        crypt_op->iv = elem->out_sg[5].iov_base;
        crypt_op->dst = elem->in_sg[0].iov_base;
        host_return_val = elem->in_sg[1].iov_base;
        *host_return_val = ioctl(*host_fd, CIOCCRYPT, crypt_op);
        if(*host_return_val) {
            DEBUG("Error in CIOCCRYPT");
            perror("ioctl");
        }

        break;

    default:
        DEBUG("Unknown ioctl_cmd");
    }
}

```

```
        break;

    default:
        DEBUG("Unknown syscall_type");
        break;
}

virtqueue_push(vq, elem, 0);
virtio_notify(vdev, vq);
g_free(elem);
}
```

5. ΠΡΟΒΛΗΜΑΤΑ

Σε αυτό το μέρος αναφέρονται προβλήματα – προβληματισμοί που αντιμετωπίστηκαν καθ’ όλη τη διάρκεια της άσκησης.

Όσον αφορά τα 2 πρώτα μέρη, ήμασταν εξοικειωμένες με την βασική ιδέα επικοινωνίας μέσω BSD sockets και τη χρήση της select από προηγούμενο εξάμηνο. Για το λόγο αυτό οι δυσκολίες που αντιμετωπίσαμε είχαν να κάνουν με τεχνικά ζητήματα και κυρίως με την λειτουργία κατά το κλείσιμο των client και server (χρήση exit – shutdown). Αρχικά γινόταν ένα «άτσαλο» κλείσιμο του socket από την μεριά του server ο οποίος τερμάτιζε χωρίς όμως να έχει ενημερώσει το client για αυτό. Εν τέλει το πρόβλημα διορθώθηκε σύμφωνα με τις παραδοχές που εξηγούνται αναλυτικά στην παράγραφο [2].

Κατά βάση προβλήματα υπήρχαν στο τελευταίο μέρος, περισσότερο κατά την αρχική κατανόηση της λειτουργίας και κάποιων δομών όπως οι scatter – gather λίστες (οι οποίες είναι ανεπαρκώς documented). Αναλυτικότερα, αρχικά υπήρξε παρανόηση για την θέση του κώδικα του backend, αφού όμως διαβάσαμε προσεκτικότερα τα αρχεία της άσκησης καταλάβαμε ότι πρόκειται για αρχείο στο building directory του qemu. Αυτό συνεπάγεται και την ανάγκη για make και make install μετά από κάθε αλλαγή στο αρχείο αυτό.