Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

8ο εξάμηνο

2017-2018

Εργαστήριο Λειτουργικών Συστημάτων

3η Εργαστηριακή Άσκηση
Κρυπτογραφική συσκευή VirtIO για QEMU-KVM

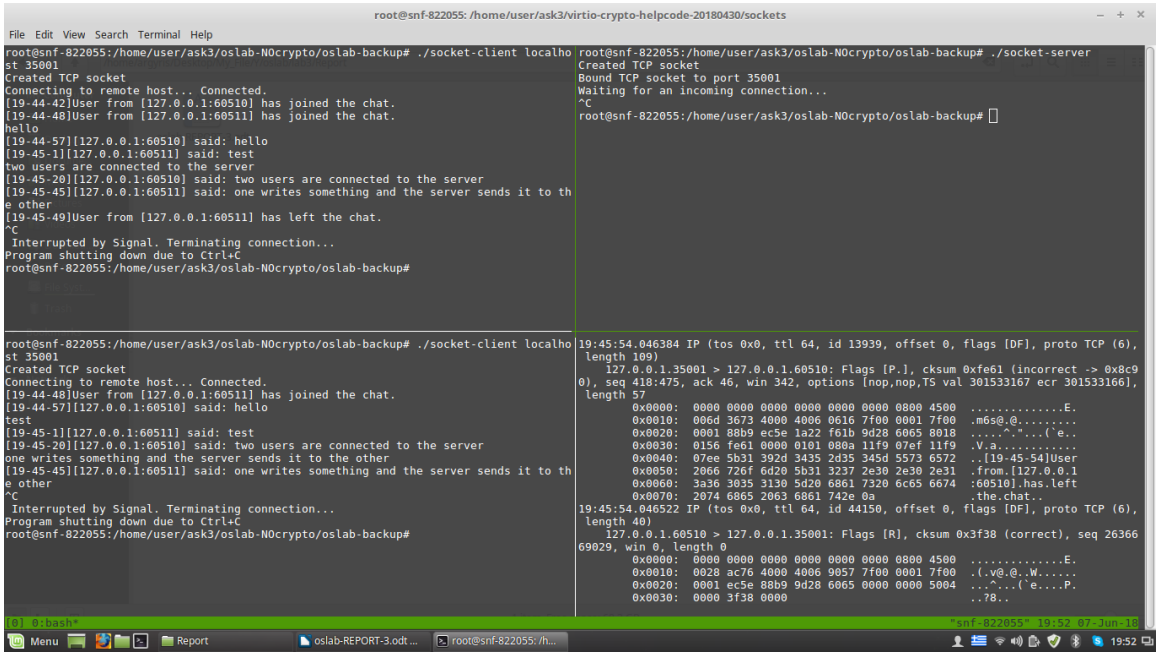
Ομάδα 18
Μουζάκης Ανάργυρος-Γεώργιος
Μουζάκης Νικόλαος

Εισαγωγή

Η παρούσα άσκηση αποτελούταν από 3 ζητούμενα.
Στα πλαίσια του πρώτου ζητούμενου κληθήκαμε να υλοποιήσουμε έναν ***μηχανισμό chat πάνω από TCP/IP***, η λειτουργία του οποίου θα βασιζόταν στο ***BSD Sockets API***. Σε αυτό το μέρος της άσκησης, η επικοινωνία θα πραγματοποιούταν χωρίς κρυπτογράφηση.
Στα πλαίσια του δεύτερου ζητούμενου, ο προηγούμενος μηχανισμός έπρεπε να συμπληρωθεί κατάλληλα, προκειμένου τα μηνύματα που ανταλλάσσονται να είναι ***κρυπτογραφημένα***. Αυτό επιτυγχάνεται με χρήση της συσκευής χαρακτήρων ***cryptodev***.
Στα πλαίσια του τρίτου ζητούμενου, το ίδιο πράγμα έπρεπε να γίνει μέσα σε περιβάλλον εικονικής μηχανής ***QEMU-KVM***. Συγκεκριμένα, έπρεπε να αξιοποιηθούν οι δυνατότητες πραγματοποίησης κλήσεων σε ***paravirtualized*** υλικό, με σκοπό την επιτάχυνση της διαδικασίας κρυπτογράφησης. Η υλοποίηση βασίστηκε στο πρότυπο ***VirtIO***, που είναι ένα ***split-driver model***. Συγκεκριμένα, όταν πραγματοποιείται μία κλήση στο cryptodev από κώδικα εντός της εικονικής μηχανής, ο driver της συσκευής είναι υλοποιημένος έτσι ώστε να πραγματοποιεί ένα ***hypercall*** στο hypervisor, που είναι πρόγραμμα που εκτελείται στο χώρο χρήστη του εξωτερικού μηχανήματος. Αυτό, με τη σειρά του, πραγματοποιεί μία κλήση συστήματος στη συσκευή cryptodev, με αποτέλεσμα να πραγματοποιείται η διαδικασία κρυπτογράφησης, χωρίς να είναι απαραίτητη η υλοποίησή της σε επίπεδο λογισμικού στην εικονική μηχανή.

Ζητούμενο 1

Υλοποιήσαμε το μηχανισμό επικοινωνίας με βάση τον κώδικα που δόθηκε. Δώσαμε τη δυνατότητα πολλοί ***clients*** να είναι συνδεδεμένοι στον ίδιο ***server***. Συγκεκριμένα, όταν κάποιος συνδέεται στο server, ο server στέλνει ειδοποίηση σε όλους τους clients που είναι συνδεδεμένοι σε αυτόν εκείνη τη στιγμή, προκειμένου να τους ενημερώση. Το ίδιο συμβαίνει και στην περίπτωση που κάποιος τερματίζει τη σύνδεσή του με το server. Επιπλέον, κάθε φορά που οποιοσδήποτε από τους clients γράφει κάποιο μήνυμα, ο server αναλαμβάνει να το προωθήσει και στους υπόλοιπους.
Η περιγραφόμενη συμπεριφορά φαίνεται στο παρακάτω screenshot.



Στα παράθυρα στο αριστερό μέρος της εικόνας φαίνονται οι clients. Στο δεξί, στο μεν πάνω παράθυρο έχουμε τον server, στο δε κάτω παράθυρο φαίνεται το αποτέλεσμα της εκτέλεσης της εντολής ***tcpdump -ni lo -vvv -XXX***. Όπως προκύπτει από τα μηνύματα σε αυτό το παράθυρο, η συνομιλία δεν είναι κρυπτογραφημένη.
Ο κώδικας των ***socket-client.c*** και ***socket-server.c*** φαίνεται παρακάτω:

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"
/* 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;
}

static int running;

static void handler (int signum)
{
	running=0;
}


int main(int argc, char *argv[])
{
	struct sigaction sigact;
	running=1;
	sigact.sa_handler=handler;
```

```
sigact.sa_flags=SA_RESTART;
sigaction(SIGINT,&sigact,NULL);
int sd, port;
ssize_t n;
char buf[buff_size];
char *hostname;
struct hostent *hp;
struct sockaddr_in sa;
fd_set rfds;

if (argc != 3) {
        fprintf(stderr, "Usage: %s hostname port\n", argv[0]);
        exit(1);
}
hostname = argv[1];
port = atoi(argv[2]); /* Needs better error checking */

/* 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");

/* Be careful with buffer overruns, ensure NUL-termination */
/* Say something... */
FD_ZERO(&rfds);
FD_SET(0,&rfds);
FD_SET(sd,&rfds);

while(running) {
        FD_ZERO(&rfds);
```

```
		FD_SET(0,&rfds);
		FD_SET(sd,&rfds);
		if (select(sd+1,&rfds,NULL,NULL,NULL)< 0) {
			if(errno==EINTR) {
				printf("\n Interrupted by Signal. Terminating connection…
\n");
				continue;
			}
			else
			perror("select");
			exit(1);
		}
		if (FD_ISSET(0,&rfds)) {
			n = read(0, buf, sizeof(buf)-1);

			if (n < 0) {
				perror("read");
				exit(1);
			}
			buf[n]='\0';
			buf[sizeof(buf)-1]='\0';
			if (insist_write(sd, buf, n) != n) {
				perror("write");
				exit(1);
			}
			continue;
		}
		if (FD_ISSET(sd,&rfds)) {
			n = read(sd, buf, sizeof(buf)-1);
			if (n < 0) {
				perror("read");
				exit(1);
			}
			buf[n]='\0';
			buf[sizeof(buf)-1]='\0';
			if (insist_write(1, buf, n) != n) {
				perror("write");
				exit(1);
			}
			continue;
		}
	}
	/*
	 * Let the remote know we're not going to write anything else.
	 * Try removing the shutdown() call and see what happens.
	 */
	printf("Program shutting down due to Ctrl+C\n");
	if (shutdown(sd, SHUT_WR) < 0) {
		perror("shutdown");
```

```
                exit(1);
        }
        return 0;
}
```

socket-server.c

```
/*
 * socket-server.c
 * Simple TCP/IP communication using sockets
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 */
#define MAX_CONN 3
#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 <errno.h>
#include <time.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"

/* Convert a buffer to upercase */
void toupper_buf(char *buf, size_t n)
{
        size_t i;

        for (i = 0; i < n; i++)
                buf[i] = toupper(buf[i]);
}

/* 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)
{
	time_t t = time(NULL);
	struct tm tm = *localtime(&t);

	fd_set rfds;
	int nfds,nconn,i,j;
	char buf[buff_size];
	char bufout[buff_size];
	char addrstr[INET_ADDRSTRLEN];
	int sd, newsd;
	int conns[MAX_CONN];
	ssize_t n;
	socklen_t len;
	struct sockaddr_in sa;
	len = sizeof(struct sockaddr_in);

	/* 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|SOCK_NONBLOCK, 0)) < 0) {
		perror("socket");
		exit(1);
	}
	fprintf(stderr, "Created TCP socket\n");

	/* Bind to a well-known port */
	memset(&sa, 0, sizeof(sa));
	memset(&conns, 0, sizeof(conns));
	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);
}
FD_ZERO(&rfds);
nfds=sd+1;
FD_SET(sd,&rfds);
nconn=MAX_CONN;
/* Loop forever, accept()ing connections */
fprintf(stderr, "Waiting for an incoming connection...\n");
while(1) {
        nfds=sd+1;
        FD_ZERO(&rfds);
        //remake fdset
        FD_SET(sd,&rfds);
        for(i=0;i<nconn;i++){
                if (conns[i]==0) continue;
                else FD_SET(conns[i],&rfds);
                if (nfds<(conns[i]+1)) nfds=conns[i]+1;
        }
        if (select(nfds,&rfds,NULL,NULL,NULL)==-1) {
                if (errno==EBADF) printf("Its a bad f d \n");
                perror("select");
                exit(1);
        }
        if (FD_ISSET(sd,&rfds)) {
        /* Accept an incoming connection */
                if ((newsd = accept(sd, (struct sockaddr *)&sa, &len)) < 0) {
                        if ((errno==EAGAIN)||(errno==EWOULDBLOCK)) continue;
                        perror("accept");
                        exit(1);
                }
                bzero(&sa,len);
                if (getpeername(newsd,(struct sockaddr *)&sa, &len)<0) {
                        perror("getpeername");
                        exit(1);
                }
                if (!inet_ntop(AF_INET, &sa.sin_addr, addrstr, sizeof(addrstr))) {
                        perror("could not format IP address");
                        exit(1);
                }
                for(i=0;i<nconn;i++) {
                        if (conns[i]==0) {conns[i]=newsd;break;}
                }
                if (i==nconn) {
                //too many connections, you cant connect now
```

```
                        strncpy(bufout,"Too many connected clients right now. Try
again later.\n",sizeof(bufout)-1);
                        bufout[sizeof(bufout)-1]='\0';
                        n=strlen(bufout);
                        if (insist_write(newsd, bufout, n) != n) {
                                perror("write to remote peer failed");
                                exit(1);
                        }
                        continue;
                }
                FD_SET(newsd,&rfds);
                if (newsd+1>nfds) nfds=newsd+1;
                t = time(NULL);
                tm = *localtime(&t);
                n=sprintf(bufout, "[%d-%d-%d]User from [%s:%d] has joined the
chat.\n",tm.tm_hour,tm.tm_min,tm.tm_sec,addrstr, ntohs(sa.sin_port));
                for(i=0;i<nconn;i++) {
                        if(conns[i]==0) continue;
                        newsd=conns[i];
                        if (insist_write(newsd, bufout, n) != n) {
                                perror("write to remote peer failed");
                                exit(1);
                        }
                }
                continue;
        }
        for (i=0;i<nconn;i++) {
                if (conns[i]==0) continue;
                newsd=conns[i];
                if (!FD_ISSET(newsd,&rfds)) continue;
                n = read(newsd, buf, sizeof(buf));
                if (n <= 0) {
                        if (n < 0)
                                perror("read from remote peer failed");
                        else
                        //      fprintf(stderr, "Peer went away\n");
                                bzero(&sa,len);
                                if (getpeername(newsd,(struct sockaddr *)&sa,
&len)<0) {
                                        perror("getpeername");
                                        exit(1);
                                }

                                if (!inet_ntop(AF_INET, &sa.sin_addr, addrstr,
sizeof(addrstr))) {
                                        perror("could not format IP address");
                                        exit(1);
                                }
                                t = time(NULL);
```

```
                                        tm = *localtime(&t);
                                        n=sprintf(bufout, "[%d-%d-%d]User from [%s:%d]
has left the chat.\n",tm.tm_hour,tm.tm_min,tm.tm_sec,addrstr, ntohs(sa.sin_port));

                                        for(j=0;j<nconn;j++) {
                                                if (conns[j]==0) continue;

                                                newsd=conns[j];
                                                if (insist_write(newsd, bufout, n) != n) {
                                                        perror("write to remote peer failed");
                                                        exit(1);
                                                }
                                        }
                                        newsd=conns[i];
                                        conns[i]=0;
                                        FD_CLR(newsd,&rfds);
                                        if (close(newsd) < 0)
                                                perror("close");
                                        newsd=-1;
                                        break;
                                }
                                buf[n]='\0';
                                bzero(&sa,len);
                                if (getpeername(newsd,(struct sockaddr *)&sa, &len)<0) {
                                        perror("getpeername");
                                        exit(1);
                                }
                                if (!inet_ntop(AF_INET, &sa.sin_addr, addrstr, sizeof(addrstr))) {
                                        perror("could not format IP address");
                                        exit(1);
                                }
                                t = time(NULL);
                                tm = *localtime(&t);
                                n=sprintf(bufout, "[%d-%d-%d][%s:%d] said:
%s",tm.tm_hour,tm.tm_min,tm.tm_sec,addrstr, ntohs(sa.sin_port),buf);
                                for(j=0;j<nconn;j++) {
                                        if (conns[j]==0) continue;
                                        newsd=conns[j];
                                        if (insist_write(newsd, bufout, n) != n) {
                                                perror("write to remote peer failed");
                                                exit(1);
                                        }
                                }
                                break;
                        }
                }
        /* This will never happen */
        return 1;
}
```

Ζητούμενο 2

Εδώ φαίνεται η λειτουργία του chat με κρυπτογράφηση:

```
user@snf-822055:~/ask3/oslab-FINAL/oslab-backup$ sudo tcpdump -ni lo -vvv -XXX
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
21:09:35.203886 IP (tos 0x0, ttl 64, id 11934, offset 0, flags [DF], proto TCP (6), length 308)
    127.0.0.1.60512 > 127.0.0.1.35001: Flags [P.], cksum 0xff28 (incorrect -> 0x0590), seq 632942053:632942309, ack 2575022499, win 359, options [nop,nop,TS val 302788456 ecr 302779236], length 256
        0x0000:  0000 0000 0000 0000 0000 0000 0800 4500  ..............E.
        0x0010:  0134 2e9e 4000 4006 0d24 7f00 0001 7f00  .4..@.@..$......
        0x0020:  0001 ec60 88b9 25b9 ede5 997b b9a3 8018  ...`..%....{....
        0x0030:  0167 ff28 0000 0101 080a 120c 2f68 120c  .g.(......../h..
        0x0040:  0b64 aaca eb63 dd02 7c23 4b4e 1e01 121e  .d...c..|#KN....
        0x0050:  7baa 0a87 4aee b41a 879f 2e13 d7be a744  {...J.........D
        0x0060:  5795 ebd2 5f6c 844f c084 a239 da88 77e4  W..._l.O...9..w.
        0x0070:  aa95 3948 2927 2d66 1429 bb31 215c 3cae  ..9H)'-f.).1!\<.
        0x0080:  b660 4d28 36d1 b9ad 9436 2d07 df3a 5cb9  .`M(6....6-..:\.
        0x0090:  c0ed b694 ae53 bdd7 4d3f 6bc0 7a47 183b  .....S..M?k.zG.;
        0x00a0:  cccl 3131 ecfc 62b4 2234 8e0f 3b5e 0efd  ..11..b."4..;^..
        0x00b0:  766c 135c 547b 8c33 ff34 0d53 fcbf c65e  vl.\T{.3.4.S...^
        0x00c0:  8e3b a5de 6435 fd03 874f ac0d 33f6 bf97  .;..d5...O..3...
        0x00d0:  f285 bb36 6348 07bc 5dd9 4ff2 5452 2d5b  ...6cH..].O.TR-[
        0x00e0:  0beb df20 733a 4b3a 368a 2cf7 10bb 9e53  ....s:K:6.,....S
        0x00f0:  52d1 101b 988e 0906 d909 ae60 a7cc 5013  R..........`..P.
        0x0100:  3208 c2b3 34f9 4465 4738 5283 80d8 12fc  2...4.DeG8R.....
        0x0110:  c3f5 eee5 1110 50d1 daeb c486 3bde 97c2  ......P.....;...
        0x0120:  90a2 bede a800 9735 7ad2 6002 b086 8a24  .......5z.`....$
        0x0130:  023c d5cd d322 2e25 36d3 0715 45df c121  .<...".%6...E..!
        0x0140:  0019                                     ..
21:09:35.204997 IP (tos 0x0, ttl 64, id 30176, offset 0, flags [DF], proto TCP (6), length 308)
    127.0.0.1.35001 > 127.0.0.1.60512: Flags [P.], cksum 0xff28 (incorrect -> 0x2968), seq 1:257, ack 256, win 359, options [nop,nop,TS val 302788456 ecr 302788456], length 256
        0x0000:  0000 0000 0000 0000 0000 0000 0800 4500  ..............E.
        0x0010:  0134 75e0 4000 4006 c5e1 7f00 0001 7f00  .4u.@.@.........
        0x0020:  0001 88b9 ec60 997b b9a3 25b9 eee5 8018  .....`.{..%.....
        0x0030:  0167 ff28 0000 0101 080a 120c 2f68 120c  .g.(......../h..
        0x0040:  2f68 9c8c 5b36 8a2d 44a6 2c11 73d8 792a  /h..[6.-D.,.s.y*
        0x0050:  965e 2b73 39e4 f9c0 0c05 d9cf e9d8 4e25  .^+s9........N%
        0x0060:  933b 343c 2bf4 0200 47b2 7c7f 9e81 8386  .;4<+...G.|.....
        0x0070:  28a8 25ce 30fa 629c 4598 c111 4b9d 52c7  (.%.0.b.E...K.R.
        0x0080:  1a43 4a33 2eb9 03ea a029 aabd 57b0 fe69  .CJ3.....)..W..i
        0x0090:  a2e3 ac9b 481f 83bc d7e0 3b3e 7961 ed6b  ....H.....;>ya.k
        0x00a0:  318f 376f e393 2b55 8459 db53 372d f071  1.7o..+U.Y.S7-.q
        0x00b0:  64f7 a31c 05e8 b2c5 c689 a232 0d1e c47c  d.........2...|
        0x00c0:  8fa2 3c15 1707 99f1 27ff 035f 0094 e50e  ..<.....'.._....
        0x00d0:  9c80 a291 c51f bcb6 f1ad f2a1 4454 4226  ...........DTB&
        0x00e0:  4683 f289 bbb2 a0ba 52df 2e37 9028 1c48  F.......R..7.(.H
        0x00f0:  f0f5 0399 6ed0 190b 60b2 bd9c cbfa e578  ....n...`......x
        0x0100:  255b a72d a4c0 fd36 d4e4 df3b c992 1561  %[.-...6...;...a
        0x0110:  a2f7 5ec8 338e 00d9 da0c 7853 46ef a32c  ..^.3.....xSF..,
        0x0120:  1a99 246c 5605 0174 5788 96bf 7564 e32a  ..$lV...tW...ud.*
        0x0130:  d71e 373d 964e 239e 1bca 82f0 b346 e47b  ..7=.N#......F.{
        0x0140:  b6d4                                     ..
21:09:35.205047 IP (tos 0x0, ttl 64, id 11935, offset 0, flags [DF], proto TCP (6), length 52)
    127.0.0.1.60512 > 127.0.0.1.35001: Flags [.], cksum 0xfe28 (incorrect -> 0x168b), seq 256, ack 257, win 367, options [nop,nop,TS val 302788456 ecr 302788456], length 0
        0x0000:  0000 0000 0000 0000 0000 0000 0800 4500  ..............E.
        0x0010:  0034 2e9f 4000 4006 0e23 7f00 0001 7f00  .4..@.@..#......
        0x0020:  0001 ec60 88b9 25b9 eee5 997b baa3 8010  ...`..%....{....
        0x0030:  016f fe28 0000 0101 080a 120c 2f68 120c  .o.(......../h..
        0x0040:  2f68                                     /h
```

Παρατηρούμε αμέσως ότι σε αντίθεση με το προηγούμενο ερώτημα δε φαίνεται στο περιεχόμενο του tcpdump (που έγινε με την εντολή tcpdump -ni lo -vvv -XXX) το περιεχόμενο των μηνυμάτων που ανταλλάσσονται. Επιπλέον, σημείωνουμε ότι και ο server όσο και όλοι οι client αποκωδικοποιούν και κρυπτογραφούν τα μηνύματα .
Οι τροποποιημένοι κώδικες που περιέχουν τη χρήση του device cryptodev επισυνάπτονται παρακάτω. Το device αυτό προσομοιώνει τη χρήση ενός accelerator στο hardware.

socket-server.c

```
/*
 * socket-server.c
 * Simple TCP/IP communication using sockets
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 */
#define MAX_CONN 3
#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 <time.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <sys/stat.h>
#include "cryptodev.h"
#include <sys/types.h>
#include <sys/socket.h>

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

#include "socket-common.h"

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

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;
}

void encrypt(unsigned  char *cipher, const unsigned char *plain,const int len, struct crypt_op *
cryp,int cfd){
        int i;
        unsigned char text[DATA_SIZE];
        for(i=0;i<DATA_SIZE;i++){
                if (i<len) text[i]=plain[i];
                else text[i]='\0';
        }
        cryp->src = text;
        cryp->dst = cipher;
```

```
        cryp->op = COP_ENCRYPT;
        if (ioctl(cfd,CIOCCRYPT,cryp)){
                perror("ioctl");
                exit(1);
        }
}

int decrypt (char *plain,const unsigned char *cipher, struct crypt_op * cryp,int cfd){
        int i;
        unsigned char text[DATA_SIZE];
        unsigned char cipher2[DATA_SIZE];
        for(i=0;i<DATA_SIZE;i++) cipher2[i]=cipher[i];
        cryp->dst = text;
        cryp->src = cipher2;
        cryp->op = COP_DECRYPT;
        if (ioctl(cfd,CIOCCRYPT,cryp)){
                perror("ioctl");
                exit(1);
        }
        for(i=0;i<buff_size;i++){
                if (text[i]!='\0') plain[i]=(char)text[i];
                else {plain[i]='\0';}
        }
        return i-1;
}



int main(void)
{
        time_t t = time(NULL);
        struct tm tm = *localtime(&t);
        int cfd,i;
        cfd = open("/dev/crypto", O_RDWR);
        if (cfd<0){
                perror("open");
                exit(1);
        }
        struct session_op sess;
        struct crypt_op cryp;
        struct {
                unsigned char   plaintext[DATA_SIZE],
                                ciphertext[DATA_SIZE],
                                iv[BLOCK_SIZE],
                                key[KEY_SIZE];
        } data;
```

```
    memset(&sess, 0, sizeof(sess));
memset(&cryp, 0, sizeof(cryp));
memset(&data.iv,0,BLOCK_SIZE);
memset(&data.key,1,KEY_SIZE);
sess.cipher = CRYPTO_AES_CBC;
sess.keylen = KEY_SIZE;
sess.key = data.key;

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

cryp.ses = sess.ses;
cryp.len = sizeof(data.plaintext);
    cryp.iv = data.iv;

    fd_set rfds;
    int nfds,nconn,j;
    char buf[buff_size];
    char bufout[buff_size];
    char addrstr[INET_ADDRSTRLEN];
    int sd, newsd;
    int conns[MAX_CONN];
    ssize_t n;
    socklen_t len;
    struct sockaddr_in sa;
    len = sizeof(struct sockaddr_in);

    /* 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|SOCK_NONBLOCK, 0)) < 0) {
            perror("socket");
            exit(1);
    }
    fprintf(stderr, "Created TCP socket\n");

    /* Bind to a well-known port */
    memset(&sa, 0, sizeof(sa));
    memset(&conns, 0, sizeof(conns));
    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);
}
FD_ZERO(&rfds);
nfds=sd+1;
FD_SET(sd,&rfds);
nconn=MAX_CONN;
/* Loop forever, accept()ing connections */
fprintf(stderr, "Waiting for an incoming connection...\n");
while(1) {
        nfds=sd+1;
        FD_ZERO(&rfds);
        //remake fdset
        FD_SET(sd,&rfds);
        for(i=0;i<nconn;i++){
                if (conns[i]==0) continue;
                else FD_SET(conns[i],&rfds);
                if (nfds<(conns[i]+1)) nfds=conns[i]+1;
        }
        if (select(nfds,&rfds,NULL,NULL,NULL)==-1) {
                perror("select");
                exit(1);
        }
        if (FD_ISSET(sd,&rfds)) {
        /* Accept an incoming connection */
                if ((newsd = accept(sd, (struct sockaddr *)&sa, &len)) < 0) {
                        if ((errno==EAGAIN)||(errno==EWOULDBLOCK)) continue;
                        perror("accept");
                        exit(1);
                }
                bzero(&sa,len);
                if (getpeername(newsd,(struct sockaddr *)&sa, &len)<0) {
                        perror("getpeername");
                        exit(1);
                }
                if (!inet_ntop(AF_INET, &sa.sin_addr, addrstr, sizeof(addrstr))) {
                        perror("could not format IP address");
                        exit(1);
                }
                for(i=0;i<nconn;i++) {
                        if (conns[i]==0) {conns[i]=newsd;break;}
                }
                if (i==nconn) {
                //too many connections, you cant connect now
```

```
                                strncpy(bufout,"Too many connected clients right now. Try
again later.\n",sizeof(bufout)-1);
                                bufout[sizeof(bufout)-1]='\0';
                                n=strlen(bufout);
                                encrypt(data.ciphertext,(unsigned char*)bufout,n,&cryp,cfd);
                                if (insist_write(newsd, data.ciphertext, DATA_SIZE) !=
DATA_SIZE) {
                                        perror("write to remote peer failed");
                                        exit(1);
                                }
                                continue;
                        }
                        FD_SET(newsd,&rfds);
                        if (newsd+1>nfds) nfds=newsd+1;
                        t = time(NULL);
                        tm = *localtime(&t);
                        n=sprintf(bufout, "[%d-%d-%d]User from [%s:%d] has joined the
chat.\n",tm.tm_hour,tm.tm_min,tm.tm_sec,addrstr, ntohs(sa.sin_port));
                        encrypt(data.ciphertext,(unsigned char*)bufout,n,&cryp,cfd);
                        for(i=0;i<nconn;i++) {
                                if(conns[i]==0) continue;
                                newsd=conns[i];
//          fprintf(stderr, "%u Peer went away\n",data.ciphertext[13]);

                                if (insist_write(newsd, data.ciphertext, DATA_SIZE) !=
DATA_SIZE) {
                                        perror("write to remote peer failed");
                                        exit(1);
                                }
                        }
                        continue;
                }
                for (i=0;i<nconn;i++) {
                        if (conns[i]==0) continue;
                        newsd=conns[i];
                        if (!FD_ISSET(newsd,&rfds)) continue;
                        n = read(newsd, data.ciphertext, DATA_SIZE);
                        if (n <= 0) {
                                if (n < 0)
                                        perror("read from remote peer failed");
                                else
                                //      fprintf(stderr, "Peer went away\n");
                                        bzero(&sa,len);
                                        if (getpeername(newsd,(struct sockaddr *)&sa,
&len)<0) {
                                                perror("getpeername");
                                                exit(1);
                                        }
```

```
                                        if (!inet_ntop(AF_INET, &sa.sin_addr, addrstr,
sizeof(addrstr))) {
                                                perror("could not format IP address");
                                                exit(1);
                                        }
                                        t = time(NULL);
                                        tm = *localtime(&t);
                                        n=sprintf(bufout, "[%d-%d-%d]User from [%s:%d]
has left the chat.\n",tm.tm_hour,tm.tm_min,tm.tm_sec,addrstr, ntohs(sa.sin_port));

                                        for(j=0;j<nconn;j++) {
                                                if (conns[j]==0) continue;

                                                newsd=conns[j];
                                                encrypt(data.ciphertext,(unsigned
char*)bufout,n,&cryp,cfd);
                                                if (insist_write(newsd, data.ciphertext,
DATA_SIZE) != DATA_SIZE) {
                                                        perror("write to remote peer failed");
                                                        exit(1);
                                                }
                                        }
                                        newsd=conns[i];
                                        conns[i]=0;
                                        FD_CLR(newsd,&rfds);
                                        if (close(newsd) < 0)
                                                perror("close");
                                        newsd=-1;
                                        break;
                                }
                                buf[n]='\0';
                                bzero(&sa,len);
                                if (getpeername(newsd,(struct sockaddr *)&sa, &len)<0) {
                                        perror("getpeername");
                                        exit(1);
                                }
                                if (!inet_ntop(AF_INET, &sa.sin_addr, addrstr, sizeof(addrstr))) {
                                        perror("could not format IP address");
                                        exit(1);
                                }
                                n=decrypt(buf,data.ciphertext,&cryp,cfd);
                                buf[n]='\0';

//                                      fprintf(stderr, "%s Peer went away\n",buf);

                                t = time(NULL);
                                tm = *localtime(&t);
                                n=sprintf(bufout, "[%d-%d-%d][%s:%d] said:
%s",tm.tm_hour,tm.tm_min,tm.tm_sec,addrstr, ntohs(sa.sin_port),buf);
```

```
                        for(j=0;j<nconn;j++){
                                if (conns[j]==0) continue;
                                newsd=conns[j];
                                encrypt(data.ciphertext,(unsigned char*)bufout,n,&cryp,cfd);
                                if (insist_write(newsd, data.ciphertext, DATA_SIZE) !=
DATA_SIZE){
                                        perror("write to remote peer failed");
                                        exit(1);
                                }
                        }
                        break;
                }
        }

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

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/stat.h>
#include "cryptodev.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"

#define DATA_SIZE 256
```

```
#define BLOCK_SIZE 16
#define KEY_SIZE 16
/* 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;
}

static int running;

static void handler (int signum)
{
        running=0;
}


int main(int argc, char *argv[])
{
        int i;
        int cfd;
        cfd = open("/dev/crypto", O_RDWR);
        if (cfd<0) {
                perror("open(/dev/cryptodev0)");
                return 1;
        }


        struct session_op sess;
        struct crypt_op cryp;
        struct sigaction sigact;
        struct {
                unsigned char   plaintext[DATA_SIZE],
                                ciphertext[DATA_SIZE],
                                iv[BLOCK_SIZE],
                                key[KEY_SIZE];
        } data;
        memset(&sess, 0, sizeof(sess));
```

```
memset(&cryp, 0, sizeof(cryp));
memset(&data.iv,0,sizeof(data.iv));
memset(&data.key,1,sizeof(data.key));
sess.cipher = CRYPTO_AES_CBC;
sess.keylen = KEY_SIZE;
sess.key = data.key;

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

cryp.ses = sess.ses;
cryp.iv = data.iv;
cryp.len = sizeof(data.plaintext);
running=1;
sigact.sa_handler=handler;
sigact.sa_flags=SA_RESTART;
sigaction(SIGINT,&sigact,NULL);
int sd, port;
ssize_t n;
char buf[buff_size];
char *hostname;
struct hostent *hp;
struct sockaddr_in sa;
fd_set rfds;

if (argc != 3) {
        fprintf(stderr, "Usage: %s hostname port\n", argv[0]);
        exit(1);
}
hostname = argv[1];
port = atoi(argv[2]); /* Needs better error checking */

/* 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");

        /* Be careful with buffer overruns, ensure NUL-termination */
        /* Say something... */
        FD_ZERO(&rfds);
        FD_SET(0,&rfds);
        FD_SET(sd,&rfds);

        while(running) {
                FD_ZERO(&rfds);
                FD_SET(0,&rfds);
                FD_SET(sd,&rfds);
                if (select(sd+1,&rfds,NULL,NULL,NULL)< 0) {
                        if(errno==EINTR) {
                                printf("\n Interrupted by Signal. Terminating connection...
\n");
                                continue;
                        }
                        else
                        perror("select");
                        exit(1);
                }
                if (FD_ISSET(0,&rfds)) {
                        n = read(0, buf, sizeof(buf)-1);

                        if (n < 0) {
                                perror("read");
                                exit(1);
                        }
                        buf[n]='\0';
                        buf[sizeof(buf)-1]='\0';
                        for(i=0;i<DATA_SIZE;i++){
                                if (i<n) data.plaintext[i]=buf[i];
                                else data.plaintext[i]='\0';
                        }
                        cryp.src = data.plaintext;
                        cryp.dst = data.ciphertext;
                        cryp.op = COP_ENCRYPT;
                        if (ioctl(cfd, CIOCCRYPT, &cryp)) {
                                perror("ioctl(CIOCCRYPT)");
                                exit(1);
                        }
```

```
                if (insist_write(sd, data.ciphertext, DATA_SIZE) != DATA_SIZE) {
                        perror("write");
                        exit(1);
                }
                continue;
        }
        if (FD_ISSET(sd,&rfds)) {
                n = read(sd, data.ciphertext, DATA_SIZE);
                if (n < 0) {
                        perror("read");
                        exit(1);
                }
                if (n<DATA_SIZE) continue;
                cryp.src = data.ciphertext;
                cryp.dst = data.plaintext;
                cryp.op = COP_DECRYPT;
                if (ioctl(cfd, CIOCCRYPT, &cryp)) {
                        perror("ioctl(CIOCCRYPT)");
                        exit(1);
                }
                for(i=0;i<DATA_SIZE;i++){
                        if (data.plaintext[i]!='\0') buf[i]=data.plaintext[i];
                        else {buf[i]='\0';break;}
                }
                n=i;
                buf[sizeof(buf)-1]='\0';
                if (insist_write(1, buf, n) != n) {
                        perror("write");
                        exit(1);
                }
                continue;
        }
}
/*
 * Let the remote know we're not going to write anything else.
 * Try removing the shutdown() call and see what happens.
 */
printf("Program shutting down due to Ctrl+C\n");
if (ioctl(cfd, CIOCFSESSION, &sess.ses)) {
        perror("ioctl(CIOCFSESSION)");
        exit(1);
}
if(close(cfd)<0) {
        perror("close(cfd)");
        exit(1);
}
if (shutdown(sd, SHUT_WR) < 0) {
        perror("shutdown");
```

```
                exit(1);
        }
        return 0;
}
```

Ζητούμενο 3

Αρχικά παρατίθεται ο κώδικας του backend, η λειτουργία του επικεντρώνεται στη χρήση virtqueue προκειμένου να πάρει τα δεδομένα που στέλνει το frontend του qemu. Θυμίζουμε ότι όλο το παρακάτω τρέχει σε χώρο χρήστη στο VM του ωκεανού. Η vq_handle_element καλείται κάθε φορά που προστίθεται κάτι στην ουρά.

virtio-crypto.c (BACKEND)

```
/*
 * Virtio Crypto Device
 *
 * Implementation of virtio-crypto qemu backend device.
 *
 * Dimitris Siakavaras <jimsiak@cslab.ece.ntua.gr>
 * Stefanos Gerangelos <sgerag@cslab.ece.ntua.gr>
 *
 */

#include <qemu/iov.h>
#include "hw/virtio/virtio-serial.h"
#include "hw/virtio/virtio-crypto.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <crypto/cryptodev.h>

static uint32_t get_features(VirtIODevice *vdev, uint32_t features)
{
	DEBUG_IN();
	return features;
}

static void get_config(VirtIODevice *vdev, uint8_t *config_data)
{
	DEBUG_IN();
}

static void set_config(VirtIODevice *vdev, const uint8_t *config_data)
{
	DEBUG_IN();
}

static void set_status(VirtIODevice *vdev, uint8_t status)
{
```

```
	DEBUG_IN();
}

static void vser_reset(VirtIODevice *vdev)
{
	DEBUG_IN();
}

static void vq_handle_output(VirtIODevice *vdev, VirtQueue *vq)
{
	VirtQueueElement elem;
	unsigned int *syscall_type;
	int *host_fd;
	int* ret;
	unsigned int* ioctl_cmd;
	DEBUG_IN();

	char output_str[100];
	if (!virtqueue_pop(vq, &elem)) {
		DEBUG("No item to pop from VQ :(");
		return;
	}

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

	if ((host_fd = (int *) malloc(sizeof(int))) == NULL) {
		perror("out of mem");
		exit(1);
	}
	syscall_type = elem.out_sg[0].iov_base;
	switch (*syscall_type) {
		case VIRTIO_CRYPTO_SYSCALL_TYPE_OPEN:
			DEBUG("VIRTIO_CRYPTO_SYSCALL_TYPE_OPEN");
			//host_fd = elem.in_sg[0].iov_base;
			*host_fd = open("/dev/crypto", O_RDWR);
			memcpy(elem.in_sg[0].iov_base, host_fd, sizeof(int));
			if (*host_fd < 0){
				DEBUG("I WAS UNABLE TO OPEN /dev/crypto");
				perror("open");
				return;
			}
			sprintf(output_str,"I WAS ABLE TO OPEN /dev/crypto returning
%d",*host_fd);
			DEBUG(output_str);
			break;

		case VIRTIO_CRYPTO_SYSCALL_TYPE_CLOSE:
			DEBUG("VIRTIO_CRYPTO_SYSCALL_TYPE_CLOSE");
			host_fd=elem.out_sg[1].iov_base;
```

```
        if (close(*host_fd) < 0){
                perror("close");
                return;
        }
        break;

case VIRTIO_CRYPTO_SYSCALL_TYPE_IOCTL:
        DEBUG("VIRTIO_CRYPTO_SYSCALL_TYPE_IOCTL");
//unsigned char *output_msg = elem.out_sg[1].iov_base;
//unsigned char *input_msg = elem.in_sg[0].iov_base;
//memcpy(input_msg, "Host: Welcome to the virtio World!", 35);
//printf("Guest says: %s\n", output_msg);
//printf("We say: %s\n", input_msg);
        host_fd = elem.out_sg[1].iov_base;
        ioctl_cmd = elem.out_sg[2].iov_base;
        sprintf(output_str,"I GOT IOCTL = %u", *ioctl_cmd);
        DEBUG(output_str);
        switch(*ioctl_cmd) {
                case CIOCGSESSION:
                        DEBUG("CIOCGSESSION");
                        struct session_op *session_op = elem.in_sg[0].iov_base;

                        unsigned char *session_key = elem.out_sg[3].iov_base;
                        ret=elem.in_sg[1].iov_base;
                        session_op->key = session_key;
                        if(ioctl(*host_fd,CIOCGSESSION,session_op)){
                                *ret = -1;
                                perror("ioctl");
                        }
                        else *ret = 0;
                        break;

                case CIOCFSESSION:
                        DEBUG("CIOCFSESSION");
                        int* ses_id = elem.out_sg[3].iov_base;
                        ret=elem.in_sg[0].iov_base;
                        if(ioctl(*host_fd,CIOCFSESSION,ses_id)) {
                                perror("ioctl");
                                *ret=-1;
                        }
                        else *ret=0;
                        break;

                case CIOCCRYPT:
                        DEBUG("CIOCRYPT");
                        struct crypt_op* crypt_op = elem.out_sg[3].iov_base;
                        unsigned char *src = elem.out_sg[4].iov_base;
                        unsigned char *iv = elem.out_sg[5].iov_base;
                        unsigned char *dst = elem.in_sg[0].iov_base;
```

```
                                        ret=elem.in_sg[1].iov_base;
                                        crypt_op->src=src;
                                        crypt_op->iv=iv;
                                        crypt_op->dst=dst;

                                        if(ioctl(*host_fd,CIOCCRYPT,crypt_op)) {
                                                perror("ioctl");
                                                *ret= -1;
                                        }
                                        else *ret=0;
                                        break;
                                default:
                                        DEBUG("Unrecognised ioctl");
                                        break;
                        }
                        break;
                default:
                        DEBUG("Unknown syscall_type");
                        break;
        }

        virtqueue_push(vq, &elem, 0);
}

static void virtio_crypto_realize(DeviceState *dev, Error **errp)
{
   VirtIODevice *vdev = VIRTIO_DEVICE(dev);

        DEBUG_IN();

   virtio_init(vdev, "virtio-crypto", 13, 0);
        virtio_add_queue(vdev, 128, vq_handle_output);
}

static void virtio_crypto_unrealize(DeviceState *dev, Error **errp)
{
        DEBUG_IN();
}

static Property virtio_crypto_properties[] = {
   DEFINE_PROP_END_OF_LIST(),
};

static void virtio_crypto_class_init(ObjectClass *klass, void *data)
{
   DeviceClass *dc = DEVICE_CLASS(klass);
   VirtioDeviceClass *k = VIRTIO_DEVICE_CLASS(klass);

        DEBUG_IN();
```

```
    dc->props = virtio_crypto_properties;
    set_bit(DEVICE_CATEGORY_INPUT, dc->categories);

    k->realize = virtio_crypto_realize;
    k->unrealize = virtio_crypto_unrealize;
    k->get_features = get_features;
    k->get_config = get_config;
    k->set_config = set_config;
    k->set_status = set_status;
    k->reset = vser_reset;
};

static const TypeInfo virtio_crypto_info = {
    .name          = TYPE_VIRTIO_CRYPTO,
    .parent        = TYPE_VIRTIO_DEVICE,
    .instance_size = sizeof(VirtCrypto),
    .class_init    = virtio_crypto_class_init,
};

static void virtio_crypto_register_types(void)
{
    type_register_static(&virtio_crypto_info);
}

type_init(virtio_crypto_register_types)
```

Παρακάτω έχουμε τον κώδικα του frontend ο ποίος πάλι τρέχει σε χώρο χρήστη για εμάς αλλά σε χώρο πυρήνα για το qemu. Πάλι γίνεται χρήση των virtqueues για να μεταφερθούν δεδομένα στο backend. Πρακτικά αποτελεί μια υλοποίηση ενός "εικονικού" driver που κάνει χρήση paravirtualization για να επιτελέσει τη λειτουργία του. Εδώ γίνεται χρήση spinlock για να αποφευχθεί η σύγκρουση μεταξύ διαφορετικών διεργασιών. Επιπλέον αναμένουμε την επεξεργασία των δεδομένων από τον host μέσω busy-wait (θα μπορούσε να υλοποιηθεί ασύγχρονα και με την has_data).

crypto-chrdev.c (FRONTEND)

```
/*
 * crypto-chrdev.c
 *
 * Implementation of character devices
 * for virtio-crypto device
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 * Dimitris Siakavaras <jimsiak@cslab.ece.ntua.gr>
 * Stefanos Gerangelos <sgerag@cslab.ece.ntua.gr>
 *
```

```
 */
#include <linux/cdev.h>
#include <linux/poll.h>
#include <linux/sched.h>
#include <linux/module.h>
#include <linux/wait.h>
#include <linux/virtio.h>
#include <linux/virtio_config.h>

#include "crypto.h"
#include "crypto-chrdev.h"
#include "debug.h"

#include "cryptodev.h"

/*
 * Global data
 */
struct cdev crypto_chrdev_cdev;

/**
 * Given the minor number of the inode return the crypto device
 * that owns that number.
 **/
static struct crypto_device *get_crypto_dev_by_minor(unsigned int minor)
{
	struct crypto_device *crdev;
	unsigned long flags;

	debug("Entering");

	spin_lock_irqsave(&crdrvdata.lock, flags);

	// this is the way iterators are implemented in kernel C
	list_for_each_entry(crdev, &crdrvdata.devs, list) {
		if (crdev->minor == minor)
			goto out;
	}
	crdev = NULL;

out:
	spin_unlock_irqrestore(&crdrvdata.lock, flags);

	debug("Leaving");
	return crdev;
}

/***********************************
 * Implementation of file operations
```

```
 * for the Crypto character device
 ***********************************/

static int crypto_chrdev_open(struct inode *inode, struct file *filp)
{
        int ret = 0;
        int err;
        unsigned int len, num_out, num_in;
        struct crypto_open_file *crof;
        struct crypto_device *crdev;
        unsigned int *syscall_type;
        int *host_fd;
        unsigned long flags;
        struct scatterlist syscall_type_sg, host_fd_sg, *sg[2];

        debug("Entering");

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

        crof=NULL;

        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;
        }

        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.
```

```
	 **/
	num_out = 0;
	num_in = 0;
	sg_init_one(&syscall_type_sg, syscall_type, sizeof(*syscall_type));
	sg[num_out++] = &syscall_type_sg;
	sg_init_one(&host_fd_sg, host_fd, sizeof(*host_fd));
	sg[num_out + num_in++] = &host_fd_sg;

	//giati spinlock??
	spin_lock_irqsave(&crdev->lock, flags);
	ret = virtqueue_add_sgs(crdev->vq, sg, num_out, num_in, &syscall_type_sg,
GFP_ATOMIC);
	if (ret < 0) {
		spin_unlock_irqrestore(&crdev->lock, flags);
		debug("Could not add buffers to the vq.");
		goto fail;
	}
	virtqueue_kick(crdev->vq);

	/**
	 * Wait for the host to process our data.
	 **/
	while (virtqueue_get_buf(crdev->vq, &len) == NULL); // busy-wait loop
	spin_unlock_irqrestore(&crdev->lock, flags);

	/* If host failed to open() return -ENODEV. */
	debug("Backend returned file descriptor %d", *host_fd);
	if (*host_fd < 0) ret = -ENODEV;
	crof->host_fd = *host_fd;

fail:
//crof MUST remain
//	kfree(crof);
	kfree(syscall_type);
	kfree(host_fd);
	debug("Leaving with ret = %d", ret);
	return ret;
}

static int crypto_chrdev_release(struct inode *inode, struct file *filp)
{
	int ret = 0;
	struct crypto_open_file *crof = filp->private_data;
	struct crypto_device *crdev = crof->crdev;
	unsigned int *syscall_type;
	unsigned int num_out, len;
	struct scatterlist syscall_type_sg, host_fd_sg, *sg[2];
	unsigned long flags;
	debug("Entering");
```

```
	syscall_type = kzalloc(sizeof(*syscall_type), GFP_KERNEL);
	*syscall_type = VIRTIO_CRYPTO_SYSCALL_CLOSE;

	/**
	 * Send data to the host.
	 **/
	num_out = 0;
	sg_init_one(&syscall_type_sg, syscall_type, sizeof(*syscall_type));
	sg[num_out++] = &syscall_type_sg;
	sg_init_one(&host_fd_sg, &crof->host_fd, sizeof(crof->host_fd));
	sg[num_out++] = &host_fd_sg;
	spin_lock_irqsave(&crdev->lock, flags);
	ret = virtqueue_add_sgs(crdev->vq, sg, num_out, 0, &syscall_type_sg, GFP_ATOMIC);
	if (ret < 0) {
		spin_unlock_irqrestore(&crdev->lock, flags);
		debug("Could not add buffers to the vq.");
		goto fail;
	}
	virtqueue_kick(crdev->vq);

	/**
	 * Wait for the host to process our data.
	 **/
	while (virtqueue_get_buf(crdev->vq, &len) == NULL); // busy-wait loop
	spin_unlock_irqrestore(&crdev->lock, flags);
fail:
	kfree(crof);
	kfree(syscall_type);
	debug("Leaving");
	return ret;
}

static long crypto_chrdev_ioctl(struct file *filp, unsigned int cmd,
				unsigned long arg)
{
	long ret = 0;
	int err;
	int *host_ret;
	uint32_t *ses_id;
	struct session_op *sess;
	struct crypt_op *cryp;
	struct crypto_open_file *crof = filp->private_data;
	struct crypto_device *crdev = crof->crdev;
	struct virtqueue *vq = crdev->vq;
	struct scatterlist sess_id_sg,syscall_type_sg, cmd_sg, session_sg, host_fd_sg, ret_sg,
ses_id_sg,
		cryp_src_sg, cryp_dst_sg, cryp_iv_sg, cryp_op_sg, seskey_sg,
		*sgs[8];
```

```
        unsigned int num_out, num_in, len, *cmd_ptr;
        unsigned long flags;
        unsigned char *ses_key, *src, *dst=NULL, *iv;
//# define MSG_LEN 100
//      unsigned char *output_msg, *input_msg;
        unsigned int *syscall_type;

        printk(KERN_CRIT "Entering");

        /**
         * Allocate all data that will be sent to the host.
         **/
//      output_msg = kzalloc(MSG_LEN, GFP_KERNEL);
//      input_msg = kzalloc(MSG_LEN, GFP_KERNEL);
        syscall_type = kzalloc(sizeof(*syscall_type), GFP_KERNEL);
        *syscall_type = VIRTIO_CRYPTO_SYSCALL_IOCTL;
        host_ret = kzalloc(sizeof(*host_ret), GFP_KERNEL);
        cmd_ptr = kzalloc(sizeof(*cmd_ptr), GFP_KERNEL);
        ses_id = kzalloc(sizeof(*ses_id), GFP_KERNEL);
        src=NULL;
        dst=NULL;
        iv=NULL;
        ses_key=NULL;
        *cmd_ptr = cmd;

        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(&host_fd_sg,&crof->host_fd,sizeof(crof->host_fd));
        sgs[num_out++]=&host_fd_sg;

        sess = kzalloc(sizeof(*sess), GFP_KERNEL);
        if (!sess) {
                return -ENOMEM;
        }

        cryp = kzalloc(sizeof(*cryp), GFP_KERNEL);
        if (!cryp) {
                return -ENOMEM;
        }


/* ?? */
```

```
        /**
         * Add all the cmd specific sg lists.
         **/
        switch (cmd) {
        case CIOCGSESSION:
                debug("CIOCGSESSION");
//              memcpy(output_msg, "Hello HOST from ioctl CIOCGSESSION.", 36);
//              input_msg[0] = '\0';
                sg_init_one(&cmd_sg, cmd_ptr, sizeof(*cmd_ptr));
                sgs[num_out++] = &cmd_sg;
                if (copy_from_user(sess, (struct session_op*) arg, sizeof(struct session_op))){
                        debug("copy_from_user");
                        return -1;
                }
                ses_key = kzalloc(sess->keylen*sizeof(char), GFP_KERNEL);
                if (!ses_key) {
                        return -ENOMEM;
                }
                if(copy_from_user(ses_key, sess->key, sizeof(char)*sess->keylen)){
                        debug("copy_from_user");
                        return -1;
                }
                sg_init_one(&seskey_sg, ses_key, sizeof(char)*sess->keylen);
                sgs[num_out++] = &seskey_sg;

                sg_init_one(&session_sg, sess, sizeof(*sess));
                sgs[num_out + num_in++] = &session_sg;

                sg_init_one(&ret_sg, host_ret, sizeof(*host_ret));
                sgs[num_out + num_in++] = &ret_sg;

                break;

        case CIOCFSESSION:
                debug("CIOCFSESSION");
//              memcpy(output_msg, "Hello HOST from ioctl CIOCFSESSION.", 36);
///             input_msg[0] = '\0';
                sg_init_one(&cmd_sg, cmd_ptr, sizeof(*cmd_ptr));
                sgs[num_out++] = &cmd_sg;

                if(copy_from_user(ses_id, (uint32_t*)arg, sizeof(*ses_id))){
                        debug("copy_from_user");
                        return -1;
                }
                sg_init_one(&sess_id_sg, ses_id, sizeof(*ses_id));
                sgs[num_out++] = &sess_id_sg;
                sg_init_one(&ret_sg, host_ret, sizeof(host_ret));
                sgs[num_out + num_in++] = &ret_sg;
```

```
                break;

        case CIOCCRYPT:
                debug("CIOCCRYPT");
//              memcpy(output_msg, "Hello HOST from ioctl CIOCCRYPT.", 33);
//              input_msg[0] = '\0';
                sg_init_one(&cmd_sg, cmd_ptr, sizeof(*cmd_ptr));
                sgs[num_out++] = &cmd_sg;
                if(copy_from_user(cryp , (struct crypt_op*)arg, sizeof( struct crypt_op))){
                        debug("copy_from_user");
                        return -1;
                }

                sg_init_one(&cryp_op_sg, cryp, sizeof(*cryp));
                sgs[num_out++] = &cryp_op_sg;

                src = kzalloc(cryp->len*sizeof(char), GFP_KERNEL);
                if (!src) {
                        return -ENOMEM;
                }

                if(copy_from_user(src, cryp->src, cryp->len*sizeof(char))){
                        debug("copy_from_user");
                        return -1;
                }

                sg_init_one(&cryp_src_sg, src, cryp->len*sizeof(char));
                sgs[num_out++] = &cryp_src_sg;

                iv = kzalloc(16*sizeof(char), GFP_KERNEL);
                if (!iv) {
                        return -ENOMEM;
                }

                if(copy_from_user(iv, cryp->iv, 16*sizeof(char))){
                        debug("copy_from_user");
                        return -1;
                }

                sg_init_one(&cryp_iv_sg, iv, cryp->len*sizeof(char));
                sgs[num_out++] = &cryp_iv_sg;


                dst = kzalloc(cryp->len*sizeof(char), GFP_KERNEL);
                if (!dst) {
                        return -ENOMEM;
                }

                sg_init_one(&cryp_dst_sg, dst, cryp->len*sizeof(char));
```

```
                sgs[num_out + num_in++] = &cryp_dst_sg;

                sg_init_one(&ret_sg, host_ret, sizeof(host_ret));
                sgs[num_out + num_in++] = &ret_sg;
                break;

        default:
                debug("Unsupported ioctl command");

                break;
        }


        /**
         * Wait for the host to process our data.
         **/
        /* ?? */
        /* ?? Lock ?? */
        spin_lock_irqsave(&crdev->lock, flags);

        err = virtqueue_add_sgs(vq, sgs, num_out, num_in, &syscall_type_sg, GFP_ATOMIC);
        if (err < 0) {
            spin_unlock_irqrestore(&crdev->lock, flags);
            debug("Could not add buffers to the vq.");
            return -EINVAL;
        }
        printk(KERN_CRIT "about to notify backend\n");
        virtqueue_kick(vq);
        printk(KERN_CRIT "backend has been notified");
        while (virtqueue_get_buf(vq, &len) == NULL)
                /* do nothing */;
        printk(KERN_CRIT "backend has sent us data");
        spin_unlock_irqrestore(&crdev->lock,flags);


//      debug("We said: '%s'", output_msg);
//      debug("Host answered: '%s'", input_msg);

//      kfree(output_msg);
//      kfree(input_msg);
        switch(cmd){
        case CIOCGSESSION:
                debug("CIOCGSESSION");
                if((*host_ret<0)|| (copy_to_user((struct session_op*)arg, sess,sizeof(struct
session_op))){
                        debug("CIOCGSESSION");
                        return -1;
                }
                break;
```

```
	case CIOCFSESSION:
		debug("CIOCFSESSION");
		if((*host_ret<0)){
			debug("CIOCFSESSION");
			return -1;
		}
		break;
	case CIOCCRYPT:
		debug("CIOCCRYPT");
		if((*host_ret<0)||(copy_to_user(((struct crypt_op*) arg)->dst, dst, cryp-
>len*sizeof(char))))){
			debug("CIOCCRYPT, with %d", err);
			return -1;
		}
		break;
	}

	kfree(syscall_type);
	kfree(host_ret);
	kfree(cmd_ptr);
	kfree(ses_id);
	kfree(sess);
	kfree(cryp);
	kfree(ses_key);
	kfree(src);
	kfree(dst);
	kfree(iv);
	debug("Leaving");

	return *host_ret;
}

static ssize_t crypto_chrdev_read(struct file *filp, char __user *usrbuf,
		size_t cnt, loff_t *f_pos)
{
	debug("Entering");
	debug("Leaving");
	return -EINVAL;
}

static struct file_operations crypto_chrdev_fops =
{
	.owner          = THIS_MODULE,
	.open           = crypto_chrdev_open,
	.release        = crypto_chrdev_release,
	.read           = crypto_chrdev_read,
	.unlocked_ioctl = crypto_chrdev_ioctl,
};
```

```
int crypto_chrdev_init(void)
{
	int ret;
	dev_t dev_no;
	unsigned int crypto_minor_cnt = CRYPTO_NR_DEVICES;

	debug("Initializing character device...");
	cdev_init(&crypto_chrdev_cdev, &crypto_chrdev_fops);
	crypto_chrdev_cdev.owner = THIS_MODULE;

	dev_no = MKDEV(CRYPTO_CHRDEV_MAJOR, 0);
	ret = register_chrdev_region(dev_no, crypto_minor_cnt, "crypto_devs");
	if (ret < 0) {
		debug("failed to register region, ret = %d", ret);
		goto out;
	}
	ret = cdev_add(&crypto_chrdev_cdev, dev_no, crypto_minor_cnt);
	if (ret < 0) {
		debug("failed to add character device");
		goto out_with_chrdev_region;
	}

	debug("Completed successfully");
	return 0;

out_with_chrdev_region:
	unregister_chrdev_region(dev_no, crypto_minor_cnt);
out:
	return ret;
}

void crypto_chrdev_destroy(void)
{
	dev_t dev_no;
	unsigned int crypto_minor_cnt = CRYPTO_NR_DEVICES;

	debug("entering");
	dev_no = MKDEV(CRYPTO_CHRDEV_MAJOR, 0);
	cdev_del(&crypto_chrdev_cdev);
	unregister_chrdev_region(dev_no, crypto_minor_cnt);
	debug("leaving");
}
```

Τέλος έχουμε το crypto-module.c που απλά αρχικοποιεί ένα spinlock.

crypto-module.c

```
#include <linux/sched.h>
```

```
#include <linux/slab.h>
#include <linux/module.h>
#include <linux/spinlock.h>
#include <linux/virtio.h>
#include <linux/virtio_config.h>

#include "crypto.h"
#include "crypto-chrdev.h"
#include "debug.h"

struct crypto_driver_data crdrvdata;

static void vq_has_data(struct virtqueue *vq)
{
	debug("Entering");
	debug("Leaving");
}

static struct virtqueue *find_vq(struct virtio_device *vdev)
{
	int err;
	struct virtqueue *vq;

	debug("Entering");

	vq = virtio_find_single_vq(vdev, vq_has_data, "crypto-vq");
	if (IS_ERR(vq)) {
		debug("Could not find vq");
		vq = NULL;
	}

	debug("Leaving");

	return vq;
}

/**
 * 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. */
	spin_lock_init(&crdev->lock);

	/**
	 * 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;
}

static void virtcons_remove(struct virtio_device *vdev)
{
	struct crypto_device *crdev = vdev->priv;

	debug("Entering");

	/* Delete virtio device list entry. */
	spin_lock_irq(&crdrvdata.lock);
	list_del(&crdev->list);
	spin_unlock_irq(&crdrvdata.lock);

	/* NEVER forget to reset virtio device and delete device virtqueues. */
	vdev->config->reset(vdev);
	vdev->config->del_vqs(vdev);

	kfree(crdev);

	debug("Leaving");
```

```
}

static struct virtio_device_id id_table[] = {
	{VIRTIO_ID_CRYPTO, VIRTIO_DEV_ANY_ID },
	{ 0 },
};

static unsigned int features[] = {
	0
};

static struct virtio_driver virtio_crypto = {
	.feature_table = features,
	.feature_table_size = ARRAY_SIZE(features),
	.driver.name =  KBUILD_MODNAME,
	.driver.owner = THIS_MODULE,
	.id_table =	id_table,
	.probe =	virtcons_probe,
	.remove =	virtcons_remove,
};

/**
 * The function that is called when our module is being inserted in
 * the running kernel.
 **/
static int __init init(void)
{
	int ret = 0;
	debug("Entering");

	/* Register the character devices that we will use. */
	ret = crypto_chrdev_init();
	if (ret < 0) {
		printk(KERN_ALERT "Could not initialize character devices.\n");
		goto out;
	}

	INIT_LIST_HEAD(&crdrvdata.devs);
	spin_lock_init(&crdrvdata.lock);

	/* Register the virtio driver. */
	ret = register_virtio_driver(&virtio_crypto);
	if (ret < 0) {
		printk(KERN_ALERT "Failed to register virtio driver.\n");
		goto out_with_chrdev;
	}

	debug("Leaving");
	return ret;
```

```
out_with_chrdev:
        debug("Leaving");
        crypto_chrdev_destroy();
out:
        return ret;
}

/**
 * The function that is called when our module is being removed.
 * Make sure to cleanup everything.
 **/
static void __exit fini(void)
{
        debug("Entering");
        crypto_chrdev_destroy();
        unregister_virtio_driver(&virtio_crypto);
        debug("Leaving");
}

module_init(init);
module_exit(fini);

MODULE_DEVICE_TABLE(virtio, id_table);
MODULE_DESCRIPTION("Virtio crypto driver");
MODULE_LICENSE("GPL");
```