Avtentikacija na podlagi zgoščevalnih funkcij

Črtomir Gorup

2. marec 2010

Kazalo

Poglavje 1

Uvod

Avtentikacija je proces v katerem za določen predmet ali osebo vzpostavimo ali potrdimo njegovo avtentičnost. Slednja zagotavlja, da so trditve o predmetu ali izjave osebe veljavne. Zanimanje za avtentične predmete sega v čas pred našim štetjem, ko se je uveljavila uporaba denarja. Od takrat naprej posamezniki ponarejajo denar, umetnine, podpise, pogodbe, čeke, dokumente, fotografije in zdravila. V dobi interneta je visoka tehnologija dostopna skoraj vsakomur in ponarejanje, še posebej informacij, je enostavno.

Trendom ponarejanja hitro sledijo tudi metode avtentikacije. Dandanes se z njo srečamo srečamo na vsakem koraku. Ob dvigu gotovine se v banki avtenticiramo z veljavnim osebnim dokumentom, hologramska nalepka na škatli zdravil povečuje naše zaupanje v njihovo avtentičnost. Potreba po avtentikaciji na različnih področjih ima za posledico veliko različnih načinov preverjanja avtentičnosti. Arheologi tako za določanje starosti predmetov analizirajo atome ogljika, bančni uslužbenci veljavne bankovce prepoznajo po vgrajenih varnostnih mehanizmih.

V seminarski nalogi se bom osredotočil na metode za avtentikacijo informacij. V prvem delu bom predstavil avtentikacijsko shemo, ki za razliko od ostalih avtentičnost zagotavlja s pomočjo zgoščevalnih funkcij. V nadaljevanju bom opisal možno uporabo avtentikacijske sheme na primeru industrijskega krmilnika scenske razsvetljave. S pomočjo obstoječih kriptografskih knjižnic bom implementiral avtentikacijo prejetih sporočil na lastnem industrijskem krmilniku scenske razsvetljave.

1.1 Avtentikacija sporočil

Potrebo po avtentikaciji preko nezavarovanega kanala prejetih sporočil srečamo pri veliko komunikacijskih protokolih (IPSec, OpenSSH, TSL). Za preverjanje avtentičnosti na najvišjem nivoju uporabljamo tehniko digitalnega podpisa, ki nam zagotavlja, da sporočilo ni bilo podpisano od nobenega

drugega pošiljatelja. Če se odrečemo nekaterim varnostnim zahtevam (npr. ločena sposobnost preverjanja avtentičnosti in generiranja avtentičnih sporočil) lahko za avtentikacijo uporabimo katero izmed simetričnih metod.

Večina takih mehanizmov temelji na tajnem ključu, ki je poznan le pošiljatelju in prejemniku, in algoritmu MAC (angl. Message Authentication Code). S slednjim pošiljatelj, na podlagi sporočila in tajnega ključa, izračuna avtentikacijsko značko (angl. authentication tag), ki jo skupaj s sporočilom pošlje prejemniku. Prejemnik za prejeto sporočilo ponovi izračun avtentikacijske značke, ter jo primerja s prejeto. V primeru ujemanja lahko prejemnik z visoko verjetnostjo trdi, da je sporočilo veljavno, ter poslano od resničnega pošiljatelja. V seminarski nalogi se bom osredotočil na algoritme HMAC (angl. Hash-based Message Authentication Code), ki varnostne značke računajo s pomočjo zgoščevalnih funkcij.

1.2 Pregled področja

Mehanizmi MAC so do leta 1995 večinoma uporabljali bločne šifre (npr. DES) [1]. Čeprav so se zgoščevalne funkcije tipično uporabljale za varno shranjevanje gesel, leta 1995 prispevek na konverenci Crypto 96' z naslovom '*Keying Hash Functions for Message Authentication*' [6]. Slednji opisuje dve novi shemi za avtentikacijo sporočil NMAC (angl. Nested Message Authentication Code) in HMAC (angl. Hash Based Authentication Code). V praktični uporabi je bolj zaživel mehanizem HMAC. Slednji je tudi predstavljen v dokumentu RFC2104, leta 2002 tudi standardiziran s strani ameriške vlade (dokument FIPS 198 [7]).

1.3 Struktura seminarske naloge

Seminarska naloga je razdeljena na dva dela. Prvi je teoretičen in v njem bom predstavil osnove zgoščevalnih funkcij ter avtentikacijsko shemo HMAC. V drugem delu bom najprej opisal problem oddaljenega krmiljenja scenske razsvetljave ter rešitev z uporabo avtentikacijske sheme HMAC. Predstavljeno rešitev bom implementiral na 8-bitnem mikroprocesorju ter podal primer delovanja. V zaključku bom podal nekaj idej za še dodatno izboljšanje varnosti. Razvita programska koda za dostop do kriptografske knjižnice in avtentikacijski mehanizem se nahajajo v prilogi.

Poglavje 2

Teoretični del

Za lažje razumevanje mehanizma HMAC bom v tem poglavju najprej naredil kratek pregled zgoščevalnih funkcij in njihovih varnostnih lastnosti. V nadaljevanju poglavja bom opisal mehanizem MAC ter podrobneje predstavil njegovo različico HMAC, ki za zagotavljanje varnosti uporablja zgoščevalne funkcije.

2.1 Zgoščevalne funkcije

Zgoščevalno funkcijo lahko formalno zapišemo s spodnjo enačbo:

$$h : \{0,1\}^* \rightarrow \{0,1\}^n$$

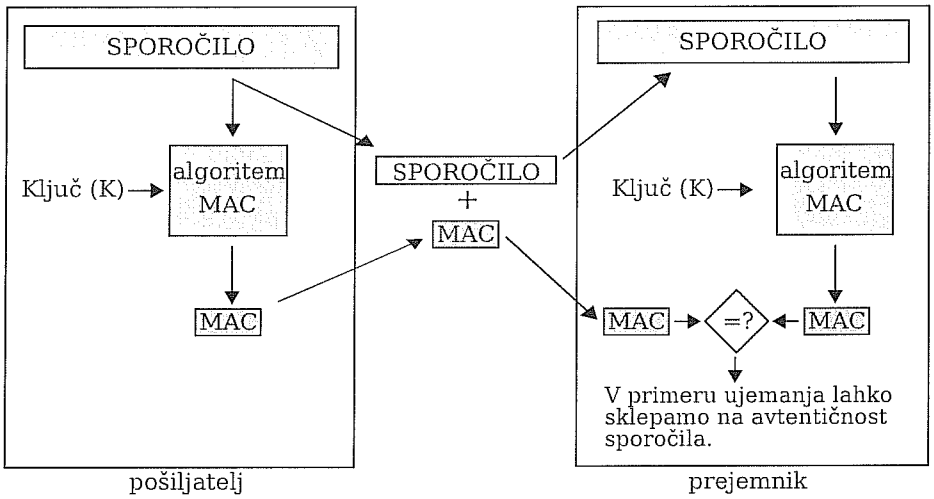
Varno zgoščevalno funkcijo h odlikujejo naslednje lastnosti:

1. Za vsako sporočilo m je izračun $h(m)$ enostaven.
2. Če je $h(m)$ podan potem ni časovno sprejemljive (boljše od napada z grobo silo) metode za iskanje sporočila m.
3. Računsko je nemogoče določiti par m in m', da velja $h(m) = h(m')$.
4. Vsaka zgostitev naj ima približno 50% bitov 1.
5. Če se vhodni sporočili m in m' razlikujeta samo za en bit, potem naj $h(m)$ in $h(m')$ izgledata kot neodvisno izbrani števili.

2.2 Mehanizem MAC

Funkcijo MAC lahko definiramo kot funkcijo, ki na podlagi tajnega ključa k za dano sporočilo m_i izračuna njegovo varnostno značko $MAC_i(m_i)$. Mehanizem MAC se uporablja za avtentikacijo sporočil, pri čemer morata pošiljatelj in prejemnik predhodno uskladiti tajni ključ k.

Slika 2.1 prikazuje shemo delovanja mehanizma MAC. Pošiljatelj za sestavljeno sporočilo s pomočjo algoritma MAC in tajnega ključa k izračuna avtentikacijsko značko, ki jo skupaj s sporočilom pošlje prejemniku. Prejemnik za prejeto sporočilo ponovno izračuna avtentikacijsko značko in jo primerja s prejeto. V primeru ujemanja lahko z veliko verjetnostjo sklepa na avtentičnost sporočila.



Slika 2.1: Shema delovanja mehanizma MAC za primer avtentikacije prejetih sporočil.

2.3 Mehanizem HMAC

Razlogov za uporabo zgoščevalnih funkcij v mehanizmih MAC je več. Zgoščevalne funkcije so v programski implementaciji hitrejše od bločnih šifer, njihove implementacije so prosto dostopne, same funkcije pa niso podvržene izvoznim omejitvam nekaterih držav.

Zgoščevalne funkcije v osnovi niso namenjene avtentikaciji sporočil, zato nimajo možnosti uporabe tajnega ključa. Bistvo konstrukcije HMAC se zato skriva v integraciji tajnega ključa k v mehanizem MAC, ki uporablja zgoščevalne funkcije.
Izračun varnostne značke za avtentikacijo HMAC opravimo po spodnji formuli.

$$\mathrm{HMAC}(m,k) = h((\bar{k} \oplus \mathrm{opad}) \mid h((\bar{k} \oplus \mathrm{ipad}) \mid m))$$

Pri tem je:

m	sporočilo za katerega računamo avtentikacijsko značko,
$\bar{k}$	tajni ključ dopolnjen z ničlami do dolžine bloka iteracijske zgoščevalne funkcije,
h	zgoščevalna funkcija,
$\oplus$	operator XOR,
opad	konstantna vrednost $0x36$ dolžine bloka iteracijske zgoščevalne funkcije,
ipad	konstantna vrednost $0x5c$ dolžine bloka iteracijske zgoščevalne funkcije,
$\|$	operator za konkatenacijo.

Konstranti opad in ipad bistveno ne pripomoreta k varnosti avtentikacijske sheme. Njihovi vrednosti sta izbrani tako, da je Hammingova razdalja med notranjo in zunanjo konkatenacijo tajnega ključa največja.

Večina sistemov za avtentikacijo HMAC uporablja zgoščevalno funkcijo MD5 ali SHA-1. Znane metode za iskanje trkov pri MD5 in SHA-1 zgoščevalnih funkcijah ne vplivajo na varnost mehanizma HMAC [2].

Poglavje 3

Praktični del

V tem poglavju bom predstavil uporabo mehanizma HMAC na primeru avtentikacije sporočil poslanih na krmilnik za upravljanje scenske razsvetljave. V uvodu bom najprej opisal osnove krmiljena scenske razsvetljave ter protokola DMX. V nadaljevanju bom pritedil mehanizem HMAC za konkretni primer, ter implementiral avtentikacijo sporočil na 8-bitnem mikroprocesorju ATMega328.

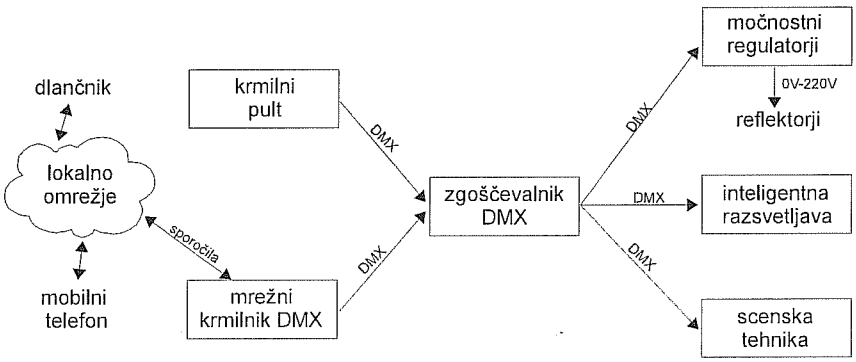
3.1 Krmiljenje scenske razsvetljave

Večina gledališč in televizijskih studiev upravlja s scensko razsvetljavo in odrsko tehniko digitalno, preko namenskega krmilnega pulta ali računalnika s primerno stojno in programsko opremo. Krmilne informacije se od izvora (krmilni pult, računalnik) prenašajo do ponora (močnostni regulatorji, inteligentne svetilke, digitalno krmiljena odrska tehnika) prenašajo preko protokola DMX512.

Protokol DMX512 je enostaven standardiziran asinhron serijski protokol namenjen komunikaciji med krmilno opremo in gledališko in studijsko tehniko. Osnovni podatkovni okvir je sestavljen iz 512 blokov (kanalov) dolžine 8 bitov. Vsak blok pripada eni ali več napravam, ki so priključene na serijsko vodilo DMX. Naprave imajo nastavljiv lasten naslov dolžine 9 bitov s katerim skrbnik sistema nastavi številko prvega bloka na spremembo katerega se naprave odzivajo.

Na sliki 3.1 je predstavljena tipična shema sistema scenske razsvetljave, kot jo srečamo v gledališčih in televizijskih studijih. Iz opisa protokola v prejšnjem odstavku je razvidno, da lahko na enem vodilu napravam pošiljamo do 512 bajtov podatkov. Če so naprave enostavne (npr. reflektorji) to pomeni, da lahko na enem vodilu DMX reguliramo svetilnost do 512 različnim reflektorjem. Če bo naprave naprednejše (npr. robotizirane svetilke) in zasedajo več kanalov, se njihovo število ustrezno zmanjša.

Iz slike 3.1 je razvidno, da je poleg krmilega pulta za upravljanje s siste-



Slika 3.1: Poenostavljena shema sistema scenske razsvetljave na podlagi protokola DMX512.

mom možno uporabiti tudi brezžično napravo (npr. dlančnik, mobilni telefon) povezano v lokalno računalniško omrežje. V tem primeru je potreben mrežni krmilnik DMX. Slednji mora omogočati vzpostavitev TCP/IP seje z brezžično napravo ter na podlagi prejetih avtentičnih sporočil generirati signal DMX.

3.2 Opis problema

V našem primeru je mrežni krmilnik DMX iz računalniškega vidika precej omejena naprava, ki ne omogoča šifrirane TCP/IP seje v realnem času. Kljub omejenim resursom je na njem potrebno zagotoviti mehanizem preverjanja avtentičnosti prejetih sporočil[1]. Oddaljeno upravljanje s scensko razsvetljavo želimo omogočiti le pooblaščenemu osebju, ki ima dostop do brezžične naprave z originalno krmilno programsko opremo. Nepooblaščen dostop do mrežnega krmilnika DMX je lahko nevaren (krmiljene scenske tehnike), lahko povzroči tudi materialno škodo (npr. preobremenitev sistema).

3.3 Avtentikacijska shema

Avtentikacijo sporočil HMAC sem za primer mrežnega krmilnika DMX razširil z uporabo inicializacijskega vektorja IV. Slednji se uporablja za izračun ključa za avtentikacijo HMAC. Avtentikacija v mojem primeru poteka v dveh korakih.

[1]Avtentična sporočila so tista, ki so bila poslana s programsko opremo priloženo mrežnemu krmilniku DMX.

I Ob vzpostavitvi seje mrežni krmilnik DMX naključno izbere inicializacijski vektor IV, ter ga pošlje krmilni aplikaciji uporabnika.

II Mrežni krmilnik ter aplikacija s pomočjo IV, prednastavljenega tajnega ključa $key1$, mehanizma HMAC ter formule $key = \mathrm{HMAC}(IV, key1)$ izračunata nov ključ key, ki se bo v trenutni seji uporabljal za avtentikacijo HMAC.

Po izračunu ključa na obeh straneh komunikacijskega kanala je krmilna aplikacija za poslana sporočila sposobna izračunati avtentikacijsko značko, mrežni krmilnik DMX pa jo je sposoben tudi preveriti. Pošiljanje avtentičnih sporočil poteka v naslednjih korakih:

I Na strani aplikacije se ukaz uporabnika c se pretvori v sporočilo m, za katerega se po formuli $a = \mathrm{HMAC}(m, key)$ izračuna avtentikacijska značka a. Sporočilo m se skupaj z avtorizacijsko značko a pošlje mrežnemu krmilniku DMX.

II Mrežni krmilnik za prejeto sporočilo po enakem postopku ponovi izračun avtentikacijske značke. V primeru ujemanja prejete in ponovno izračunane avtentikacijske značke lahko krmilnik z veliko verjetnostjo sklepa na avtentično sporočilo in izvede prejeti ukaz.

3.4 Opis platforme

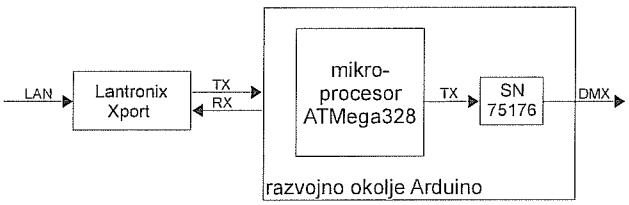
Arduino [5] je prosto dostopen projekt mikroprocesorskega okolja zasnovanega na procesorjih proizvajalca Atmel. Njegovo jedro predstavlja 8-bitni RISC mikroprocesor Atmel ATmega328P [3]. V čipu je poleg CPE integriran tudi pomnilnik Flash velikosti 32Kb, pomnilnik EEPROM velikosti 1Kb in delovni pomnilnik SRAM velikosti 2048 bajtov. Procesor teče pri frekvenci 16MHz. Binarna programska koda je shranjena v pomnilniku Flash, konstante pa v pomnilniku EEPROM. Procesor nima nobene strojne podpore za kriptiranje.

Generiranje signala DMX zagotavlja, preko digitalnega izhoda procesorja povezan, linijski gonilnik SN75176.

Za komunikacijo LAN skrbi, preko serijskih vrat povezan, dodatni vmesnik Lantronix XPort. Slednji deluje kot TCP/IP sklad in je potreben za vzpostavitev seje TCP/IP s krmilno aplikacijo uporabnika.

3.5 Implementancija

Implementacijo na platformi Arduino sem si poenostavil z uporabo programske kriptografske AVR-Crypto-Lib [4] napisane v programskem jeziku



Slika 3.2: Poenostavljena shema mrežnega krmilnika DMX na platformi Arduino.

C. Knjižnica je prilagojena izvajanju na mikroprocesorjih družine AVR. Programska knjižnica AVR-Crypto-Lib obsega implementacije velikega števila metod. Sam sem uporabil le implementacije zgoščevalnih funkcij ter implementacijo mehanizma HMAC. Za uporabljene funkcije sem implementiral vmesnik v programskem jeziku C++. Vmesnik se nahaja v obliki statične knjižnice in je primeren za uporabo v Arduino razvojnem okolju. Programska koda vmesnika je dostopna v prilogi na strani 14.

Preostali del programske kode, ki služi za izračun inicializacijskega vektorja, izmenjavo podatkov med krmilnikom in krmilno aplikacijo sem realiziral v programskem jeziku C. Programska koda je dostopna v prilogi na strani 14.

3.6 Primer avtentikacije

Za celovitejše razumevanje delovanja bom v nadaljevanju prikazal primer komunikacije ASCII med krmilnikom DMX ter krmilno programsko opremo. Spodnji primer vključuje pošiljanje inicializacijskega vektorja, avtentikacijo pristnega sporočila ter zavrnitev neavtentičnega. Znak > nakazuje pošiljanje znakov v smeri programska oprema krmilnik DMX, znak > pa obratno.

```
< CI192.168.1.30
> EthernetToDMX ready
> IV=jdas932nkj
> Command: Sxxx@xxxHddddddddddddddddddddddddddddddd#
< S200@50Hb004a892de958b173b25f9f70c8f7cb1#
> Message is authentic!
< S201@50Hb004a892de958b173b25f9f70c8f7cb1#
> Authentication has failed! Wrong HMAC!
< D
```

Razčlenimo zgornjo komunikacijo po logičnih enotah.

```
< CI192.168.1.30
```

Komunikacija se začne na strani programske opreme z ukazom `CI`[2], ki mu sledi IP številka brezžične naprave (v našem primeru 192.168.1.30).

```
> EthernetToDMX ready
> IV=jdas932nkj
> Command: Sxxx@xxxHddddddddddddddddddddddddddddddd#
```

Krmilnik po prejetem `CI` ukazu naključno izbere inicializacijski vektor ter izračuna tajni ključ, ki bo uporabljen za avtentikacijo HMAC. Za inicializacijski vektor `jdas932nkj` in prednastavljen tajni ključ `rncJhiekDe` je trenutni tajni ključ `7ac7ef6cd928085ed254ed57e368273a`$_{\mathbf{HEX}}$. Zadnja vrstica prikazuje format edinega ukaza, to je ukaza za nastavitev določenega kanala na želeno jakost. Veljaven ukaz je sestavljen iz treh podatkov in se nahaja v naslednji obliki `Sp1@p2Hp3#`. Podatek `p1` je številka kanala (število do 512), podatek `p2` predstavlja vrednost kanala (število do 255), podatek `p3`pa je avtentikacijska značka izračunana s trenutnim tajnim ključem in nizom `"Sp1@p2"`.

```
< S200@50Hb004a892de958b173b25f9f70c8f7cb1#
> Message is authentic!
```

Uporabnik nastavi kanal številka 200 na vrednost 50. Temu ustreza sporočilo `"S200@50"` in varnostna značka `b004a892de958b173b25f9f70c8f7cb1`$_{\mathbf{HEX}}$. Krmilnik ponovi izračun varnostne značke za prejeto sporočilo in ker je slednja enaka prejeti, sporočilo sprejme kot avtentično, in na izhodu vrednost kanala številka 200 spremeni na 50.

```
< S201@50Hb004a892de958b173b25f9f70c8f7cb1#
> Authentication has failed! Wrong HMAC!
```

V primeru, ko avtentikacijska značka ne ustreza prejetemu sporočilu, se sporočilo zavrne. Veljavna avtentikacijska značka za spročilo `"S201@50"` je `203a421805740e2dda1d618cbf595ca2`$_{\mathbf{HEX}}$.

```
< D
```

Seja se zaključi s prejetim znakom `D`.

[2]angl. Connection Incoming

Poglavje 4

Zaključek

Kljub temu, da je mehanizem HMAC prvotno namenjen avtentikaciji na področju internetnih protokolov vidimo, da se izkaže tudi na ostalih področjih. Zaradi svoje enostavnosti je zelo primeren za implementacijo na vgradnih sistemih. Implementacija predstavljena v tej seminarski nalogi zasede le 7552 bajtov izvršne programske kode.

Kljub temu glede implementirane avtentikacijske sheme obstajajo nekateri varnostni pomisleki oziroma možnosti za izboljšavo.

Prvi pomislek se nanaša na implementacijo generatorja naključnih števil. Slednji za seme uporablja vrednost nepovezanega analognega vhoda mikroprocesorja. Z uporabo natančnega laboratorijskega napajalnika je možno na vhod povezati električni tok znane napetosti in tako omejiti izbor semena generatorja naključnih števil. Ta problem bi lahko rešili, z uporabo strojnega vezja za uro v realnem času. Seme generatorja naključnih števil bi lahko izračunali na podlagi trenutnega časa.

Na prvi pogled se zdi napad z izbranim sporočilom mogoč. Napadalec bi lahko v daljšem časovnem obdobju spremljal promet med programsko opremo in krmilnikom in si za vsak inicializacijski vektor shranil videna sporočila ter njihove avtentikacijske značke. Uporaba takih podatkov je v praksi zelo težka, saj je možnih $5.9 \cdot 10^{19}$ različnih inicializacijskih vektorjev[1].

Avtentikacijsko shemo lahko še dodatno izboljšamo z omejitvijo generiranja inicializacijskih vektorjev. Trenutna implementacija avtentikacijske sheme ob vsaki vzpostavitvi seje izračuna nov inicializacijski vektor. To lahko izkoristi napadalec in z večkratno vzpostavitvijo seje vpliva na izbran vektor. Problem lahko rešimo tako, da krmilnik izbere nov vektor šele, ko je bil prejšnji vsaj enkrat uporabljen.

[1]Inicializacijski vektor dolžine 10 je sestavljen iz množice 95 izpisljivih znakov ASCII. Število vseh različnih inicializacijskih vektorjev je približno 95^{10}.

Literatura

[1] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In *Advances in Cryptology — CRYPTO '94 Proceedings*, volume 839. Springer Berlin, 1994.

[2] Schneier Bruce. Sha-1 broken. `http://www.schneier.com/blog/archives/2005/02/sha1_broken.html`, februar 2010.

[3] Atmel corporation. Mikroprocesor atmega328p. `http://www.atmel.com/dyn/products/product_card.asp?PN=ATmega328P`, februar 2010.

[4] Otte Daniel. Avr-crypto-lib. `http://avrcryptolib.das-labor.org`, februar 2010.

[5] Arduino foundation. Razvojno okolje arduino. `http://arduino.cc`, februar 2010.

[6] Bellare Mihir, Canetti Ran, and Krawczyk Hugo. Keying hash functions for message authentication. In *Advances in Cryptology — CRYPTO '96*, volume 1109, pages 1–15. Springer Berlin, 1996.

[7] Federal Information Processing Standards Publication. The keyed-hash message authentication code (hmac). Technical report, Information Technology Laboratory NIST, 2002.

Dodatek A

Programska koda

A.1 Vmesnik med okoljem Arduino in knjižnico AVR-Crypto-Lib

```
#include "Crypto.h"

extern "C"{
 #include "sha1.h"
 #include "md5.h"
 #include "hmac-md5.h"
 #include "hmac-sha1.h"
 #include <stdint.h>
 #include <string.h>
}

void CryptoClass::hashMD5(uint8_t* dst, const void* msg,
                          uint32_t l_b){
 md5((md5_hash_t *)dst, msg, l_b);
}

void CryptoClass::hashMD5(uint8_t* dst, const void* msg){
 int l;
 l = 8 * strlen((char*)msg);
 md5((md5_hash_t *)dst, msg, l);
}

void CryptoClass::hashSHA1(uint8_t* dst, const void* msg,
                           uint32_t l_b){
 sha1((sha1_hash_t *)dst, msg, l_b);
}
```

```
void CryptoClass::hashSHA1(uint8_t* dst, const void* msg){
 int l;
 l = 8 * strlen((char*)msg);
 sha1((sha1_hash_t *)dst, msg, l);
}

void CryptoClass::HMAC_SHA1(void* dst, void* key, uint16_t keyl_b,
                            void* msg, uint32_t msgl_b){
 hmac_sha1(dst, key, keyl_b, msg, msgl_b);
}

void CryptoClass::HMAC_MD5(void* dst, void* key, uint16_t keyl_b,
                           void* msg, uint32_t msgl_b){
 hmac_md5(dst, key, keyl_b, msg, msgl_b);
}

CryptoClass Crypto;
```

A.2 Sprejem sporočil ter preverjanje avtentičnosti

```
#include <Crypto.h>
#include <DmxSimple.h>

#define HASH_LEN 16
#define DEVICE_XPORT_RESET 2
#define DMX_OUT 3
#define MAX_CHANNELS 512

char IV[10];
char key[] = "rncJhiekDe";
char msg[20];

int pt;
int msg_len = 0;
int value = 0;
int channel;

uint8_t calculated_key[HASH_LEN];
uint8_t received_hmac[HASH_LEN];
uint8_t calculated_hmac[HASH_LEN];
```

```
void initXPort(){
  digitalWrite(DEVICE_XPORT_RESET, LOW);
  delay(50);
  digitalWrite(DEVICE_XPORT_RESET, HIGH);
  delay(1000);
}

void initDMX(){
  DmxSimple.maxChannel(MAX_CHANNELS);
  DmxSimple.write(MAX_CHANNELS, 0);
}

void initPins(){
  pinMode(DEVICE_XPORT_RESET, OUTPUT);
  DmxSimple.usePin(DMX_OUT);
  randomSeed(analogRead(0));
}

void displayWelcome(){
  // Private key.
  for(int i=0; i<10; i++){
    IV[i] = (int)random(32, 127);
  }
  Crypto.HMAC_MD5(calculated_key, key, 80, IV, 80);

  Serial.println("EthernetToDMX ready");
  Serial.print("IV=");
  Serial.print(IV);
  Serial.println("\nCommand: Sxxx@xxxHddddddddddddddddddd
                 ddddddddddddddd#");
}

void setup() {
  Serial.begin(9600);
  initPins();
  initXPort();
  initDMX();
}

void loop() {
  int c= -1;

  while(!Serial.available());
  c = Serial.read();
```

```
if(c == 'D'){
  for(int i=1; i<=512; i++){
    DmxSimple.write(i, 0);
  }
}
else if (c == 'C'){
  displayWelcome();
}
else if(c == 'S'){
  value = 0;
  channel=0;

  msg_len = 0;
  msg[msg_len++] = c;

  for(int i=0; i<42; i++){
    while(!Serial.available());
    c = Serial.read();

    if ((c>='0') && (c<='9')) {
      value = 10*value + c - '0';
      msg[msg_len++] = c;
    }
    else if(c == '@'){
      channel = value;
      msg[msg_len++] = c;
      value = 0;
    }
    else if(c == 'H'){
      for(int t=0; t<HASH_LEN; t++){
              int r, r2;
              while(Serial.available()<2);
              r = charToInt(Serial.read());
              r = r << 4;
              r2 = charToInt(Serial.read());
              received_hmac[t] = r + r2;
      }

      while(!Serial.available());
      c = Serial.read();

      if (c == '#'){
        Crypto.HMAC_MD5(calculated_hmac, calculated_key,
```

```
                                HASH_LEN * 8, msg, msg_len * 8);
            if(hmac_cmp(received_hmac,calculated_hmac) == 0){
              Serial.write("Message is authentic!\n");
            }
            else{
              Serial.write("Authentication has failed! Wrong HMAC!\n");
              break;
            }

            // validate input range
           if((channel >= 0) && (channel <= 512) && (value >= 0) &&
              (value <= 255)){
             DmxSimple.write(channel, value);
             break;
           }
         }
       }
     }
   }
   loop();
}

int hmac_cmp(uint8_t* a, uint8_t* b){
  int i;
  for(i=0; i<HASH_LEN; i++){
    if(a[i] != b[i]){
      return 1;
    }
  }
  return 0;
}

int charToInt(char t){
  int tt = (int)t;
  if(tt >= 48 && tt <= 57){
    return (tt - (int)'0');
  }
  else if(tt >= 97 && tt <= 102){
    return tt - (int)'a' + 10;
  }
  return -1;
}
```