KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

Ignas Žilinskas

Būsto kompiuterinės sistemos saugos sistemos sudarymas ir tyrimas.

Magistro darbas

Darbo vadovas
Prof. Dr. Egidijus Kazanavičius

Kaunas, 2012

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

Ignas Žilinskas

Būsto kompiuterinės sistemos saugos sistemos sudarymas ir tyrimas.

Magistro darbas

Recenzentas
Prof. Dr. Rimantas Plėštys
2012-05-

Vadovas
Prof. Dr. Egidijus Kazanavičius
2012-05-

Atliko
IFN-0/3 gr. stud.
Ignas Žilinskas
2012-05-18

Kaunas, 2012

Turinys

1. Pratarmė

Populiarėjant būsto kompiuterinėms sistemoms, jų saugumas darosi vis aktualesnė problema, nes vis daugiau vagių pasirenka atakuoti šias sistemas, kad patektų į namus, ar nepatekę į juos išgautų jiems reikalingą informaciją arba sutrikdytų sistemos darbą. Dėl to šiame darbe nagrinėjamos būsto kompiuterinės sistemos saugumo problemos ir siūlomi jų sprendimo būdai. Kadangi saugumo tema yra labai plati, darbe apsiribojama saugių duomenų mainų pačioje sistemoje užtikrinimu.

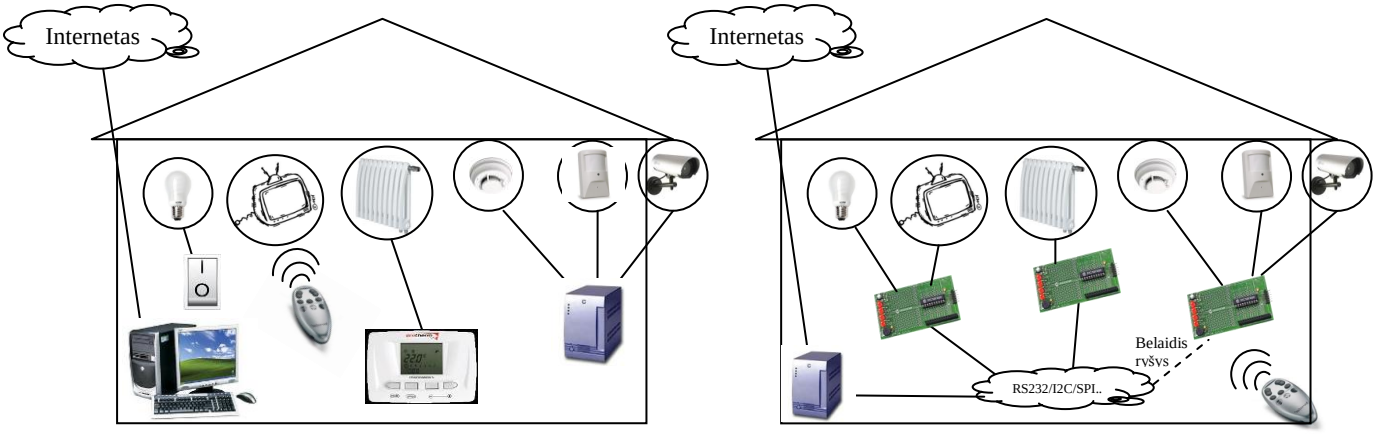
Duomenų mainų saugumo problema buvo pasirinkta dėl jos aktualumo. Perėmus tarp sistemos įtaisų siunčiamus duomenis galima išsiaiškinti kaip veikia sistema ir išgauti įvairią informaciją apie namų šeimininką. O šiuos duomenis suklastojus galima sutrikdyti visos sistemos darbą. Nepaisant šių grėsmių, šiuo metu nėra nustatytų konkrečių būdų kaip nuo jų apsisaugoti. Todėl šio darbo tikslas yra pasiūlyti efektyvius duomenų mainų apsaugojimo būdus.

2. Įvadas

Darbo objektas

Šiuolaikiniuose namuose ir biuruose yra aibė įvairių sistemų: elektros tiekimo, šildymo, vėdinimo, kondicionavimo, apsaugos ir t.t. Tačiau kiekvienai jų turint nesusietas su kitomis sistemomis valdymo galimybes, dažnai taip nutinka, kad šios sistemos dirba neekonomiškai ir neefektyviai. Norint apjungti visas namuose veikiančias sistemas į vieną, buvo sugalvota „protingų namų“ (angl. Smart House) kompiuterinė sistema.

„Protingų namų“ galimybės beveik neribotos, jas riboja tik mūsų fantazija – sujungti į vieną kompiuterinę sistemą ir valdyti galima viską, kas valdoma elektra. Tokia sistema gali savarankiškai palaistyti veją, įjungti ar išjungti apsaugos sistemą, įvairius buities prietaisus, apšvietimą, reguliuoti šildymą, kondicionavimo sistemą ir t.t. O kad visą tai valdyti, vartotojui pakanka spustelėti vieną ar kelis mygtukus.



1 pav. Namai be kompiuterinės sistemos ir su ja.

„Protingų namų“ sistemos turi daug privalumų, tačiau jos turi ir problemų.

Problemos, su kuriomis susiduria būsto kompiuterinės sistemos

Apžvelgus anksčiau darytus darbus, susijusius su būsto kompiuterinės sistemos saugumu, buvo rasta, kad nemažai autorių nagrinėjo įvairias problemas ir siūlė savo sprendimo būdus. Keletas nagrinėtų problemų ir pasiūlytų sprendimų:

- Žurnalo „Computer Methods and Programs in Biomedicine“ straipsnyje apie „protingus namus“ buvo paminėta gana svarbi šių sistemų problema. Tai įvairių standartų ir protokolų suderinimas, nes šiuo metu įvairūs įtaisai naudoja įvairius komunikacijos būdus ir todėl sunku juos visus sujungti į vieningą sistemą. Straipsnio autoriai remdamiesi įvairiais darbais siūlo įvesti bendrus informacijos perdavimo

standartus, įtaisams, kurie būtų naudojami „protingų namų“ sistemose. Jų manymu tokioms sistemoms labiausiai tinkamas informacijos perdavimas yra belaidis. [1]

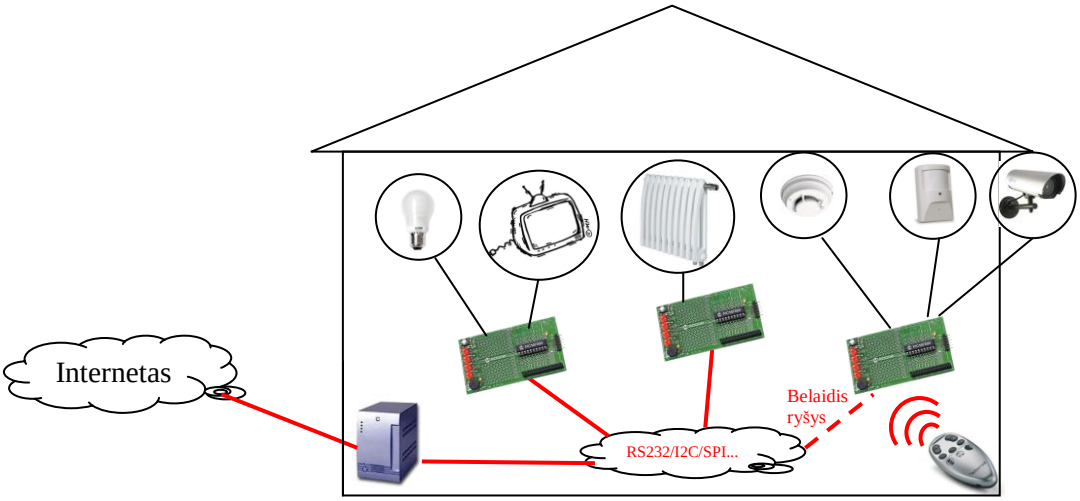
- Žurnale „International Journal of Advanced Science and Technology“ spausdintame straipsnyje apie „protingų namų“ saugumą buvo rašyta apie DDoS atakas panaudojant elektroninius laiškus, kai serveris būna užkraunamas didžiuliu kiekiu el. laiškų ir nebegali daugiau nieko aptarnauti. Autoriai teigia, kad šia problemą galima patikimai išspręsti panaudojus el. laiškų filtrus. [2]
- Dar keliuose straipsniuose buvo nagrinėta šeimininko atpažinimo problema. Nes dabar dažniausiai naudojamus slaptažodžius yra gana lengva nulaužti. Šią problemą spręsti autoriai siūlo panaudojant neuroninius tinklus. [3] Prie šeimininko atpažinimo būtų galima paminėti ir dar vieną straipsnį, kuriame nagrinėjamas prieigos prie kompiuterinės namų sistemos suteikimas svečiui. [4]
- Taip pat buvo peržvelgti keli straipsniai nagrinėjantys saugius duomenų mainų protokolus. Visi straipsniuose minimi protokolai buvo šifruojami naudojant kriptografiją. Taip pat buvo paminėta kad ir tokie šifruoti protokolai turi savo silpnumų ir ne visada yra visiškai saugūs. [5]

Toliau apibrėžiamos būsto kompiuterinės sistemos problemos, susijusios su šiuo darbu (2 pav.). Jas galima išskirti į dvi grupes:

- **Suderinamumas**. Reikia užtikrinti, kad informacijos mainai sklandžiai vyktų tarp sistemos įtaisų, kurie naudoja skirtingas fizines sąsajas ir skirtingus duomenų mainų protokolus. To reikia tam, kad prie sistemos galima būtų prijungti kuo įvairesnius įtaisus, taip išplečiant sistemos galimybes.
- **Saugumas**. Sistemos įtaisai, gali būti sujungti tarpusavyje duomenų mainų protokolais, kurie šiuo metu daugeliu atvejų būna atviri ir neapsaugoti. Į saugumą galima pažiūrėti iš dviejų pusių:
    - Saugumas vartotojo atžvilgiu. Vartotojas nori būti tikras, kad jo kompiuterinės namų sistemos nieks neapgaus, neišjungs ar kitaip nesutrikdys jos darbo, netyčia ar norėdamas pakenkti. Jam rūpimus sistemos saugumo klausimus galima suformuluoti taip:
        - apsauga nuo fizinio įsibrovimo į namus ir sistemos išjungimo ar valdymo perėmimo.
        - apsauga nuo įsilaužimo ir valdymo perėmimo ar duomenų suklastojimo/sunaikinimo per internetą.

- apsauga nuo laidais ar belaidžiu ryšiu siunčiamų duomenų nuskaitymo ar suklastojimo.
- apsauga nuo sistemos darbo sutrikdymo ar valdymo perėmimo panaudojus kai kurių jos įtaisų naudojamą belaidį ryšį.
- saugūs duomenų mainai tarp skirtingais protokolais bendraujančių įtaisų.

- Saugumas sistemos gamintojo atžvilgiu. Gamintojas nori, kad vartotojas rinktųsi tik jo siūlomus sprendimus ir įtaisus savo namų kompiuterinei sistemai. Todėl yra bandoma sugalvoti duomenų mainų būdus ir šifravimo algoritmus, kad sistema atpažintu tik vieno gamintojo įtaisus, o kitų gamintojų įtaisai su sistema negalėtų „susikalbėti“. Gamintojas yra suinteresuotas šiuos duomenų šifravimo algoritmus laikyti paslaptyje ir apsisaugoti nuo jų atskleidimo paimant iš įdiegtos sistemos. Taigi, gamintojui rūpimus sistemos saugumo klausimus būtų galima apibrėžti taip:
  - apsauga nuo laidais ar belaidžiu ryšiu siunčiamos informacijos išanalizavimo, tokiu būdu atskleidžiant vieno ar kito algoritmo veikimo principą.
  - apsauga nuo laidais ar belaidžiu ryšiu siunčiamos informacijos išanalizavimo, ir taip atsirandančios galimybės sistemai siųsti komandas iš jai nepriklausančių įtaisų.
  - apsaugoti sistemą nuo kito gamintojo įtaisų prijungimo.



2 pav. Namų kompiuterinės sistemos probleminės vietos.

Darbe keliamos problemos

Šiame darbe apsiribosime tik tomis saugos problemomis, kurias galima išspręsti pačios kompiuterinės sistemos viduje. Tai saugaus duomenų perdavimo tarp sistemos įtaisų užtikrinimas laidais ir belaidžiu ryšiu, bei apsauga nuo sistemos valdymo perėmimo ar darbo sutrikdymo suklastojus ar sugadinus perduodamus duomenis.

Tolesniuose skyriuose bus apžvelgti dažniausiai naudojami tokių sistemų standartai, rinkoje siūlomi įtaisai ir tų įtaisų naudojami protokolai bei sąsajos.

Darbo uždaviniai

Įvardinus darbe nagrinėjamas problemas galima suformuluoti ir darbo uždavinius:

- Realizuoti tarp sistemos įtaisų siunčiamų duomenų apsaugos algoritmus, kad šių duomenų nebūtų galima perimti, o tai padarius, nebūtų galima jų suprasti ar suklastoti.
- Realizuoti algoritmą, apsaugantį sistemą nuo jos darbo sutrikdymo galimybės panaudojus jos įtaisų naudojamą belaidį ryšį. Sistema turi žinoti kaip reaguoti sutrikus belaidžiam ryšiui ir mokėti atpažinti bei reaguoti į piktavališkus belaidžio ryšio trikdžius.
- Apsaugoti sistemą nuo galimybės prie jos prijungti jai nepriklausančius įtaisus.
- Realizuoti eksperimentinę sistemą.
- Ištirti eksperimentinėje sistemoje įdiegtą apsaugą ir įvertinti jos patikimumą.

3. Standartų, skirtų būsto kompiuterinei sistemai sujungti ir valdyti, analizė

Egzistuoja keletas standartų, kurie aprašo kaip turėtų būti sujungta ir valdoma būsto kompiuterinė sistema. Vienas iš jų yra „**C-Bus**“ [13]. Tai yra mikroprocesoriais grįsta namų ir biurų valdymo sistema. Ji naudojama apšvietimo, garso ir vaizdo įtaisų ir įvairių kitų elektrinių prietaisų ar sistemų valdymui. Kiekvienas tokios sistemos įtaisas turi savo mikroprocesorių, kas leidžia kiekvieną įtaisą atskirai suprogramuoti. Ši sistema nereikalauja centrinio kompiuterio ar kontrolerio.

Pagrindiniai „C-Bus“ sistemos privalumai:

- Patikima valdymo sistema, bet nedidelė mazgų kaina.
- Platus įrankių, leidžiančių sąveikauti su įterptinėmis ir kompiuteriu valdomomis sistemomis, pasirinkimas.

- Vienas „C-Bus“ kabelis gali valdyti neribotą skaičių įtaisų.
- Didelis lankstumas keičiant sistemą. Funkcijos gali būti keičiamos, pridedamos, pašalinamos, perprogramuojamos bet kuriuo metu ir bet kuriame tinklo taške be sudėtingos techninės įrangos.
- „C-Bus“ yra lengvai sumontuojama.
- „C-Bus“ gali valdyti ir skaitmeninius ir analoginius įtaisus.

Kitas plačiai naudojamas standartas yra „**X-10**“ [14]. Tai atviras standartas, skirtas elektrinių įtaisų komunikacijai namų kompiuterinėse sistemose. Jis naudoja jau esantį elektros tinklą signalų siuntimui ir įtaisų valdymui. Taip pat šiame standarte yra apibrėžtas ir belaidis, radijo bangomis veikiantis protokolas.

„X-10“ sistemos privalumai:

- Didelis valdiklių ir suderinamų sistemų pasirinkimas.
- Valdikliai nėra brangūs.
- Sistemą paprasta įdiegti.
- Diegiant šią sistemą nereikia keisti elektros instaliacijos.
- Galima valdyti iki 256 apšvietimo grupių, buitinės technikos, vaizdo ar garso aparatūros.

Dar vienas būsto kompiuterinės sistemos standartas yra „**KNX**“ [15]. Tai „KNX“ asociacijos sukurta technologija, skirta valdyti visoms namuose esančioms sistemoms, nuo apšvietimo ir garso aparatūros iki kondicionavimo ir šildymo. Ši technologija taip pat gali būt atsakinga už aplinkos stebėjimą, apsaugą ir signalinius pranešimus. „KNX“ standarto tikslas – padaryti kuo lankstesnę sistemą, nepriklausančią nuo konkretaus įtaisų gamintojo ar komunikavimo būdo. Tokioje sistemoje yra suderinami įvairių gamintojų įtaisai.

„KNX“ privalumai:

- Tai patvirtintas tarptautinis standartas
  - ISO/IEC 14543-3 tarptautinis standartas, patvirtintas 2006.
  - EN 50090 Europos standartas, patvirtintas 2003.
  - EN 13321-1 ir EN1332-2 standartai, patvirtinti 2006.
  - GB/Z 20965 kinų standartas, patvirtintas 2007.
  - ANSI/ASHRAE 135 Jungtinių Valstijų standartas, patvirtintas 2005.
- „KNX“ sertifikavimo procesas užtikrina, kad skirtingų gamintojų įvairūs įtaisai, naudojami skirtingose sistemose sklandžiai veiks ir komunikuos tarpusavyje.

- Kiekvienas gamintojas, gavęs „KNX“ sertifikatą atitinka tarptautinius standartus, taip pasiekiama aukšta kokybė.
- Unikali, nuo gamintojo nepriklausoma programinė įranga, suteikianti galimybę planuoti, projektuoti ir konfigūruoti sistemą, susidedančią iš „KNX“ sertifikuotų įtaisų.
- „KNX“ gali būti naudojama visoms buto ar viso pastato sistemoms valdyti.
- „KNX“ yra lengvai pritaikoma įvairių tipų namams. Ši sistema gali būti naudojama tiek mažame vienos šeimos name, tiek dideliuose pastatuose (viešbučiuose, ligoninėse, mokyklose).
- „KNX“ palaiko skirtingų konfigūracijų mazgus.
- „KNX“ palaiko skirtingus komunikavimo būdus
  - Vytų porų kabeliai
  - Elektros tiekimo linija
  - Radijo bangos
  - IP/Interneto sąsaja
- „KNX“ gali būti sujungta su kitomis sistemomis.
- „KNX“ yra nepriklausoma nuo konkrečios programinės ar techninės įrangos.

4. Būsto kompiuterinės sistemos įtaisų analizė

Šiame skyriuje pateikiama keletas rinkoje siūlomų įtaisų, kuriuos galima panaudoti projektuojant būsto kompiuterinę sistemą. 1 lentelėje pateikiamas įtaisų palyginimas.

Kompanijos Pervasa siūloma platforma „Atlas“ [6] gali susidėti iš keleto skirtingų platformų, sujungiamų į vieną mazgą. Kiekviena iš platformų atlieka tam tikrą funkciją. Taip galima surinkti mazgus, geriausiai atitinkančius keliamus reikalavimus. Į šių mazgų komplektaciją įeina ir jiems skirta programinė įranga.

3 pav. Atlas mazgas [6].

„Atlas“ siūlomos platformos:

- Universali antena dirbanti 2.4 GHz dažniu, kuri gali būti naudojama belaidžiam ryšiui, tokiam kaip 802.11 b/g ir ZigBee. Kaina – 40$.
- Stiprintuvų sąsajos modulis, skirtas prijungti iki 6 stiprintuvų. Kaina – 80$.
- Analoginių jutiklių modulis leidžia prijungti iki 32 analoginių jutiklių. Kaina – 110$.

- Skaitmeninių jungčių modulis prie kurio galima prijungti iki 16 skaitmeninių jutiklių, jungiklių ir t.t. Kaina – 80$.
- Bendro naudojimo I/O modulis suteikia galimybę prijungti ir analoginius ir skaitmeninius jutiklius. Šis modulis leidžia naudoti 40 skaitmeninių įvesties/išvesties kontaktų ir 8 analoginius įvesties kontaktus. Taip pat šis modulis suteikia galimybę užprogramuoti kontaktus pritaikant specialiam naudojimui. Kaina – 90$.
- Wi-Fi komunikacijos modulis užtikrina 802.11 b/g belaidį ryšį tarp mazgų. Kaina – 200$.
- ZigBee komunikacijos modulis užtikrina ZigBee belaidį ryšį tarp mazgų. Kaina – 170$.
- Interneto komunikacijos modulis leidžia mazgus sujungti į tinklą naudojant interneto kabelius. Kaina – 160$.
- Skaičiavimo modulis, skirtas surinktų duomenų apdorojimui ir gautų komandų vykdymui. Turi 8 bitų, 8 Mhz ATmega128L mikrokontrolerį. Kaina – 200$.

„PICkit 2“ – tai kompanijos Microchip platforma *DM164120-2(PIC16F887)* su jos programavimui skirtu priedu ir programine įranga [7]. Ši platforma turi 40 MHz, 8 bitų mikrokontrolerį, 8Kb flash ir 368b RAM atmintį.
Naudojamos jungtys: 66 I/O, 15 10bitų ASK, 2 SPI/I^2C, 2 A/E/USART.
Visas „PICkit 2“ komplektas kainuoja apie 100$.


4 pav. PICkit 2 [7].

„TelosB“ – tai įtaisas iš Berkeley universiteto, platinamas Crossbow firmos [8]. Jis turi 16 bitų, 8MHz, RISC architektūros procesorių, 10Kb operatyvinę, 48Kb flash ir 1Mb išorinę atmintį. Šis įtaisas turi USB jungtį per kurią gali komunikuoti su kompiuteriu ir būti užprogramuojamas. Kitos naudojamos jungtys yra: UART, Digital I/O,I^2C,SPI, 12 bitų ASK ir SAK. Taip pat jame yra įmontuotas ir 2.4 GHz radijo siųstuvas-imtuvas, palaikantis IEEE 802.15.4 protokolą.


5 pav. TelosB [8].

„TelosB“ įtaise gali būti įdiegtos Contiki, TinyOS, SOS ir MantisOS operacinės sistemos. Jo kaina yra 134 $.

„iMote2“ yra galingas mazgas, turintis daug atminties (256Kb SRAM, 32Mb flash, 32Mb SDRAM). Jame įmontuotas Intel PXA271 32 bitų, RISC architektūros procesorius, kurio dažnis gali kisti nuo 13 iki 416MHz. Jo išplėtimo jungtys palaiko daug standartų (3xUART, 2xSPI, I^2C, SDIO, GPIOs). „iMote2“, kaip ir „TelosB“ turi USB jungtį ir tokių pačių charakteristikų radijo siųstuvą-imtuvą. Šiame įtaise galima įdiegti Microsoft .NET Micro, TinyOS ir netgi Linux operacines sistemas. Tačiau jo ir kaina yra gana didelė ir siekia 299$ [9].

6 pav. iMote2 [9].

Įtaisų palyginimas

1 lentelė. Įtaisų palyginimas.

	Procesorius	**Atmintis**	**Jungtys**	**Kaina**
Atlas	8 bitų, 8 Mhz	128Kb flash, 4Kb SRAM	Wi-Fi, ZigBee, Ethernet, GPIO, ASK,	~450 $
PICkit 2	8 bitų, 20 MHz	8Kb flash, 368b RAM	I/O, ASK, SPI, I^2C, A/E/USART	100 $
TelosB	16 bitų, 8MHz	48Kb flash, 10Kb RAM, 1Mb flash išorinė	USB, Wi-Fi, ZigBee, UART, Digital I/O, I^2C, SPI, ASK	134 $
iMote2	32bit, 13-416Mhz	32Mb flash, 32Mb SDRAM	USB, Wi-Fi, UART, SPI, I^2C, SDIO, GPIO	299 $

Visi peržvelgti įtaisai yra programuojami ir gali būsto kompiuterinėje sistemoje veikti be centralizuoto valdymo. Todėl kiekvienas įtaisas yra atsakingas už saugumą ir turės atitikti apsaugos sistemos keliamus techninius reikalavimus. Tačiau šie įtaisai netinka „X-10“ standartą naudojančiose sistemose. Tokios sistemos dažniausiai būna valdomos centralizuotai, iš vieno kontrolerio ar kompiuterio, su specialiu priedu, todėl jų apsauga bus realizuojama būtent valdymo įtaise.

5. Įtaisuose naudojamų protokolų ir sąsajų analizė

RS232 [10]

RS232 yra asinchroninės nuoseklios komunikacijos standartas. Tai reiškia, kad duomenys čia perduodami po vieną bitą ir perdavimui nėra iš anksto nustatytų laiko intervalų. Duomenų siuntimas gali prasidėti bet kada ir imtuvo užduotis yra aptikti siunčiamos žinutės pradžią ir pabaigą. Siunčiama informacija turi būti sudalinama į žodžius. Žodžio ilgis gali kisti nuo 5 iki 8 bitų. Svarbu, kad siųstuvas ir imtuvas naudotų vienodo ilgio žodžius. Asinchroninėj

komunikacijoj yra reikalingi papildomi bitai žymintys informacijos siuntimo pradžią ir pabaigą.

Duomenys yra siunčiami iš anksto nustatytu dažniu. Siųstuvas ir imtuvas turi dirbti tuo pačiu dažniu. Kai gaunamas žinutės pradžios bitas, siųstuvas skaičiuoja laiką, kada bus siunčiamas kitas informacijos bitas ir tuo momentu patikrina linijos įtampos lygį.

RS232 standarte linijos įtampa gali turėti tik dvi būsenas, aukštą (1) ir žemą (0). Kai linija nenaudojama, įtampa yra aukštoje būsenoje. Siunčiamos žinutės pradžios bitas nuleidžia įtampą į žemą būseną ir taip praneša imtuvui, kad prasidėjo duomenų siuntimas. Jei imtuvas nepastebėtų pražios bito pradėtų nuskaitinėti žinutę vėliau ir taip būtų gaunami blogi duomenys. Kad to išvengti yra naudojamas žinutės pabaigos bitas, kuris įtampą pakelia į aukštą būseną.

Šis standartas buvo sukurtas 20 kbit/s greičiui. Taip pat laido ilgis turi būti ne ilgesnis kaip 15 metrų.

UART [11]

UART – Universalus Asinchroninis Imtuvas-Siųstuvas (angl. *Universal Asynchronous Receiver – Transmiter*). Šiuo protokolu bendraujantis įtaisas siunčiamus duomenis išskaido į bitus ir nuosekliu būdu išsiunčia. Signalą priimantis įtaisas gautus bitus vėl sujungia.

Šiuo protokolu bendraujančių įtaisų tarpusavio santykis yra master/slave. Bendravimas gali būti inicijuojamas tik vieno įtaiso tinkle, pagrindinio (master) įtaiso. Duomenys yra perduodami po 8 bitus. Nėra lyginumo (parity). Sparta gali siekti 112kbit/s ir daugiau, priklauso nuo įtaisų. UART dažniausiai naudoja kitas standartines sąsajas, kurios apibrėžia įtampos lygius ir kitas susijungimo charakteristikas. Tokie standartai yra EIA RS232, RS422, RS 485. Dėl to daugelyje sistemų UART jungtis prijungta prie mikroschemų, kurios generuoja signalus, suderinamus su RS232 reikalavimais.

I^2C [12]

I^2C (angl. *Inter-Integrated Circuit*) – tai Philips Semiconductors sukurta paprasta dvikryptė dviejų laidų magistralė. Jau pavadinimas sako, kad jos tikslas – suteikti galimybę integruotoms schemoms paprastai bendrauti tarpusavyje.

Pagrindinės protokolo savybės:

- Bendravimas vyksta tik per du laidus: nuosekli duomenų linija (SDA – serial data line) ir nuosekli sinchronizavimo linija (SCL – serial clock line);

- Nėra griežtų spartos reikalavimų, pagrindinis (master) įtaisas generuoja sinchronizavimo dažnį;
- Tarp visų komponentų egzistuoja paprastas master/slave sąryšis. Pagrindinis įtaisas inicijuoja bendravimą. Kiekvienam įtaisui, prijungtam prie magistralės, programiniu būdu suteikiamas unikalus adresas;
- I2C protokolas palaiko daug pagrindinių įtaisų ir užtikrina arbitravimą ir kolizijų aptikimą, taip išvengiant duomenų sugadinimo;
- Dvikrypčiai, 8 bitų duomenų perdavimai gali būti atliekami 100 kbit/s, 400 kbit/s, 1 Mbit/s arba 3.4 Mbit/s greičiu;
- Prie magistralės prijungiamų įtaisų skaičius ribojamas tik magistralės talpinės varžos

SPI [12]

SPI – nuosekli periferinė sąsaja (angl. *Serial Peripheral Interface*). Tai sinchroninis, nuoseklus duomenų perdavimo būdas. Ši sąsaja yra dvikryptė. Prietaisai komunikuoja *master/slave* režimu, kur pagrindinis įtaisas inicijuoja duomenų siuntimą. Yra galimas tik vienas pagrindinis įtaisas keli *slave* įtaisai. Prieš siųsdamas duomenis pagrindinis įtaisas pirmiausia nustato siuntimo dažnį, kurį palaiko *slave* įtaisas.

SPI privalumai:

- Dvikryptė komunikacija;
- Didesnis pralaidumas nei I^2C;
- Lankstus protokolas siunčiamai informacijai, žodžio ilgis neribojamais 8 bitais, galima pasirinkti žinutės dydį, turinį ir tikslą;
- Ypatingai paprasta techninė sąsaja. (mažesni galios reikalavimai, nei I^2C, *slave* įtaisai naudoja pagrindinio įtaiso dažnį ir jiems nereikia unikalaus adreso, nereikalingi siųstuvai-imtuvai)

SPI trūkumai:

- Reikia daugiau laidų, nei I^2C sąsajoje.
- Nėra aparatinio *slave* įtaisų patvirtinimo (Pagrindinis įtaisas gali siųsti informacija, kai niekas jos nepriima ir apie tai nežinoti)
- Palaiko tik vieną pagrindinį įtaisą.
- Nėra klaidų aptikimo protokolo
- Jautrumas triukšmams padidina klaidingų duomenų tikimybę
- Palaiko tik netolimus atstumus, panašius į RS-232.

GPIO sąsaja

GPIO (*General Purpose Input/Output*) leidžia lengvai išplėsti įvesties/išvesties jungtis, naudojant tokias standartines sąsajas kaip I^2C ar SPI. Kiekviena GPIO jungtis gali būti atskirai programiniu būdu sukonfigūruojama, kaip įėjimo ar išėjimo jungtis.

IEEE 802.11

Belaidžio ryšio standartas IEEE 802.11 (*Institute of Electrical and Electronics Engineers*) užtikrina duomenų perdavimą belaidžiu būdu, 2.4, 3.6 ir 5 GHz dažnių ruože. Šis standartas buvo sukurtas 1997 metais ir buvo pirmasis standartas skirtas įtaisų sujungimui į belaidį tinklą. Nuo 1997 metų buvo sukurta nemažai šio standarto atšakų. Deja, šis standartas yra neatsparus įsilaužėliams ir yra neapsaugotas nuo slapto duomenų rinkimo (*sniffing*).

Protokolų ir sąsajų analizės apibendrinimas

Iš išvardintų sąsajų bei protokolų nei viename nėra numatyta duomenų apsauga. Todėl kad ir koks duomenų perdavimo protokolas bus naudojamas, reikės papildomai pasirūpinti, kad perduodamų duomenų niekas neperimtu, o jei ir perimtų, kad negalėtų jų suprasti ar suklastoti. Šiame darbe bus pasiūlyti duomenų apsaugos algoritmai, kurie nepriklausys nuo konkretaus protokolo, taip palikdami laisvę rinktis patogiausią duomenų perdavimo būdą.

6. Darbui pasirinkta techninė ir programinė įranga

Darbe pasiūlyti duomenų mainų apsaugojimo būdai bus patikrinti realizavus eksperimentinę būsto kompiuterinę sistemą. Šiai sistemai realizuoti pasirinktas „PICkit 2“ įtaisas, tiksliau PIC16F877 įtaisas, nes jo kaina yra nedidelė lyginant su kitais anksčiau aprašytais įtaisais. Taip pat šis įtaisas turi daugumą jungčių naudojamų ir kituose įtaisuose. Mikrokontrolerio ir atminties parametrai yra panašūs kaip ir kitų nelabai brangių įtaisų. Šiai sistemai nereikia galingo įtaiso, turinčio daug atminties, nes tada išaugtų tokių sistemų kaina. Realizuoti duomenų apsaugos algoritmai turės sklandžiai veikti įtaisuose su ribotais skaičiavimo ir atminties resursais.

Programavimui bus naudojama MPLAB aplinka su integruotu PIC C Compiler c kalbos kompiliatoriumi. Vietoj RS232 sąsajos, bus naudojamas PICkit2 įtaisas, skirtas pasirinktos PIC16F877 schemos programavimui ir tuo pačiu galintis atlikti RS232 sąsajos funkciją. Šis įtaisas, kartu su jam skirta programine įranga, taip pat bus naudojamas siunčiamų signalų stebėjimui.

7. Darbe keliami uždaviniai

Apibrėžus darbe keliamas būsto kompiuterinių sistemų problemas ir išanalizavus tokiose sistemose naudojamus įtaisus bei duomenų perdavimo protokolus, darbui buvo iškelti tokie uždaviniai:

- Realizuoti tarp sistemos įtaisų siunčiamų duomenų apsaugos algoritmus, kad šių duomenų nebūtų galima perimti, o tai padarius, nebūtų galima jų suprasti ar suklastoti. Šie algoritmai neturi priklausyti nuo naudojamų protokolų ir turi veikti įtaisuose su ribotais skaičiavimo ir atminties resursais.
- Realizuoti algoritmą, apsaugantį sistemą nuo jos darbo sutrikdymo galimybės panaudojus jos įtaisų naudojamą belaidį ryšį. Sistema turi žinoti kaip reaguoti sutrikus belaidžiam ryšiui ir mokėti atpažinti bei reaguoti į piktavališkus belaidžio ryšio trikdžius.
- Apsaugoti sistemą nuo galimybės prie jos prijungti jai nepriklausančius įtaisus.
- Realizuoti eksperimentinę sistemą.
- Ištirti eksperimentinėje sistemoje įdiegtą apsaugą ir įvertinti jos patikimumą.

8. Būsto kompiuterinės sistemos saugos sistemos specifikacija

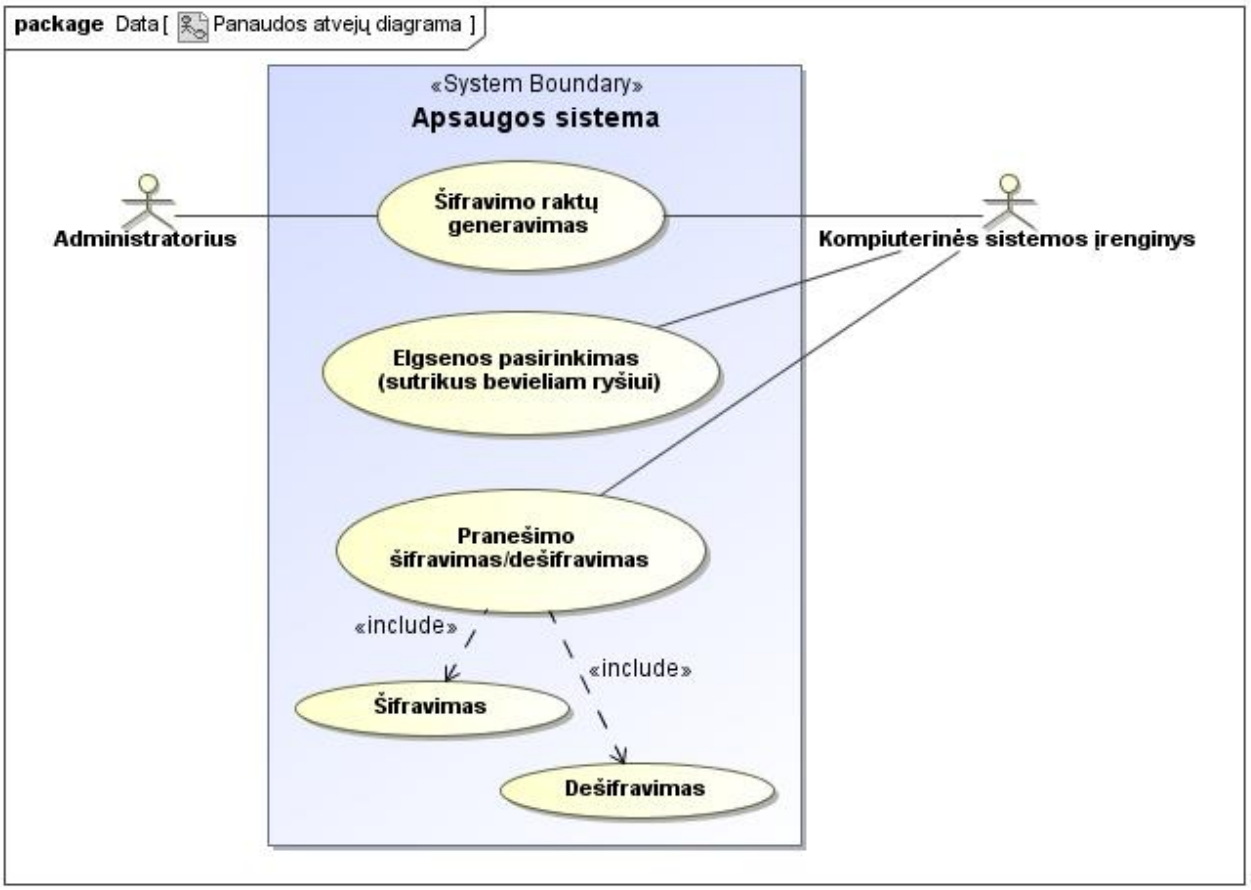
8.1. Projektuojama sistema

Projektuojama būsto kompiuterinės sistemos apsaugos sistema, kuri užtikrintų saugų duomenų ir komandų perdavimą tarp kompiuterinės sistemos įtaisų. Šią saugos sistemą sudarys siunčiamus pranešimus šifruojanti ir šifravimo raktus generuojanti posistemė ir už įtaisų komunikaciją radijo ryšiu, bei teisingą jų reakciją į radijo ryšio trikdžius atsakinga posistemė.

8.2. Funkciniai reikalavimai projektuojamai apsaugos sistemai

- Pranešimų, siunčiamų tarp būsto kompiuterinės sistemos įtaisų, šifravimas.
- Saugus šifravimo raktų apsikeitimas tarp sistemos įtaisų.
- Teisinga įtaisų, naudojančių belaidį ryšį, reakcija į ryšio trikdžius.
- Galimybė valdyti apsaugos sistemą.

Projektuojamos sistemos funkcijos atsispindi panaudos atvejų diagramoje (7 pav.).



7 pav. Būsto kompiuterinės sistemos panaudos atvejų diagrama.

8.3. Saugos sistemos reikalavimai, keliami kompiuterinės sistemos techninei įrangai

Būsto kompiuterinę sistemą sudarantys įtaisai turi tenkinti šiuos reikalavimus:

- Šifravimo ir dešifravimo algoritmai turi sklandžiai veikti sistemos įtaisuose, tam šie turi turėti 8 bitų mikroprocesorių ir bent 8 Kb programai skirtą atmintį, bei 368 baitų RAM atmintį.[1]

8.4. Reikalavimai duomenų šifravimo algoritmams

- Duomenų šifravimas ir dešifravimas turi sklandžiai veikti įtaisuose su 8 bitų mikroprocesoriumi.
- Dviejų vienodų pranešimų šifrogramos turi būti skirtingos.
- Šifravimas ir dešifravimas turi užtrukti ne ilgiau kaip po 0,5 sekundės, kad nestabdytų visos sistemos darbo.
- Šifravimo raktų perdavimas tarp įtaisų turi užtikrinti raktų apsaugą.

[1] Tiek atminties turi eksperimentinėje sistemoje naudoti įtaisai.

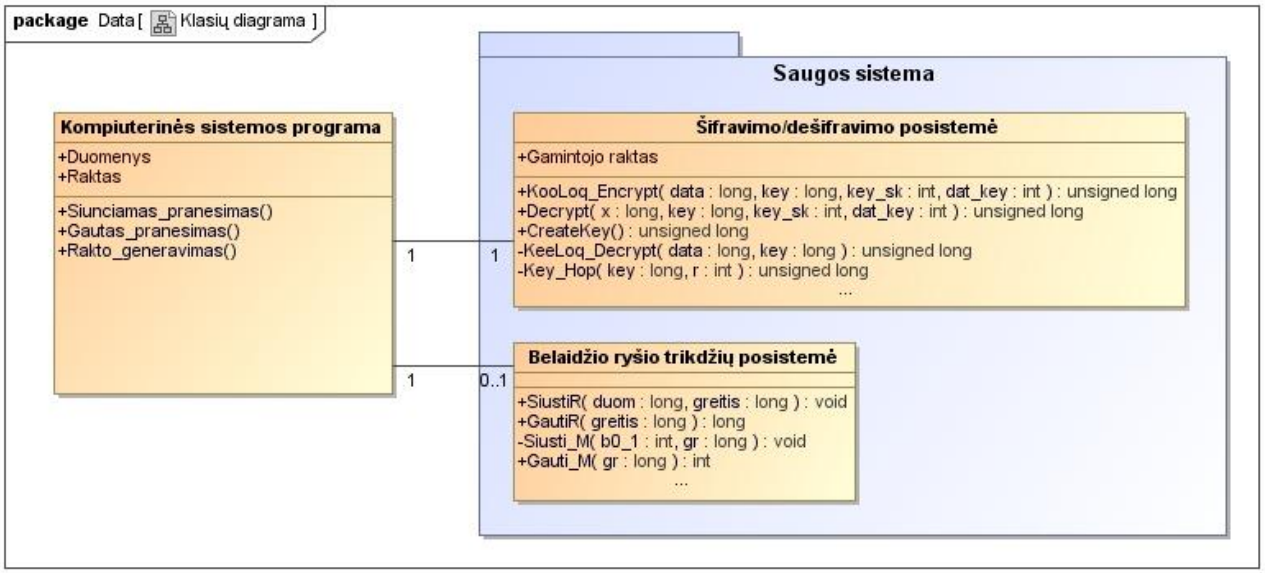
8.5. Reikalavimai vartotojo sąsajai

- Paprasta ir intuityviai valdoma.
- Turi suteikti galimybę inicijuoti naujų šifravimo raktų generavimą
- Turi realiu laiku pranešti apie belaidžio ryšio trikdžius, bei suteikti galimybę peržiūrėti jų istoriją.

9. Saugos sistemos projektavimas

9.1. Saugos sistemos architektūra

Projektuojamos sistemos komponentai ir jų ryšiai pavaizduoti klasių diagramoje (8 pav.).



8 pav. Projektuojamos būsto kompiuterinės sistemos saugos sistemos architektūra.

Diagramoje galima matyti, dalį būsto kompiuterinės sistemos, saugos sistemą su išskirtomis posistemėmis, bei ryšius tarp jų. Prie ryšių nurodytas kardinalumas rodo, kad į kompiuterinę sistemą visada bus įtraukta šifravimo posistemė, o belaidžio ryšio trikdžių posistemė bus įtraukta tik tuo atveju, jei sistemoje bus naudojamas belaidis ryšys.

9.2. Perduodamų duomenų šifravimo metodas

Tarp įtaisų siunčiamus duomenis bei komandas reikia apsaugoti nuo perėmimo ir suklastojimo, tam kad išspręsti įvade aprašytas saugumo problemas. Tačiau būsto kompiuterinėse sistemose naudojami įtaisai turi ribotus skaičiavimo pajėgumus ir nedidelę atmintį, o dauguma šifravimo algoritmų yra tam labai reiklūs.

9.2.1. Pasirinktas duomenų šifravimo algoritmas

Ieškant tinkamo duomenų šifravimo algoritmo, buvo peržvelgtos kelios publikacijos, kurių autoriai susidūrė su panašiomis šifravimo problemomis, kurios kyla ir šiame darbe [16],[17],[18],[19]. Buvo nuspręsta realizuoti tris skirtingus šifravimo algoritmus ir juos ištyrus pasirinkti tą, kuris tenkina iškeltus reikalavimus ir užtikrina didesnį saugumą. Atsižvelgiant į turimų įtaisų techninius duomenis buvo prieita išvados, kad asimetrinė kriptografija yra netinkama. Tačiau vis tiek nuspręsta realizuoti RSA algoritmą, kad būtų galima palyginti simetrinės ir asimetrinės kriptografijos skirtumus. Iš egzistuojančių simetrinės kriptografijos algoritmų, buvo pasirinktas AES algoritmas. Tai yra vienas plačiausiai naudojamų blokinio šifravimo algoritmų ir kol kas tyrėjams dar nėra pavykę pilnai jo „nulaužti“. Taip pat nuspręsta realizuoti KeeLoq šifravimo algoritmą, kuris iš pasirinktų algoritmų kelia mažiausius reikalavimus techninei įrangai ir yra greičiausias, bet tuo pačiu ir lengviau „nulaužiamas“.

9.2.2. Šifravimo raktų apsikeitimo metodas

RSA algoritmas naudoja skirtingus raktus informacijai užšifruoti ir jai dešifruoti, todėl viešojo rakto perėmimas nekeltų grėsmės, kad užšifruota informacija bus perskaityta. Tačiau yra galimybė, kad panaudojus šį raktą bus bandoma siųsti komandas sistemos įtaisui ir taip valdyti arba išjungti jį ar visą sistemą. Bet kadangi RSA algoritmas yra realizuojamas tik palyginimui su kitais dviem algoritmais, ir saugos sistemoje nebus naudojamas, todėl ir jo raktų saugos klausimas šiame darbe nebus nagrinėjamas.

Pasirinktas AES algoritmas naudoja vieną raktą informacijai užšifruoti ir dešifruoti. Todėl šifravimo raktas turi būti saugiai perduodamas tarp įtaisų. Tam buvo pasirinktas Diffie-Hellman‘o raktų apsikeitimo algoritmas. Jis leidžia dviem ar daugiau vartotojų susikurti bendrą slaptą raktą naudojant neapsaugotus komunikavimo kanalus. Šis algoritmas remiasi diskretinio logaritmo problema, todėl bus saugus tol, kol nebus pakankamai efektyvių metodų, išsprendžiančių šią problemą.

KeeLoq algoritmas turi savo raktų apsikeitimo būdų [20]. Šiame darbe naudojamas vienas iš saugesnių. KeeLoq raktų apsikeitimui reikalingas visiems sistemos įtaisams žinomas gamintojo raktas. Naujas raktas, skirtas duomenims šifruoti sudaromas iš įtaiso serijinio numerio ir sugeneruoto atsitiktinio skaičiaus. Galima sugeneruoti ir visą raktą. Šis raktas yra užšifruojamas su gamintojo raktu, naudojant KeeLoq algoritmą, ir išsiunčiamas kitam įtaisui. Kitas įtaisas, žinodamas gamintojo raktą, gali iššifruoti gautą raktą. Naudojant šį raktų apsikeitimą yra svarbu, kad gamintojo raktas išliktų paslaptyje. Panaudojus gamintojo raktą

yra išsprendžiamas vienas iš šio darbo uždavinių. Prie sistemos nebus galima prijungti kito gamintojo įtaisų, kurie neturi ar turi kitokį gamintojo raktą.

9.3. Sistemos įtaisų komunikavimas belaidžiu ryšiu ir reakcija į jo trikdžius

Apsaugos sistema turės posistemę, kuri bus atsakinga už komunikavimą belaidžiu ryšiu. Ji užtikrins perduodamų duomenų teisingumą ir praneš sistemai ir vartotojui, jei jie bus perduoti su klaidomis ar belaidis ryšys bus sutrikdytas.

9.4. Saugos sistemos vartotojo sąsaja

Saugos sistemai atskiros vartotojo sąsajos daryti nereikia, tačiau kuriant visos kompiuterinės sistemos vartotojo sąsają, reikia, kad ji tenkintu ir saugos sistemos keliamus reikalavimus. Saugos sistemos reikalavimuose vartotojo sąsajai buvo minėta, kad ji turi leisti inicijuoti naujų šifravimo raktų generavimą, bei suteikti galimybę peržiūrėti belaidžio ryšio trikdžių istoriją. Vartotojo sąsaja turi pranešti kai sutrinka belaidis ryšys. Šias funkcijas reikia integruoti į bendrą kompiuterinės sistemos vartotojo sąsają.

10. Eksperimentinė būsto kompiuterinė sistema

Tam, kad patikrinti saugos sistemos veikimą, buvo realizuota eksperimentinė būsto kompiuterinė sistema. Sistemoje naudojami įtaisų analizėje aptarti ir pasirinkti įtaisai su PIC16F877 mikroprocesoriumi ir kompiuteris. Matavimų duomenys šiai sistemai imami iš naudojamuose įtaisuose įtaisytų potenciometrų. Vienas iš įtaisų jungiamas prie kompiuterio laidais naudojant RS232 sąsają. Su juo radijo bangomis komunikuoja kitas įtaisas. Radijo ryšiui naudojami 433MHz siųstuvas ir imtuvas. Jie jungiami prie PIC16F877 įtaisų RE0 (duomenų išsiuntimas) ir RE1 (duomenų gavimas) kontaktų. Pirmasis, prie kompiuterio prijungtas įtaisas yra valdantysis ir valdo šalutinį įtaisą, siųsdamas jam komandas.

Naudojantis šia eksperimentine sistema bus ištirtas saugos sistemos veikimas ir įvertintas jos efektyvumas.

10.1. Funkciniai reikalavimai eksperimentinei būsto kompiuterinei sistemai

- Potenciometro matavimų apdorojimas ir išsiuntimas į kompiuterį ar kitą sistemos įtaisą.
- Galimybė komunikuoti per RS232 sąsają ir belaidžiu ryšiu.

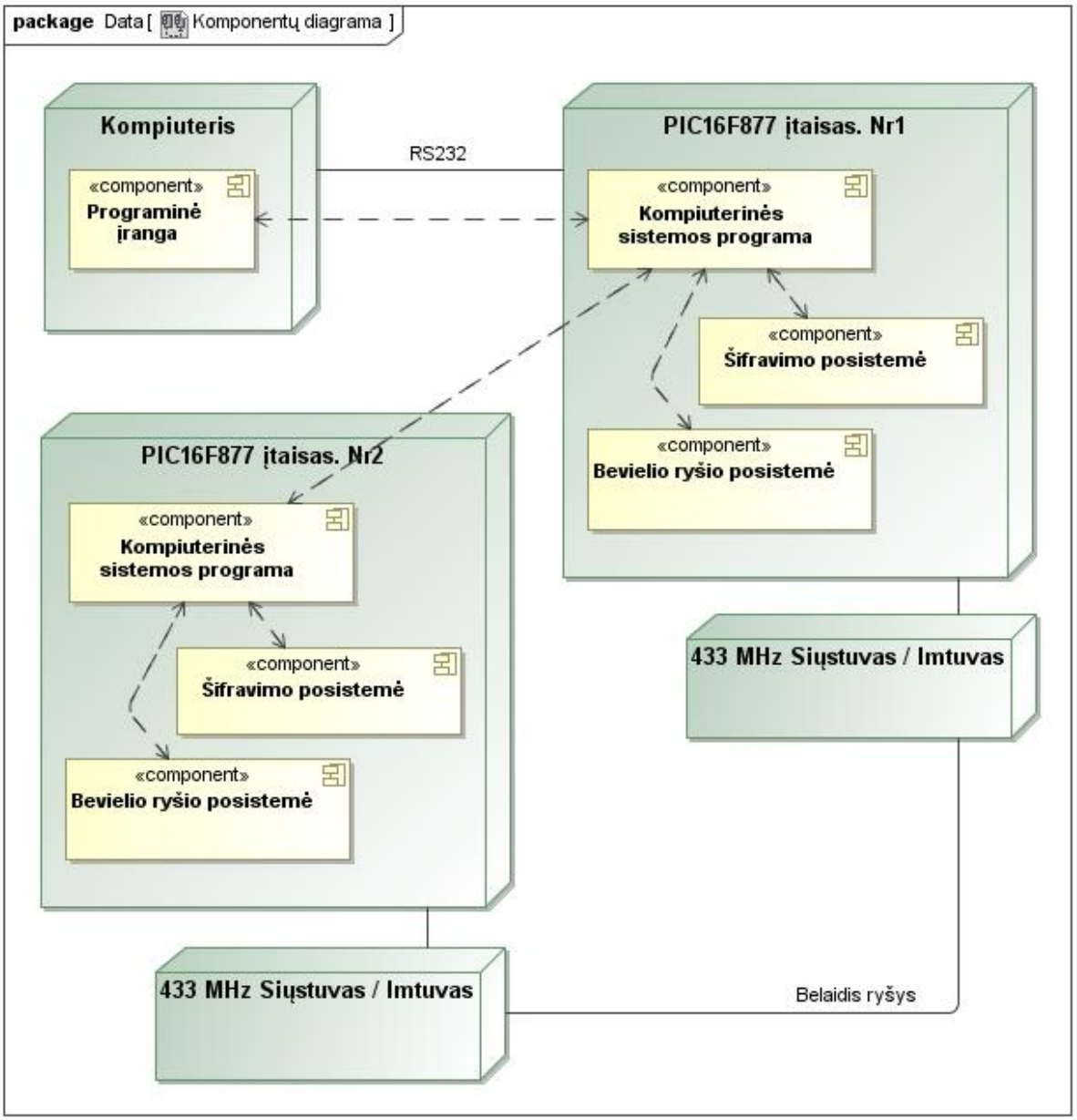
- Galimybė inicijuoti belaidžio ryšio trikdžius ir tuo pačiu stebėti kaip į juos reaguoja sistemos įtaisai.

10.2. Techniniai reikalavimai eksperimentinei būsto kompiuterinei sistemai

- Kompiuteris turi turėti galimybę susijungti su sistemos įtaisu per RS232 sąsają.
- Sistemoje naudojami įtaisai turi turėti galimybę komunikuoti radijo ryšiu.

10.3. Eksperimentinės sistemos architektūra

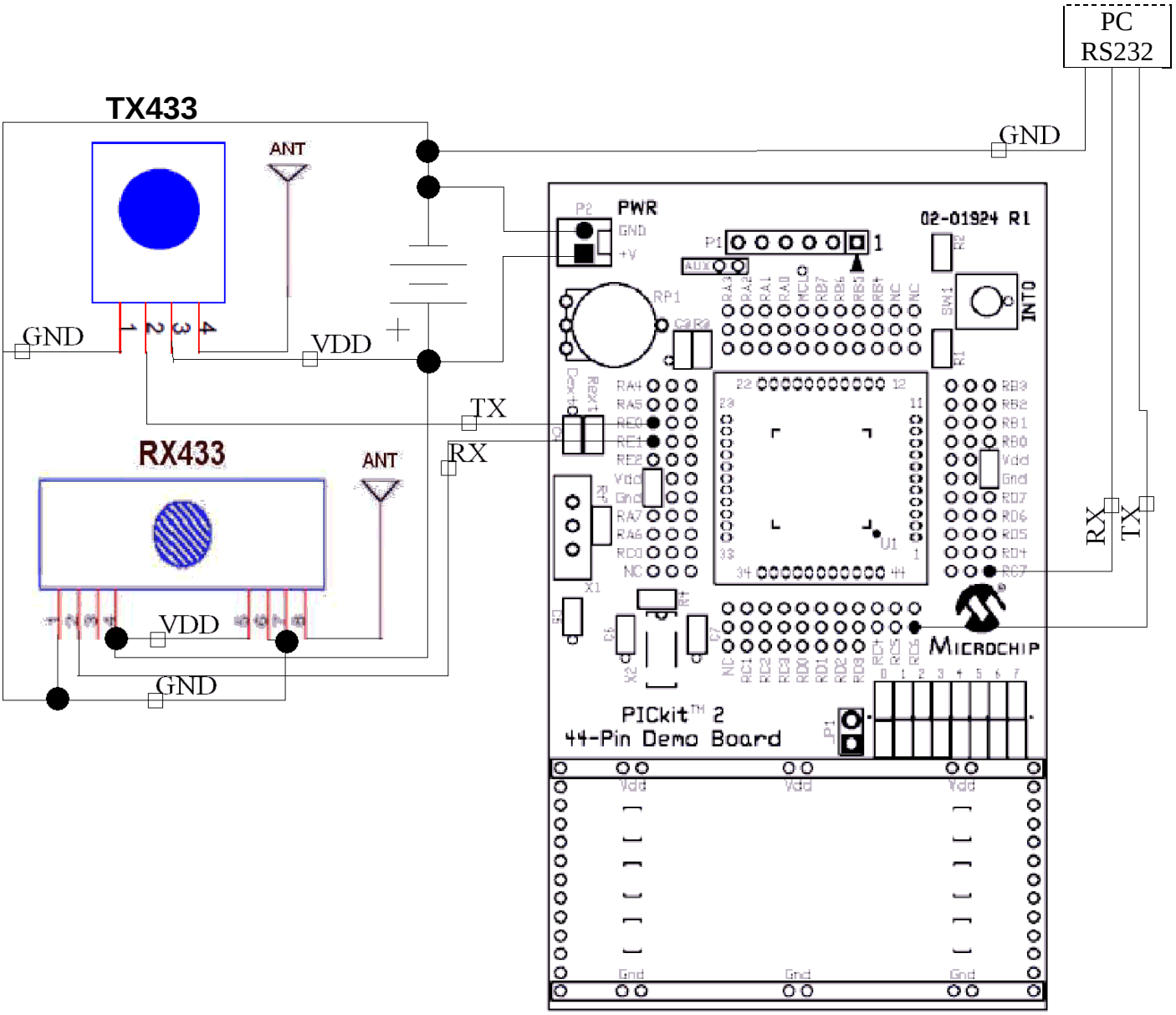
Eksperimentinės sistemos įtaisų ir jų komponentų ryšiai pavaizduoti komponentų diagramoje (9 pav.).



9 pav. Eksperimentinės būsto kompiuterinės sistemos architektūra.

Ištisinė linija diagramoje žymi kaip yra sujungiami įtaisai. Punktyrinė linija rodo kokie sistemos komponentai sąveikauja tarpusavyje.

Žemiau pateikta schema su PIC16F877 įtaisu, prie jo prijungtais siųstuvu ir imtuvu, bei pažymėtomis jungtimis, naudojamomis komunikavimui su kompiuteriu per RS232 sąsają.



10 pav. Eksperimentinės sistemos įtaiso ir naudojamų jo kontaktų schema[25][26].

10.4. Eksperimentinės sistemos veikimo aprašymas.

Pirmasis PIC16F877 įtaisas yra valdantysis, jis komunikuoja su šalutiniu įtaisu ir kompiuteriu. Šalutinis įtaisas yra budėjimo režime ir laukia iš valdančiojo įtaiso atsiunčiamos komandos. Gavęs komandą atlieka tai ko jo prašoma ir laukia kitos komandos. Jei pastebimas belaidžio ryšio sutrikimas, siunčiamas pranešimas valdančiajam įtaisui, bei įjungiami apie tai pranešantys diodai.

Naudojamų komandų ir pranešimų sąrašas:

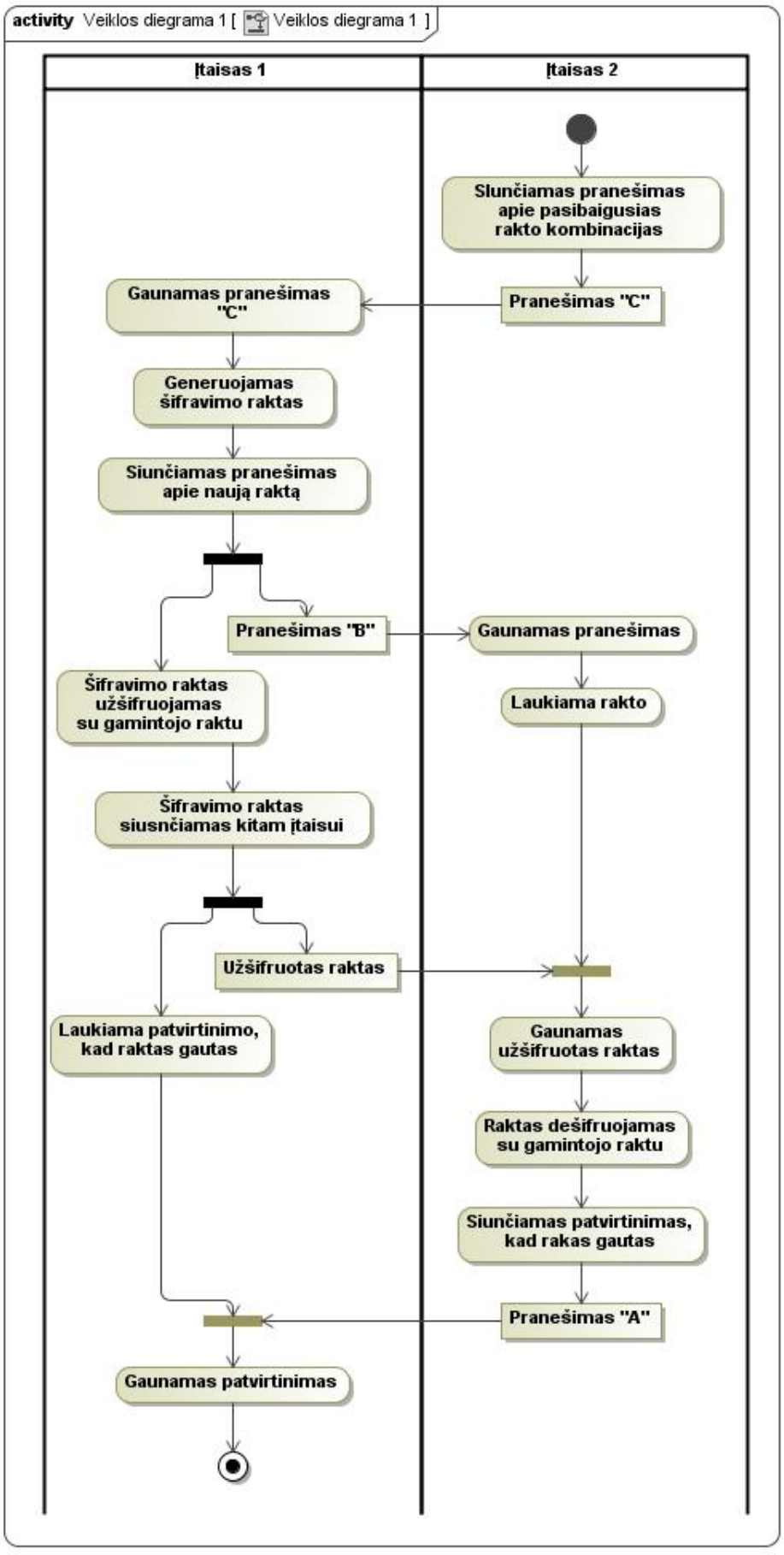
2 lentelė. Komandų ir pranešimų sąrašas.

Komanda / pranešimas	**Kodas (šešioliktaine forma)**
Naujo šifravimo rakto prašymas	1
Prašymas atsiųsti matavimų rezultatą	2
Pranešimas, kad duomenys priimti teisingai	9
Pranešimas apie gautą naują šifravimo raktą	A
Pranešimas apie siunčiamą naują šifravimo raktą	B
Pranešimas apie pasibaigusį šifravimo raktą	C
Pranešimas apie belaidžio ryšio sutrikimą	D
Pranešimas, kad duomenys neiššifruojami	E
Pranešimas, kad raktas nenusiųstas	F

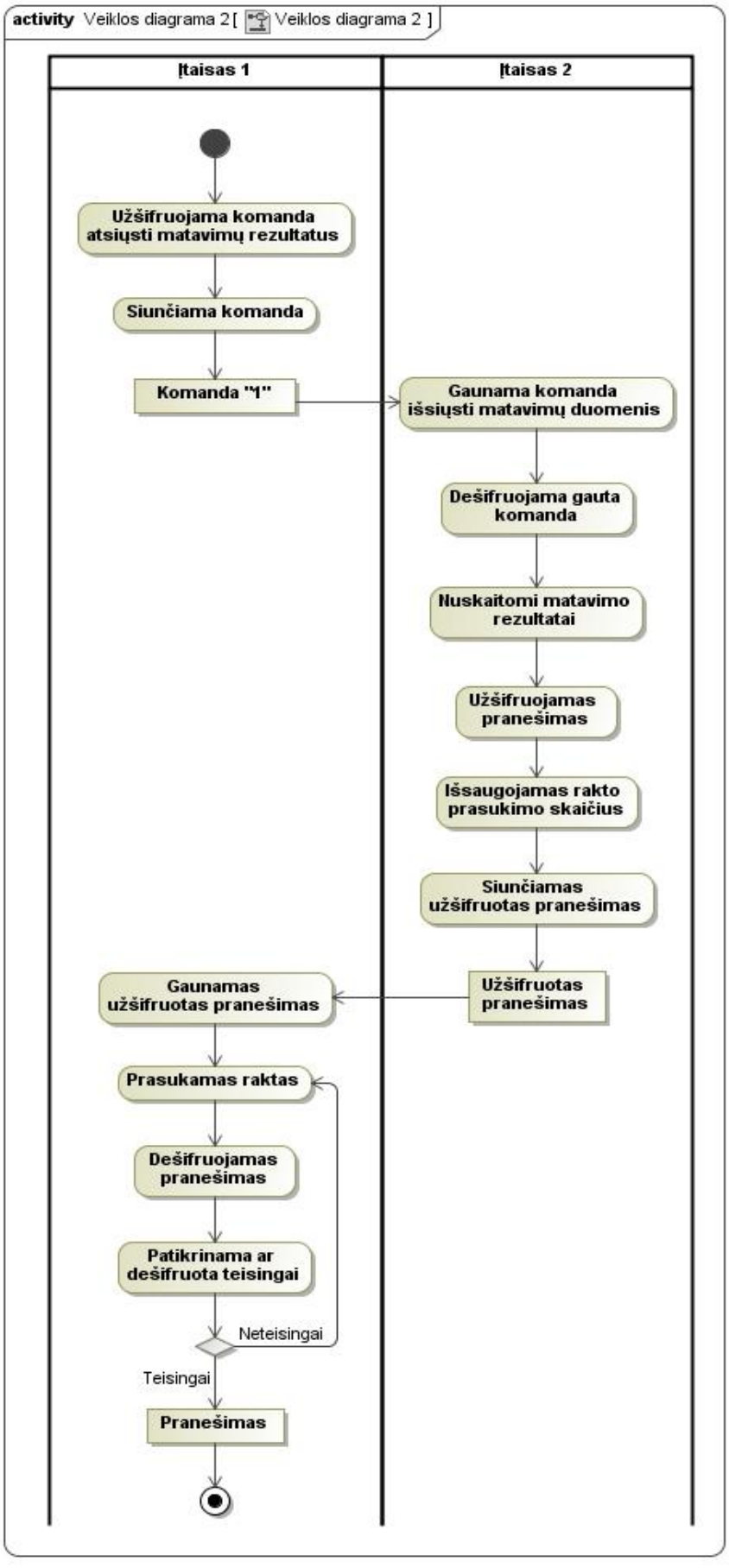
Komandos ir pranešimai koduojami keturiais bitais. Jų vieta siunčiamame pakete bus aprašyta 11.1.2 skyriuje aprašant KeeLoq šifravimo realizavimą ir pavaizduota 13 paveiksle.

10.5. Saugos sistemos ir eksperimentinės sistemos sąveika.

Žemiau pateiktose veiklos diagramose pavaizduota kaip bendrauja du įtaisai kai generuojamas naujas šifravimo raktas ir kai siunčiamas pranešimas. Naudojamas KeeLoq šifravimo algoritmas.



11 pav. Įtaisuose atliekami veiksmai, generuojant naują raktą.



12 pav. Įtaisuose atliekami veiksmai, siunčiant pranešimą.

11. Būsto kompiuterinės sistemos saugos sistemos realizacija

11.1. Šifravimo posistemė

11.1.1. Sistemos dalis, kuriai kuriama apsauga.

Prieš realizuojant duomenų šifravimo posistemę, buvo sukurta programa PIC16F877 įtaisui, kurios pagalba buvo testuojamas šifravimas. Ši programa atlieka matavimus, imdama duomenis iš PIC16F877 schemoje įmontuoto potenciometro ir versdama juos į skaitmeninę formą. Pagal gautus duomenis yra uždegami diodai. Šifravimo posistemės užduotis yra užšifruoti jai pateikiamus duomenis neužtrunkant ilgiau kaip 0,5 sekundės ir juos teisingai dešifruoti, taip pat greičiau kaip per 0,5 sekundės. Matavimų duomenys yra 8 bitų ilgio. Taip pat ir sistemos komandos neužima daugiau kaip 8 bitų. Todėl siunčiami pranešimai gali būti 8 bitų ilgio.

11.1.2. Šifravimo realizavimas

Realizuojant šifravimo posistemę buvo realizuoti trys šifravimo būdai. Tai RSA, AES ir KeeLoq šifravimai. Šie trys šifravimo būdai realizuoti norint juos palyginti ir nustatyti kuris iš jų geriausiai atitinka keliamus reikalavimus ir yra tinkamas naudoti PIC16F877 ir panašiuose įtaisuose.

RSA šifravimo realizavimas

Realizuojant RSA algoritmą PIC16F877 įtaisuose buvo susidurta su problema. Naudojant ilgesnius šifravimo raktus, įvykdavo perpildymas ir skaičiavimai būdavo klaidingi. Mažinant rakto ilgį buvo bandoma patikrinti su kokio ilgio raktais galima atlikti užšifravimą. Skaičiavimai teisingai buvo atlikti tik su 8 bitų ilgio raktu. Tačiau tokio ilgio raktas neužtikrina saugumo. RSA standarto puslapyje yra pateikiama rekomendacija naudoti ne trumpesnius kaip 512 bitų ilgio raktus [21].

Matuojant šifravimo ir dešifravimo laiką buvo naudojami 8 ir 14 bitų ilgio raktai (2 lentelė). Su trumpesniu raktu skaičiavimai buvo atlikti greitai ir tenkino iškeltus laiko reikalavimus. Tačiau buvo pastebėta, kad rakto ilgį padidinus du kartus, skaičiavimų laikas padidėja apie 100 kartų.

AES šifravimo realizavimas.

Realizuojant standartinį AES algoritmą, taip pat buvo susidurta su problema. Masyvai su 256 elementais yra per dideli, todėl logaritminių lentelių PICkit2 įtaise saugoti neišeina. Ši

problema buvo išspręsta nesaugant lentelių atmintyje, bet išskaičiuojant reikšmę kai jos prireikia. Tiesa, šie skaičiavimai sulėtina šifravimą.

Bandant šifravimo posistemę įrašyti į PIC16F877 atmintį, paaiškėjo, kad programai neužtenka ROM atminties. Taip pat šifravimui naudojant 128 bitų ilgio šifravimo raktus bei 128 bitų duomenų blokus, neužtenka ir RAM atminties.

Susidūrus su šiomis problemomis buvo mažinama programos apimtis, bei naudojamų duomenų ilgiai tol, kol šifravimo posistemė galėjo veikti PIC16F877 įtaise. Tačiau tai buvo pasiekta vykdant tik atskiras šifravimo posistemės dalis ir naudojant 17 bitų ilgio raktą, bei tokio pačio ilgio pranešimą. Kadangi sistemoje pranešimų ilgiai yra iki 8 bitų, tai sutrumpėjęs pranešimo ilgis nėra problema. Tačiau 17 bitų ilgio raktas yra nesaugus. Taip pat ir pats AES algoritmas negali būti panaudotas PIC16F877 įtaise, nes su juo programa užima perdaug vietos, kad tilptų į programai skirtą atmintį.

KeeLoq šifravimo realizavimas.

KeeLoq algoritmo privalumas yra tas, kad jo rakto bitai po kiekvieno atlikto šifravimo prasislenka, taip kiekvienas šifravimas atliekamas vis su kitokiu raktu. Tai padidina saugumą šifruojant ir su nedidelio ilgio raktu. Algoritme naudojami 64 bitų ilgio raktai ir 32 bitų ilgio duomenų blokai. Realizuojant KeeLoq šifravimą duomenų blokai buvo sumažinti iki 16 bitų. Taip buvo padaryta atsižvelgiant į tai, kad perduodami duomenys nėra ilgesni nei 8 bitai ir siunčiamos komandos ar pranešimai neilgesni kaip 4 bitai. Todėl didesni duomenų blokai tik labiau apkrautų sistemą. Rakto ilgis taip pat buvo sumažintas, nes naudojant ilgesnį nei 16 bitų ilgio raktą sudėtingėja programos kodas, taip pat didėja sunaudojamos atminties kiekis. Tokio ilgio raktas turi 16 skirtingų kombinacijų. Esant poreikiui, raktui išnaudojus visas 16 kombinacijų, galima sugeneruoti naują raktą. Tačiau dėl rakto saugumo nebuvo nuspręsta, šis sprendimas bus padarytas atlikus saugos sistemos tyrimą. Tam atvejui, jei 16 bitų ilgio raktas pasirodytų esąs nesaugus, buvo realizuota galimybė šifruoti naudojant 32 ir 64 bitų ilgio raktus nekeičiant pačio šifravimo algoritmo. Norint šifruoti 32 bitų ilgio raktu, sugeneruojami du 16 bitų ilgio raktai. Duomenys pirmiau užšifruojami vienu raktu, o gautas užšifruotas pranešimas dar kartą užšifruojamas antruoju raktu. Duomenų dešifravimas vyksta analogiškai. Naudojant 64 bitų ilgio raktą šifravimas vyksta taip pat, tik užšifruojama jau keturiais 16 bitų ilgio raktais.

Yra galimybė, kad dešifruojant gautą pranešimą bus bandoma dešifruoti naudojant ne tą rakto kombinaciją, kuria jis buvo užšifruotas ir dėl to neteisingai iššifruojami duomenys. Dėl to, į likusius tuščius duomenų bloko 4 bitus įrašoma konstanta, kuri yra žinoma visiems

įtaisams (13 pav.). Tai leidžia patikrinti ar gautas pranešimas yra teisingai dešifruotas. Jei aptinkama, kad jis dešifruotas neteisingai, pakeičiama rakto kombinacija ir vėl dešifruojama. Taip kartojama, kol pranešimas dešifruojamas teisingai.

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Konstanta, rakto patikrinimui				Komanda / pranešimas				Duomenys							

13 pav. Užšifruojamo duomenų bloko sandara.

Realizavus šifravimą ir išmatavus užšifravimo, bei dešifravimo greitį buvo įsitikinta, kad jis yra gerokai didesnis nei reikalaujama. Pranešimo užšifravimui ir dešifravimui naudojant 16 bitų ilgio raktus sugaištama po ~5ms (3 lentelė), o didinant šifravimo rakto ilgį dvigubai, taip pat ir šifravimo laikas pailgėja dvigubai. Jei gautą pranešimą bandoma dešifruoti su netinkama rakto kombinacija ir tenka patikrinti visas rakto kombinacijas, kol pranešimas teisingai dešifruojamas, su 64 bitų ilgio raktu tam sugaištama daugiausiai 0,09 sekundės. Trumpesnių raktų maksimalus dešifravimo laikas nuo šio skiriasi tik skiriasi tik keliomis tūkstantosiomis sekundės dalimis.

Eksperimentinė programa su realizuotu KeeLoq šifravimu ir raktų generavimu bei apsikeitimu, priklausomai nuo rakto ilgio, užima 52-72 procentų programai skirtos atminties (ROM) ir šifravimui sunaudoja iki 34 procentų darbinės atminties (RAM) (3 lentelė).

Realizuotų šifravimų palyginimas ir išvados

Trijų realizuotų šifravimo būdų palyginimas matomas žemiau pateitoje 3 lentelėje.

3 lentelė. Realizuotų RSA, AES ir KeeLoq šifravimų palyginimas.

	RSA		**AES [2] (17 bitų raktas)**	**KeeLoq (16 bitų raktas)**	**KeeLoq (32 bitų raktas)**	**KeeLoq (64 bitų raktas)**
	8 bitų raktai	**14 bitų raktai**				
Užšifravimo / dešifravimo laikas	0,06s / 0,09s	8s / 6s	2,27s	0.005s / maks. 0,084s	0,011s / maks. 0,087s	0,022s / maks. 0,09s
ROM / RAM atminties sunaudojimas	23% / 18%		> 100% / 80%	52% / 24%	63% / 31%	72% / 34%

Palyginus realizuotus šifravimo būdus buvo nuspręsta toliau darbe naudoti KeeLoq šifravimą. Tik šis būdas tenkino laiko ir techninės įrangos keliamus reikalavimus, tuo pačiu palikdamas galimybę naudoti raktus su rekomenduojamu 64 bitų ilgio raktu. Kokio ilgio šifravimo raktus naudoti bus nuspręsta atliekant saugos sistemos tyrimą.

[2] AES šifravimo laikas gautas sudėjus atskirų programos dalių veikimo laikus.

11.1.3. Raktų apsikeitimas

RSA ir AES algoritmai šiame darbe daugiau nebus naudojami, dėl to toliau bus realizuojamas tik KeeLoq šifravimui skirtas raktų generavimas ir apsikeitimas jais.

Naudojant KeeLoq šifravimą, kiekvienas įtaisas būsto kompiuterinėje sistemoje turi turėti vienodą gamintojo raktą. Šis raktas yra naudojamas užšifruoti naujai sugeneruotam raktui, kai jis persiunčiamas kitam įtaisui. Taip pat jis užtikrina, kad prie sistemos nebus prijungtas kito gamintojo įtaisas.

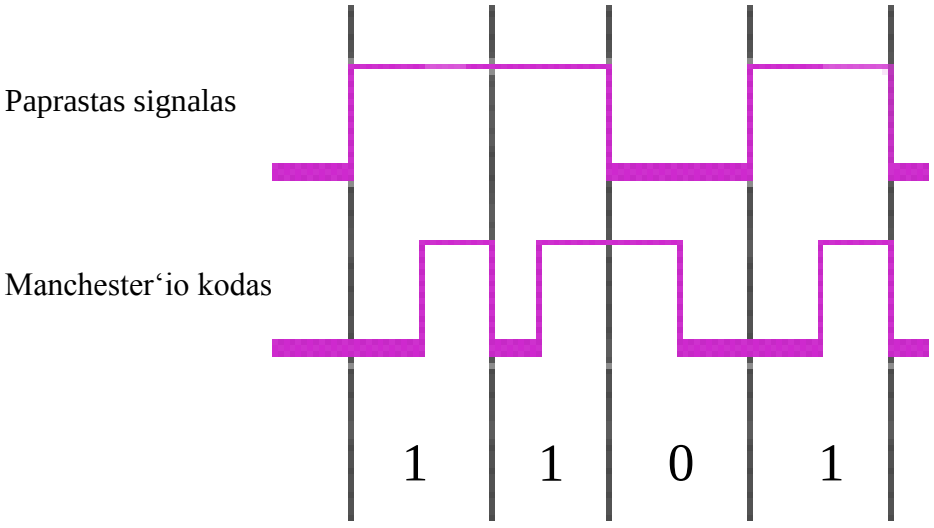
Kuriamoje sistemoje naują 16 bitų ilgio raktą sudaro keturi atsitiktinai sugeneruoti 4 bitų ilgio skaičiai. Naujas raktas yra užšifruojamas su gamintojo raktu naudojant tą patį KeeLoq algoritmą, kaip ir duomenų šifravimui. Užšifruotas raktas gali būti siunčiamas kitam įtaisui.

Šifravimui naudojant 32 bitų ilgio raktus, raktas sudaromas iš dviejų 16 bitų raktų. Pranešimai užšifruojami su vienu raktu, o gautas užšifruotas pranešimas dar kartą užšifruojamas su kitu raktu. Šis būdas leidžia nesunkiai padidinti ar sumažinti naudojamų raktų ilgį.

11.2. Belaidžio ryšio trikdžių posistemė.

Ši posistemė pirmiausia atsako už teisingą duomenų perdavimą belaidžiu ryšiu. Tam buvo sukurtas protokolas pagal kurį informaciją siunčiantis įtaisas ir ją priimantis įtaisas nustato siuntimo pradžią ir persiunčia nustatyto dydžio duomenų bloką. Protokole įdėta duomenų bloko kontrolinė suma leidžia patikrinti ar duomenys yra teisingai gauti.

Norint sumažinti triukšmo poveikį belaidžio ryšio kokybei, buvo realizuotas Manchester‘io kodas [24]. Šis kodas kiekvieną bitą koduoja dviejų bitų pora – 01 atitinka 1 ir 10 atitinka 0 (14 pav.). Tai sumažina tikimybę, kad triukšmas įtakos neteisingą pranešimo gavimą.

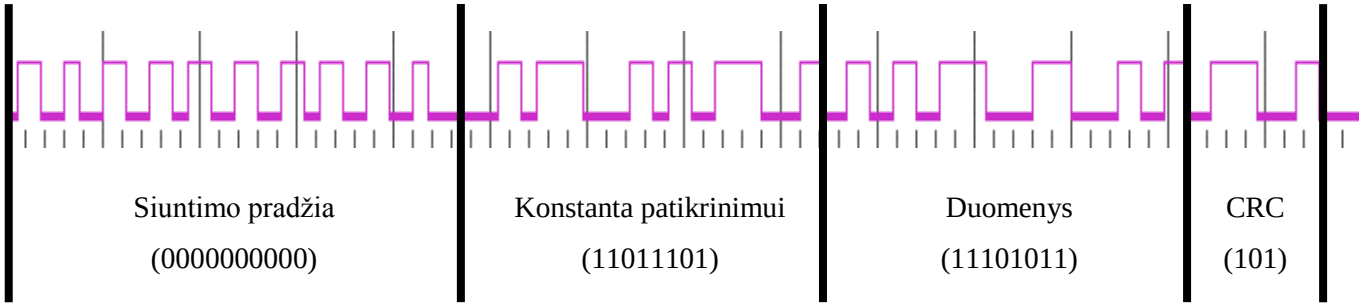


14 pav. Manchester‘io kodo pavyzdys.

Belaidžiu ryšiu siunčiamas paketas susideda iš 29 bitų, kuriuos galima suskirstyti į keturias dalis. Paketo pradžioje siunčiamas 10 bitų ilgio nulinis signalas, kurio trukmė yra 10ms. Šis signalas yra pastebimas priimančio įtaiso, kuris suskaičiavęs gautus bitus žino, kad tai yra siunčiamo paketo pradžia. Toliau siunčiama 8 bitų ilgio konstanta, žinoma siųstuvui ir imtuvui. Ji naudojama dar kartą patikrinti ar tai tikrai žinomo siųstuvo siųstas paketas. Trečioji paketo dalis yra 8 bitų ilgio duomenų blokas, o ketvirtoji – siunčiamų duomenų kontrolinė suma, kuri skirta patikrinimui ar duomenys gauti be klaidų. Patikrinimui naudojamas CRC (cyclic redundancy check - ciklišKas perteklinis patikrinimas) algoritmas.

Bendra siunčiamo paketo trukmė yra ~35 ms.

Žemiau pateiktame 15 paveiksle parodytas signalas siunčiant vieną paketą.



15 pav. Belaidžiu ryšiu siunčiamas paketas.

12. Būsto kompiuterinės sistemos saugos sistemos tyrimas

12.1. Šifravimo posistemės tyrimas

12.1.1. Žinomos KeeLoq atakos, jų galimas poveikis sistemai ir apsauga nuo jų.

a) Slapto pasiklausymo ir slopinimo ataka (Eavesdropping-and-jamming) [22]

Ši ataka naudojama norint įsilaužti į mašinas. Ją galima lengvai suprasti per pavyzdį. Tarkime, kad mašinos šeimininkas su nuotoliniu pulteliu bando užrakinti ar atrakinti savo mašiną. Tuo metu, netoliese esantis vagis su savo imtuvu priima radijo bangomis siunčiamą raktą ir parinkęs tinkamą dažnį su savo siųstuvu slopina šeimininko siunčiamą signalą, kad mašina jo nepriimtų. Jei tai pavyksta, jis turi veikiantį raktą, kurį galima bus panaudoti tik vieną kartą. Tačiau, jei po to šeimininkas dar bent kartą atrakintų ar užrakintų savo mašiną, vagies turimas raktas nebetektų prasmės.

Nagrinėjamoj sistemoj siunčiami ne vienkartiniai raktai, kuriuos perėmus vėliau galima patekti į sistemą, bet užšifruoti duomenys ar komandos. Ši ataka tiriamai sistemai įtaką galėtų padaryti tik tuo atveju, jei būtų perimta siunčiama komanda ir pastebėta kaip į ją sureaguoja sistema. Šią komandą nusiuntus antrą kartą, būtų iššaukta tokia pati sistemos reakcija. Šiuo atveju slopinimas nebūtų reikalingas. Tačiau turimas užšifruotas pranešimas su komanda nebetektų prasmės sugeneravus naują šifravimo raktą.

Realizuotoje šifravimo posistemėje nuo šios atakos apsisaugoma pakeičiant šifravimo raktą, kai pasibaigia galimos jo kombinacijos. Dėl to, perėmus pranešimą su komanda ir jį vėliau nusiuntus įtaisui, jis nebus teisingai dešifruotas. Dešifruodamas tokį pranešimą įtaisas bandys tai padaryti su visom likusiom rakto kombinacijom. Nepavykus dešifruoti bus siunčiamas pranešimas, kad reikia sugeneruoti naują šifravimo raktą.

Pasibaigus šifravimo rakto kombinacijoms dešifruoti pavyksta tik pranešimus apie tai, kad reikalingas naujas šifravimo raktas ir apie naujo rakto atsiuntimą.

Bandymo nusiųsti anksčiau išgautą komandą sistemos įtaisui rezultatas matomas žemiau esančioje 4 lentelėje.

4 lentelė. Bandymas nusiųsti anksčiau panaudotą užšifruotą komandą.

Skirtingų komandų siuntimų skaičius	Sistemos priimtų ir įvykdytų komandų skaičius	
	Neapsaugota sistema	Apsaugota sistema
10	10	0

Atlikus bandymą įsitikinta, kad realizuota sistema atmeta visas gautas komandas, kurios yra užšifruotos su jau panaudota raktų kombinacija.

b) Šoninio kanalo (Side-channel) ataka

Šios atakos remiasi informacija, gauta iš kriptografinės sistemos fizinės realizacijos, vietoj matematinių rakto išskaičiavimų ar teorinių algoritmo trūkumų ieškojimo. Naudinga informacija gali būti gaunama stebint įtaiso atliekamų veiksmų laikus, galios sunaudojimą, skleidžiamą elektromagnetinį lauką, ar netgi skleidžiamą garsą. Viena iš dažniausiai naudojamų šio tipo atakų yra galios analizės ataka.

Galios analizės ataka (Power analysis) [22][23]

Ši ataka yra paremta tuo, kad kiekvienas veiksmas mikroprocesoriuje užtrunka tam tikrą laiką ir pareikalauja tam tikros galios. Skirtingų operacijų sugaištamas laikas ir sunaudojama galia dažniausiai šiek tiek skiriasi. Atakuotojas, naudojantis šį būdą, stebi ir analizuoja atakuojamos sistemos įtaiso galios sunaudojimą. Iš surinktų duomenų galima brėžti galios sunaudojimo laike kreivę ir ją analizuoti vizualiai. Kreivėje galima pastebėti pasikartojimus, kuriuos galima susieti su viena ar kita operacija. Naudojant šią ataką galima iš sistemos išgauti naudojamą šifravimo raktą ar kitą slaptą informaciją. Galios analizės ataka ypač veiksminga tuomet, kai KeeLoq algoritmas yra realizuotas techninėje įrangoje, nes tuomet konkreti operacija kiekvieną kartą užtrunka tiksliai tokį patį laiką ir sunaudoja tiek pat galios.

Tokio tipo atakos tiriamai kompiuterinei sistemai gali būti pavojingos, jei įsilaužėlis gali patekti prie sistemos įtaisų ir juos stebėti. Kadangi šios atakos sėkmė priklauso daugiau nuo būsto kompiuterinės sistemos techninės įrangos, nei nuo šifravimo algoritmų parinkimo, tai šiame darbe nenagrinėjami apsaugos nuo šios atakos būdai. Ši ataka paminėta dėl to, kad diegiant sistemą būtų atkreiptas dėmesys į sistemos įtaisų prieinamumą pašaliniams asmenims.

c) Atakos, paremtos matematiniais skaičiavimais

Yra naudojamos kelios technikos išanalizuoti KeeLoq šifravimą pasitelkiant matematinius skaičiavimus, tačiau apžvelgtuose darbuose rašoma, kad šifravimui naudojant 64 bitų ilgio raktą iššifruoti pranešimą užtrunka mažiausiai 3 valandas [20], o gali užtrukti ir iki kelių dienų. Taip pat šie skaičiavimai reikalauja didelių skaičiavimo pajėgumų. Daugeliui reikia atlikti apie 2^{40} – 2^{50} KeeLoq užšifravimų. Dar vienas matematiniais skaičiavimais paremtų atakų trūkumas yra tas, kad norint pasiekti anksčiau paminėtą užšifravimų skaičių, joms yra reikalingas didelis užšifruotų ir dešifruotų pranešimų porų skaičius. Net ir efektyviausiems metodams reikia atlikti $2^{44,5}$ užšifravimus ir žinoti 2^{16} užšifruotų ir nešifruotų pranešimų porų[23].

Realizuotoje sistemoje šifravimo raktai yra keičiami taip dažnai, kad surinkti tokį skaičių su vienu raktu užšifruotų ir dešifruotų pranešimų yra neįmanoma. Net naudojant 64 bitų ilgio raktus, vienu raktu užšifruotų pranešimų gali būti tik 64. Tačiau yra galimybė panaudojus tokias atakas išsiaiškinti gamintojo raktą, kuriuo yra užšifruojami nauji šifravimo raktai. Kadangi gamintojo raktas yra pastovus, jį išsiaiškinęs įsilaužėlis galėtų perimti sistemos valdymą ar prie jos prijungti sistemai nepriklausančius įtaisus. Dėl to buvo nuspręsta ištirti šią galimybę ir įvertinti kiek sistema yra atspari šiai atakai.

Pirma problema iškylanti įsilaužėliui yra gauti užšifruotų ir neužšifruotų šifravimo raktų porų. Surinkti didelį skaičių užšifruotų raktų būtų sunku, nes nėra žinoma ar sugautame pakete yra siunčiamas gamintojo raktu užšifruotas raktas ar paprastu raktu užšifruotas pranešimas. Tuo tarpu gauti neužšifruotų raktų stebint vien siunčiamus paketus būtų neįmanoma. Tam reiktų pasitelkti b) punkte aprašytas atakas. Buvo padaryta prielaida, kad įsilaužėlis gauna vieną užšifruotą ir neužšifruotą šifravimo raktą. Turėdamas tik vieną raktą jis gali bandyti visais įmanomais gamintojo rakto variantais užšifruoti turimą šifravimo raktą ir tikrinti kada gautas užšifruotas raktas sutampa su turimu užšifruotu raktu. Tyrime tokiu pačiu būdu buvo mėginta sužinoti gamintojo raktą.

Tirti 16, 32, 48 ir 64 bitų ilgio raktai. Buvo matuota per kiek laiko sužinomas gamintojo raktas naudojant skirtingo ilgio raktus. Skaičiavimai atlikti dvejuose skirtingo galingumo kompiuteriuose, bei eksperimentinėje sistemoje naudojamame įtaise. Žemiau esančioje lentelėje pateikti tyrimo rezultatai.

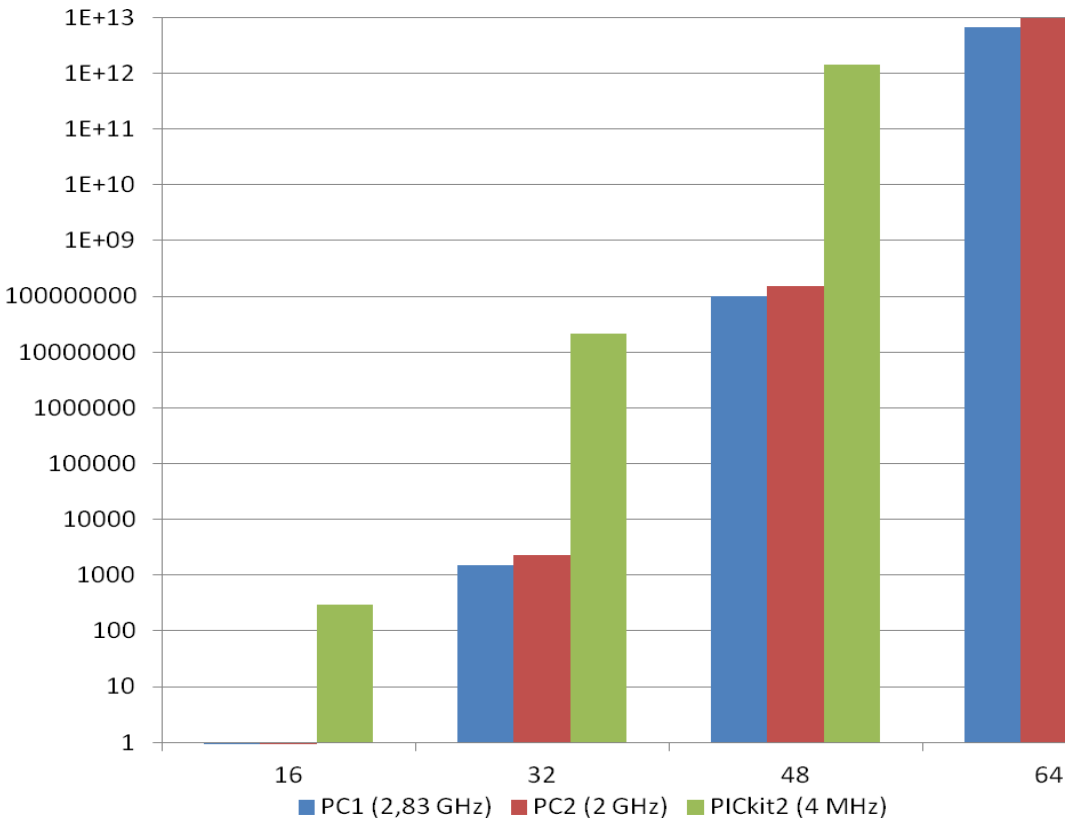
5 lentelė. Matematiniais skaičiavimais grįstos atakos rezultatai[3].

Rakto ilgis	16 bitai	32 bitai	48 bitai	64 bitai
KeeLoq šifravimų skaičius	2^{16}	2^{32}	2^{48}	2^{64}
PC1 (2,83 GHz)	0,023 s.	1526 s.	3,2 metai	207829,8 metai
PC2 (2 GHz)	0,032 s.	2294 s.	4,8 metai	312425,6 metai
PICkit2 (4 MHz)	298 s.	21362875 s.	44394,9 metai	2909463770,7 metai

Pagal tyrimo rezultatus nubrėžus grafiką aiškiai matoma, kaip sėkmingos atakos laikas yra priklausomas nuo raktų ilgio. Grafikas pateiktas 16 paveiksle. Atvaizduojant rezultatus buvo naudojama logaritminė y ašies skalė, nes paprastame grafike būtų matomi tik paskutinio

[3] 4 lentelėje pirmuose dvejuose stulpeliuose pateikti duomenys gauti atliekant realius skaičiavimus (išskyrus PICkit2 įtaiso antrąjį skaičių), o kiti skaičiai gauti atlikus teorinius skaičiavimus.

stulpelio duomenys. PICkit2 įtaiso paskutinė reikšmė nebuvo įtraukta į grafiką, nes yra labai didelė.



16 pav. Rakto ilgio ir sėkmingos atakos laiko priklausomybė.

Iš pateiktų tyrimo rezultatų matoma, kad naudojant 16 bitų ilgio raktą, gamintojo raktas gali būti surastas per kelias šimtąsias sekundės dalis, jei skaičiavimai atliekami kompiuteriu. Skaičiavimus atliekant viename iš sistemos įtaisų, gamintojo raktui surasti užtrunkama apie 5 minutes. Naudojant tokio ilgio šifravimo raktus šią ataką galima taikyti šifravimo raktams gauti, nes rakto radimo laikas yra trumpas ir jį bus galima naudoti, kol sistema pakeis raktus. Be to, šifravimo rakto išgavimas iš sistemos yra paprastesnis, nes siunčiamas užšifruotas komandas yra paprasčiau pastebėti stebint sistemos veikimą, taip pat, jas nesunku atspėti žinant, kad joms yra skirti tik 4 bitai. Taip turint užšifruotą ir nešifruotą komandą per kelias sekundės dalis galima įsilaužti į sistemą. Dėl to 16 bitų ilgio ratas bus laikomas nesaugiu ir nebus naudojamas realizuotoje sistemoje.

Naudojant 32 bitų ilgio raktą gamintojo ar šifravimo raktas gali būti surastas maždaug per pusvalandį, skaičiavimus atliekant kompiuteriu. Neatsižvelgiant į tai, kad neužšifruotą šifravimo raktą gauti yra labai sunku, bus laikoma, kad 32 bitų ilgio gamintojo raktas yra nesaugus. Tokio ilgio šifravimo raktus jau būtų galima naudoti, tačiau atsižvelgus į tai, kad tyrime naudota ataka yra viena paprasčiausių ir yra gerokai efektyvesnių, buvo nutarta šio rakto taip pat atsisakyti.

Remiantis tyrimo rezultatais nutarta realizuotoje šifravimo posistemėje naudoti 64 bitų ilgio raktus. Šifravimo raktų ilgis galėtų būti ir 48 bitų, nes tyrimo rezultatai rodo, kad tokio ilgio raktas naudojant paprastus skaičiavimus surandamas per 3-4 metus. Kadangi šifravimo raktai sistemoje gali būti pakeičiami net keletą kartų per dieną (tai priklauso nuo to kaip dažnai įtaisai tarpusavyje bendrauja), tai toks rakto radimo laikas sistemai grėsmės nekelia. Tačiau, atsižvelgus į nedidelį užimamos atminties ir šifravimui reikalingo laiko skirtumą, buvo pasirinktas ilgesnis ir tuo pačiu saugesnis raktas.

12.1.2. Šifravimo posistemės tyrimo apibendrinimas

Ištyrus šifravimo posistemę buvo nuspręsta naudoti 64 bitų ilgio raktus. Tyrimo rezultatai rodo, kad tokio ilgio gamintojo raktas patikimai apsaugo sistemą. Taip pat, tokio ilgio šifravimo raktai užtikrina siunčiamų duomenų ir pranešimų konfidencialumą ir vientisumą, nes jų radimo laikas yra nepalyginamai didesnis, jei jų naudojimo laikas.

Naudojant ankstesnio skyriaus a) punkte aprašytą ataką buvo bandyta siųsti sistemai anksčiau perimtas užšifruotas komandas. Gauti rezultatai parodė, kad neapsaugota nuo šios atakos sistema sureaguoja į visas jai nusiųstas komandas, o apsaugota sistema atmeta 100 procentų visų komandų.

Punkte b) minimos šoninio kanalo atakos pavojus tiriamai sistemai nebuvo vertintas, nes jos sėkmė priklauso ne nuo šifravimo algoritmo programinės realizacijos, bet nuo sistemoje naudojamos techninės įrangos, bei jos prieinamumo pašaliniams asmenims.

12.2. Belaidžio ryšio trikdžių posistemės tyrimas

Sutrikus belaidžiam ryšiui, jį naudojantys įtaisai nebegali gauti ar perduoti komandų, pranešimų, duomenų ir naujų šifravimo raktų. Dėl to toliau bus įvardinti belaidžio ryšio trikdžiai, galintys padaryti įtaką sistemos veikimui ar jos saugumui. Tyrimo metu bus sudarytos sąlygos kiekvienam iš aprašytų trikdžių atsirasti ir stebima sistemos reakcija į jį. Tai padės įvertinti belaidžio ryšio posistemės efektyvumą.

12.2.1. Galimi belaidžio ryšio trikdžiai ir sistemos atsakas į juos

Belaidžio ryšio slopinimas.

Tokį belaidžio ryšio sutrikimą tiesiogiai pastebės įtaisas, kuris belaidžiu ryšiu valdo kitus įtaisus. Jei jo siunčiami paketai su komandomis ar kitų įtaisų, jam atgal siunčiami paketai su duomenimis ar pranešimais bus slopinami, jis išsiuntęs komandą nesulauks atsakymo. Sistemoje, kurioje šis trikdis neįvertintas, įtaisas išsiuntęs komandą ir nesulaukiantis

atsakymo, pasiliktų laukimo būsenoje taip sustabdydamas sistemos ar jos dalies darbą. Realizuotoje belaidžio ryšio posistemėje numatyta, kad įtaisas išsiuntęs komandą, atsakymo laukia tam tikrą laiką. Jei per jį atsakymas negaunamas, siuntimas kartojamas. Jei per tris kartus atsakymas negaunamas, įtaisas praneša apie belaidžio ryšio trikdį kompiuteriui išsiunčiamu pranešimu, bei uždegdamas diodus.

Tam, kad būtų ištirta sistemos reakciją į šį trikdį, eksperimentinės sistemos įtaisui, kuris gauna komandas iš valdančio įtaiso ir siunčia jam atsakymus, buvo atjungtas siųstuvas. Tai sudarė aukščiau aprašytą situaciją. Iš 6 lentelėje pateiktų rezultatų matoma, kad sistemoje, kurioje šis trikdis neįvertintas, visais 10 bandytų atvejų sustoja darbas. Tuo tarpu apsaugotoje sistemoje nei vienu atveju darbas nebuvo sustojęs. Sutrikus ryšiui įtaisas nutraukė paketo laukimą ir apie tai pranešė.

Suklastotas paketas

Iš kito, sistemai nepriklausančio įtaiso išsiunčiamas paketas, kurio pradeda klausyti sistemos įtaisai. Čia galimi du atvejai. Pirmu atveju siunčiantis įtaisas naudoja kitokią paketo struktūrą, nei sistemos įtaisai. Tada sistemos įtaisas suskaičiuos paketo pradžioje esančius nulius, bei patikrins ar po jų einantys bitai atitinka sistemos įtaisams žinomą konstantą. Jei po šių patikrinimų bus nustatyta, kad tai nėra sistemoje naudojamas paketas, jis bus atmestas. Antru atveju, siunčiantis įtaisas išsiunčia paketą, kuris yra tokios pačios struktūros, kaip ir sistemoje naudojami paketai. Šią struktūrą nesunku sužinoti perėmus kelis tarp sistemos įtaisų siunčiamus paketus. Tokį paketą gavęs įtaisas atpažįsta jį ir juo atsiųstus duomenis perduoda šifravimo posistemei dešifruoti. Tolesnis patikrinimas atliekamas šifravimo posistemėje. Atsiųsti duomenys gali būti išgalvoti arba perimti iš anksčiau tarp sistemos įtaisų siųstų paketų. Ir vienu ir kitu atveju jie nebus teisingai dešifruoti su likusiomis šifravimo rakto kombinacijomis ir jie nebus laikomi teisingi. Siekiant išvengti to, kad po tokio paketo gavimo nebūtų be reikalo generuojamas naujas šifravimo raktas gautus duomenis toliau bandoma dešifruoti su jau panaudotomis rakto kombinacijomis. Jei dešifravimas pavyksta, vadinasi raktą reikia keisti, nes įsilaužėlis turi perėmęs pranešimų užšifruotų su šiuo raktu ir gali juos panaudoti įsilaužimui į sistemą. Jei dešifravimas nepavyksta, rakto keisti nereikia ir iki tol buvusias nepanaudotas rakto kombinacijas toliau galima naudoti pranešimų šifravimui.

Apie šią ataką taip pat pranešama pranešimu kompiuteriui, bei uždegamais diodais.

Tiriant sistemos reakciją į suklastotus paketus sistemos įtaisui buvo susti 5 paketai su bereikšmiais duomenimis ir 5 anksčiau iš sistemos išgauti paketai. Visi paketai buvo tokios struktūros, kaip ir naudojami sistemoje. Kitokios struktūros paketų siuntimas būtų panašus į

sistemos kurtinimo ataką. Kadangi ši ataka yra analogiška aprašytai 11.1.1 a) punkte, kur buvo tiriamas šifravimo posistemės atsparumas atakoms, tai ir rezultatai yra tokie patys. Realizuotoje sistemoje šią ataką atpažįsta ir atremia šifravimo posistemė.

Kurtinimas

Šios atakos metu iš sistemai nepriklausančio įtaiso be sustojimo siunčiami bereikšmiai duomenys. Taip paketo laukiantis įtaisas negali nustatyti kada siunčiamas paketas. Sistemos reakcija į šią ataką tokia pati, kaip ir signalo slopinimo atveju. Dėl to ir sprendimas naudojamas toks pats. Iš tyrimo rezultatų matome, kad abiem atvejais sistemos reakcijos yra labai panašios. Tiriant šią ataką pasitaikė keli atvejai, kai siunčiamą paketą pavyko sėkmingai pagauti.

Tyrimo rezultatai

6 lentelė. Belaidžio ryšio posistemės tyrimo rezultatai.

Belaidžio ryšio trikdis	**Mėginimų sutrikdyti ryšį skaičius**	**Sistemos reakcija**	
		Neapsaugota sistema	**Apsaugota sistema**
Ryšio slopinimas	10	Laukimas x10	Pranešimas x10
Suklastotas paketas	10	Gautas paketas x10	Pranešimas x10
Kurtinimas	10	Laukimas x9 Gautas paketas x1	Pranešimas x8 Gautas paketas x2

12.2.2. Belaidžio ryšio trikdžių posistemės tyrimo apibendrinimas

Belaidžio ryšio sutrikimas neišvengiamai atsiliepia sistemos darbui, nes nuslopinus signalus, sugadinus siųstuvą ar kitaip sutrikdžius ryšį, nutrūksta komunikacija tarp sistemos įtaisų. Realizuojant belaidžio ryšio posistemę siekta išvengti situacijų, kai sutrikus ryšiui sistemos įtaisai nebežino kaip elgtis ir lieka laukti. Atlikus tyrimą, buvo įsitikinta, kad aukščiau aprašytos atakos neįveda apsaugotos sistemos į nenumatytą situaciją. Įtaisai visais tirtais atvejais atpažįsta belaidžio ryšio sutrikimą ir apie tai praneša pranešimu, jei turi tam galimybę, ir diodais, įtaisytais juose.

13. Išvados

- Išanalizavus rinkoje siūlomas būsto kompiuterines sistemas, bei jose naudojamus duomenų mainų protokolus, buvo nustatyta, kad tokiose sistemose neužtikrinamas perduodamų duomenų konfidencialumas ir vientisumas.

    - Atlikus būsto kompiuterinėse sistemose naudojamų įtaisų analizę nustatyta, kad šie įtaisai turi ribotus skaičiavimo ir atminties resursus. Dažniausiai naudojami 8-16 bitų mikroprocesoriai ir iki 128Kb programai skirtos, bei iki 10 Kb darbinės atminties. Eksperimentinės sistemos realizavimui buvo pasirinktas PIC16F877 įtaisas. Tai leis ištirti ar realizuotas apsaugos metodas yra tinkamas įtaisams su 8 bitų mikroprocesoriumi ir turintiems ne daugiau kaip 8 Kb programai skirtos atminties, bei 368 b darbinės atminties.

    - Tarp sistemos įtaisų siunčiamų duomenų konfidencialumo ir vientisumo užtikrinimui buvo pasirinktas KeeLoq šifravimo algoritmas, jis vienintelis iš bandytų šifravimo algoritmų telpa į 8 Kb įtaiso atmintį, tuo pačiu užtikrindamas didžiausią saugumą. Jis leidžia naudoti 64 bitų ilgio raktą, tuo tarpu AES ir RSA šifravimo algoritmai naudojamame įtaise gali šifruoti ne didesniu kaip 17 bitų raktu.

    - Eksperimentiškai ištyrus realizuotą sistemą, buvo nustatyta, kad 16 bitų ilgio raktą galima surasti per 0,02 sekundės naudojant matematinius skaičiavimus, o 64 bitų ilgio raktas tokiu būdu būtų ieškomas keliasdešimt ar net kelis šimtus tūkstančių metų. Naudojant KeeLoq algoritmą siunčiami duomenys užšifruojami per 0,02 sekundės, o jų dešifravimas užtrunka 0,02 – 0,09 sekundės.

- Realizuojant sistemos apsaugą nuo jos darbo sutrikdymo, sutrikdžius jos įtaisų naudojamą belaidį ryšį eksperimentinėje būsto kompiuterinėje sistemoje buvo realizuota belaidžio ryšio posistemė, kurioje numatyti galimi belaidžio ryšio trikdžiai ir nustatyta kaip sistema turėtų į juos reaguoti. Atliekant šios posistemės tyrimą, buvo bandoma sutrikdyti belaidį ryšį su kiekviena ataka po 10 kartų. Įsitikinta, kad visos aprašytos atakos apsaugotoje sistemoje yra atpažįstamos ir sistema į jas atitinkamai sureaguoja. Tuo tarpu neapsaugotoje sistemoje visais atvejais, sutrikus belaidžiam ryšiui, sustodavo sistemos

darbas. Tyrimo rezultatai rodo, kad realizuota posistemė apsaugo sistemą nuo galimų belaidžio ryšio trikdžių.

- Sistemos apsauga nuo galimybės prie jos prijungti jai nepriklausančius įtaisus buvo realizuota įgyvendinant KeeLoq šifravimą. KeeLoq algoritme numatytas raktų apsikeitimas tarp sistemos įtaisų, kurio metu persiunčiamas raktas yra užšifruojamas gamintojo raktu. Būtent šis gamintojo raktas ir yra siūlomas sprendimas, nes jis yra žinomas tik vieno gamintojo įtaisams. Įtaisai, nežinantys šio rakto, negalės komunikuoti su sistemos įtaisais.

Literatūros sąrašas

[1] **Chan M., Esteve D., Escriba C., Campo E.** A review of smart homes – Present state and future challenges.
Computer Methods and Programs in Biomedicine,
2008, 91, 55-81p.

[2] **Robles R. J., Tai-hoon K.** A Review on Security in Smart Home Development.
International Journal of Advanced Science and Technology,
2010, 15, 13-22p.

[3] **Joseph A., Bong D.B.L., Mat D.A.A.** Application of Neural Network in User Authentication for Smart Home System.
International Journal of Social Sciences,
2010, 5:3, 146-153p.

[4] **Johnson M., Stajano F.** Usability of Security Management: Defining the permissions of Guests.
Security Protocol, 14 International Workshop,
Cambridge, 2006, 176-183p.

[5] **Jurcut A., Coffey T., Dojen R., Gyorodi R.** Security Protocol Design: A Case Study Using Key Distribution Protocols.
Journal of Computer Science & Control Systems,
2009, 2:2, 16-21p.

[6] „Atlas“ mazgo aprašymas, [žiūrėta 2010.12.22],
prieiga per Internetą: <http://www.pervasa.com/AHP.php>;

[7] „PICkit 2“ mazgo aprašymas, [žiūrėta 2010.12.22],
prieiga per Internetą:
<http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en023805>;

[8] „TelosB“ mazgo aprašymas, [žiūrėta 2010.12.22],
prieiga per Internetą: <http://www.willow.co.uk/html/telosb_mote_platform.html>;

[9] „iMote2“ mazgo aprašymas, [žiūrėta 2010.12.22],
prieiga per Internetą: <http://docs.tinyos.net/index.php/Imote2>;

[10] S232 protokolo aprašymas, [žiūrėta 2010.12.22],
prieiga per Internetą: <http://www.lammertbies.nl/comm/info/RS-232_specs.html>;

[11] UART protokolo aprašymas, [žiūrėta 2010.12.22],
prieiga per Internetą:
<http://www.senseair.se/TechNotes/comprot%200700xx%20rev%203_04.pdf>;
[12] I^2C ir SPI protokolų aprašymas, [žiūrėta 2010.12.22],
prieiga per Internetą:
<http://www.byteparadigm.com/kb/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html>;
[13] „C-Bus“ sistemos aprašymas, [žiūrėta 2011.01.25],
prieiga per Internetą: <http://www.cbus-enabled.com/what-cbus.htm>
[14] „X-10“ sistemos aprašymas, [žiūrėta 2011.01.25],
prieiga per Internetą:
<http://www.smarthouse.lt/Default.aspx?Element=ViewArticles&TopicID=73>
[15] „KNX“ sistemos aprašymas, [žiūrėta 2011.01.25],
prieiga per Internetą: <http://www.knx.org/knx-standard/introduction>
[16] **Gaubatz G., Kaps J. P., Sunar B.** Public Key Cryptography in Sensor Networks.
Security in Ad-hoc and Sensor Networks,
2004, 2-18p.
[17] **Kaps J.P.** Cryptography for Ultra-Low Power Devices. [žiūrėta 2011.06.11]
Prieiga per internetą: <http://www.crypto.wpi.edu/Publications/Documents/phd_kaps.pdf >
[18] **Kumar S., Girimondo M., Weimerskirch A., Paar C., Patel A., Wander A. S.** Embedded End-To-End Wireless Security With Ecdh Key Exchange. [žiūrėta 2011.06.11]
Prieiga per internetą:
<http://users.soe.ucsc.edu/~awander/stuff/Embedded_E2E_Security_ECC.pdf>
[19] **H. Rifa-Pous, J. Herrera-Joancomarti.** Computational and Energy Costs of Cryptographic Algorithms on Handheld Devices.
Future Internet,
2011, 3, 31-48p.
[20] **Idan Sheetrit, Avishai Wool.** Cryptanalysis of KeeLoq code-hopping using a Single FPGA.
Cryptology ePrint Archive,
2011
[21] RSA Laboratories. Rekomenduojamas šifravimo raktų ilgis, [žiūrėta 2012.04.28],
prieiga per Internetą: <http://www.rsa.com/rsalabs/node.asp?id=2264>

[22] **M. Kasper, T. Kasper, A. Moradi, C. Paar.** Breaking KeeLoq in a Flash. AFRICACRYPT 2009,
2009, 403-420.
[23] **T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, M. Shalmani.** On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme.
Advances in Cryptology, CRYPTO 2008,
2008, 203-220.
[24] Atmel. Manchester Coding Basics, [žiūrėta 2012.05.01],
prieiga per Internetą: <www.atmel.com/Images/doc9164.pdf>.
[25] Microchip Technology Inc. 44-pin demo board user's guide.
2007.
[26] PIC16F877 mikroprocesoriaus aprašymas, [žiūrėta 2012.03.22],
prieiga per Internetą:
<http://ww1.microchip.com/downloads/en/DeviceDoc/30292c.pdf>;

Creation and analysis of Security System for House Computer System

Summary

This paper presents gaps in the house computer systems security and suggests a way to solve it. It was noticed, that communication among house computer system's devices isn't encrypted or secured in any other way. This situation enable attacker to get data, sent among devices, and use it to affect the system. Another problem is attacks against wireless communication. If system's devices can't detect such attacks, it is vulnerable. This paper suggests using KeeLoq encryption to make communication safer. Research shows, that this type of encryption is effective in house computer system. There was created subsystem of wireless communication in this work. It enables house computer system to identify wireless attacks and respond to it properly. Research shows, that this subsystem correctly detects attacks and always help system to take right actions.

Terminų ir santrumpų žodynas

PC (Personal Computer) – Asmeninis kompiuteris.

CRC (Cyclic Redundancy Check) – Ciklinis perteklinis patikrinimas. Vienas iš kontrolinės sumos algoritmų.

RAM (Random Access Memory) – Laisvosios kreipties atmintinė. Darbinė atmintinė, naudojama saugoti duomenims, su kuriais tuo metu dirba įtaisas.

ROM (Read Only Memory) – Pastovioji atmintinė. Į šią atmintinę įtaiso programavimo metu yra įrašoma programa. Šioje atmintinėje įrašyti duomenys nekinta ir gali būti perrašyti tik kito programavimo metu.

Kb, Mb – Kilobaitas, Megabaitas. Informacijos matavimo vienetas.

KHz, GHz – Kilohercas, Megahercas. Matavimo vienetai, kuriais matuojamas procesoriaus daržnis.

GND – kontaktas / laidas, jungiamas prie maitinimo šaltinio neigiamo kontakto.

VDD – kontaktas / laidas, jungiamas prie maitinimo šaltinio teigiamo kontakto.

RX – kontaktas, skirtas duomenų priėmimui.

TX – kontaktas, skirtas duomenų išsiuntimui.

Priedai

1 priedas. Šifravimo posistemės programos kodas.

```
#define KeeLoq_NLF      0x3A5C
#define SNKey1          0xFA9F
#define SNKey2          0xD6C2
#define SNKey3          0xB8A4
#define SNKey4          0xE795

unsigned long     KeeLoq_Decrypt (unsigned long data, unsigned long key)
{
      unsigned long     x, tmp;
      int  x1, x2, x3, x4;
      unsigned long tmp1;
      int r, a, b, c, d, e;
      int nlf;

      x = data;
      for (r = 0; r < 19; r++)
      {
           a = (((x)>>(0))&1);
           b = (((x)>>(4))&1);
           c = (((x)>>(9))&1);
           d = (((x)>>(12))&1);
           e = (((x)>>(14))&1);
           nlf = ((((((((((0x00000 | e) << 1) | d) << 1) | c) << 1) | b) << 1) | a));
           x1 = (((x)>>(15))&1);
           x2 = (((x)>>(7))&1);
           x3 = (((key)>>(15))&1);
           x4 = (((KeeLoq_NLF)>>(nlf))&1);
           tmp = (x1^x2^x3^x4);
           x = (x << 1)| tmp;
           tmp1 = (((key)>>(15))&1);
           key = (key << 1) | tmp1;
      }
      x = x & 0x0000ffff;
      return x;
}
//-----------------------------------------------------------------------------------------------
unsigned long     Key_Hop (unsigned long key, long r)
{
      int i;
      unsigned long tmp1;

      for (i = 0; i < r; i++)
      {
           tmp1 = (((key)>>(0))&1);
           tmp1 = tmp1 << 15;
           key = (key >> 1) | tmp1;
      }
      return key;
}
```

```
//-----------------------------------------------------------------------------------------------
unsigned long KeeLoq_Encrypt (unsigned long data, unsigned long key, int key_sk, int dat_key)
{
	unsigned long x, tmp, tmp1;
	int  x1, x2, x3, x4, r, a, b, c, d, e, nlf;

		if(key_sk > 0)
			key = Key_Hop (key, key_sk);

		if (dat_key == 0){
			tmp = (0x000f & KeeLoq_NLF) << 12;
			x = data;
			x = x | tmp;
		} else {
			x = data;
		}

	for (r = 0; r < 19; r++)
	{
		a = (((x)>>(1))&1);
		b = (((x)>>(5))&1);
		c = (((x)>>(10))&1);
		d = (((x)>>(13))&1);
		e = (((x)>>(15))&1);

		nlf = ((((((((((0x00000 | e) << 1) | d) << 1) | c) << 1) | b) << 1) | a));

		x1 = (((x)>>(0))&1);
		x2 = (((x)>>(8))&1);
		x3 = (((key)>>(0))&1);
		x4 = (((KeeLoq_NLF)>>(nlf))&1);

		tmp = (x1^x2^x3^x4);
		tmp = (tmp << 15);
		x = (x >> 1)| tmp;
		tmp1 = (((key)>>(0))&1);
		tmp1 = tmp1 << 15;
		key = (key >> 1) | tmp1;
	}
	return x;
}
//-----------------------------------------------------------------------------------------------
unsigned long     Decrypt2 (unsigned long x, unsigned long key, int key_sk, int dat_key)
{
	unsigned long data, t, t1;
	int hop = 0;
	int sk = 0;

	hop = key_sk+3;
	if (dat_key == 0){
		t1 = 0;
		t = 0x000f & KeeLoq_NLF;
```

```
            while (t != t1 && sk < 17)
            {
                  key = Key_Hop (key, hop);
                  data = KeeLoq_Decrypt(x, key);
                  t1 = (0xf000 & data) >> 12;
                  t1 = t1 & 0x000f;
                  hop = 1;
                  key_sk++;
                  sk++;
            }
            if (key_sk < 15){
                  if (((0x0f00 & data) >> 8) != 0)
                        data = data & 0x0f00;
                  else
                        data = data & 0x00ff;
            }
            else if (sk < 17){      // pranešimas iššifruojamas su jau panaudota rakto kombinacija
                  if ((((0x0f00 & data) >> 8) == 10) || (((0x0f00 & data) >> 8) == 11))
                        data = data & 0x0f00;
                  else
                        data = 0x0C00;    // grąžinamas pranešimas, kad reikia naujo rakto
            } else                        // pranešimo nepavyko dešifruoti
                  data = 0x0E00;          // grąžinamas pranešimas, kad dešifruoti nepavyko
      } else {
            key = Key_Hop (key, hop);
            data = KeeLoq_Decrypt(x, key);
      }
      return data;
}
//-----------------------------------------------------------------------------------------------
int key_sk_plius (int sk)
{
      if (sk < 16)
            sk++;
      else
            sk = 0;
      return sk;
}
//-----------------------------------------------------------------------------------------------
unsigned long     Encrypt (unsigned long data, unsigned long key1, unsigned long key2,
unsigned long key3, unsigned long key4, int key_sk, int dat_key)
{
      unsigned long x;
      x = KeeLoq_Encrypt(data, key1, key_sk, dat_key);
      x = KeeLoq_Encrypt(x, key2, key_sk, 1);
      x = KeeLoq_Encrypt(x, key3, key_sk, 1);
      x = KeeLoq_Encrypt(x, key4, key_sk, 1);
      return x;
}

//-----------------------------------------------------------------------------------------------
```

```
unsigned long     Decrypt (unsigned long x, unsigned long key1, unsigned long key2, unsigned
long key3, unsigned long key4, int key_sk, int dat_key)
{
	unsigned long data;
	int kint = 0;
	int sk = 0;

	while (kint == 0 && sk < 17){
		data = Decrypt2(x, key4, key_sk, 1);
		data = Decrypt2(data, key3, key_sk, 1);
		data = Decrypt2(data, key2, key_sk, 1);
		data = Decrypt2(data, key1, key_sk, dat_key);
		if (((0x0f00 & data) >> 8) == 14)
			key_sk = key_sk_plius (key_sk);
		else
			kint = 1;
		sk++;
	}
	return data;
}
//-------------------------------------------------------------------------------------------
doubleCreateKey ()
{
	unsigned long key;
	int sk, sk2, i;

	sk = 0;
	key = 0x0000;
	for (i = 0; i < 4; i++)
	{
		sk = (rand() % 14);
		if (sk > 1 && sk != sk2)
		{
			key = (key << 4) | sk;
			sk2 = sk;
		}
		else
			i--;
	}
	return key;
}
```

2 priedas. Belaidžio ryšio posistemės programos kodas.

```
void Siust_M(int b0_1, unsigned long gr)
{
	if (b0_1 == 0){
		OUTPUT_HIGH(PIN_E0);
		delay_us(gr/2);
		OUTPUT_LOW(PIN_E0);
		delay_us(gr/2);
```

```
      } else {
           OUTPUT_LOW(PIN_E0);
           delay_us(gr/2);
           OUTPUT_HIGH(PIN_E0);
           delay_us(gr/2);
      }
}
//-----------------------------------------------------------------------------------------------
int Gaut_M(unsigned long gr)
{
      int b1, b2, b;
      b = 2;
      b1 = input_state(PIN_E1);
      delay_us(gr/2);
      b2 = input_state(PIN_E1);
      delay_us(gr/2);
      if (b1 == 0 && b2 == 1)
           b = 1;
      else if (b1 == 1 && b2 == 0)
           b = 0;
      return b;
}
//-----------------------------------------------------------------------------------------------
int CRC(unsigned long duom, int crc)
{
      int cc = 0b1011;
      unsigned long tmp = 0b00000000000;
      unsigned long tmp1 = 0b0000;
      int i = 11;
      tmp = 0x00ff & duom;
      tmp = tmp << 3;

      if (crc != 0){
           tmp = tmp | crc;
      }
      while (tmp >> 3 != 0){
           if (tmp >> (i-1) == 0){
                 i--;
           } else {
                 tmp1 = cc << (i-4);
                 tmp = tmp ^ tmp1;
                 i--;
           }
      }
      crc = tmp;
      return crc;
}
//-----------------------------------------------------------------------------------------------
void SiustiR(unsigned long duom, unsigned long greitis)
{
      unsigned long greit;
      int i, ii, bitas, cr;
```

```
	greit = 1000000/greitis;
	cr = CRC(duom, 0);

	// siuntimo pradžia
	for(i = 0; i < 10; i++)
		Siust_M(0, greit);
	delay_us(greit/2);

	// preambolė
	for(i = 0; i < 2; i++){
		OUTPUT_HIGH(PIN_D7);
		Siust_M(1, greit);
		OUTPUT_HIGH(PIN_D7);
		Siust_M(1, greit);
		OUTPUT_LOW(PIN_D7);
		Siust_M(0, greit);
		Siust_M(1, greit);
	}
	// duomenų siuntimas
	ii = 7;
	for(i = 0; i < 8; i++)
	{
		bitas = 0b00000001 & (duom >> ii);
		ii--;
		if(bitas == 1){
			Siust_M(1, greit);
			OUTPUT_HIGH(PIN_D7);
		}else{
			Siust_M(0, greit);
			OUTPUT_LOW(PIN_D7);
		}
	}
	// CRC
	OUTPUT_D(cr);
	ii = 2;
	for(i = 0; i < 3; i++)
	{
		bitas = (0b001 & (cr >> ii));
		ii--;
		if(bitas == 1){
			Siust_M(1, greit);
			OUTPUT_HIGH(PIN_D7);
		}else{
			Siust_M(0, greit);
			OUTPUT_LOW(PIN_D7);
		}
	}
	// baigiamas siuntimas
	OUTPUT_LOW(PIN_E0);
	OUTPUT_LOW(PIN_D7);
	delay_us(greit*2);
}
```

```
//-----------------------------------------------------------------------------------------
unsigned long GautiR(unsigned long greitis)
{
	unsigned long greit, duom, pre, negauta;
	int i, gauta, bitas, cr;

	i = 0;
	negauta = 0;
	gauta = 0;
	duom = 0x0000;
	pre = 0x0000;
	greit = 1000000/greitis;

	// Laukia siuntimo
	while (gauta == 0 && negauta < 60000)
	{
		negauta++;
		duom = 0x0000;
		i = 0;
		if (input_state(PIN_E1)==1){
		delay_us(greit/2);
		if (input_state(PIN_E1)==0){
		delay_us(greit/2);
		while (Gaut_M(greit) == 0)
		{
			i++;
		}
		if (i > 8 ){                          // gautas pradžios signalas
			// tikrina preambolę
			pre = 0x0000;
			i = 0;
			for(i = 0; i < 8; i++){
				pre = (pre << 1) | Gaut_M(greit);
			}
			if (pre == 0b11011101){       // preambolė teisinga
				// duomenų gavimas
				delay_us(200);
				greit = greit + 30;
				duom = 0x0000;
				for(i = 0; i < 8; i++)
				{
					bitas = Gaut_M(greit);
					if (bitas != 2)
						duom = (duom << 1) | bitas;
					else{
						duom = (duom << 1) | 0;
					}
				}

				// CRC
				cr = 0b000;
				for(i = 0; i < 3; i++)
```

```
                        {
                              bitas = Gaut_M(greit);
                              if (bitas != 2)
                                    cr = (cr << 1) | bitas;
                              else{
                                    cr = (cr << 1) | 0;
                              }
                        }
                        if (CRC(duom, cr) == 0){
                              gauta = 1;
                        }
                  }
            }
      }
      }
      greit = 1000;
      }
      if(negauta > 59999)
            duom = 0x0D00;
      return duom;
}
```

3 priedas. Eksperimentinės būsto kompiuterinės sistemos programos kodas. Valdantis įtaisas.

```
#include <16F887.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#use delay(clock=4000000)
#use rs232(baud=3200, parity=N, xmit=PIN_C6, rcv=PIN_C7, bits=8, stream=PC)

#use fast_io(A)
#use fast_io(B)

#FUSES INTRC,NOWDT,NOPUT,NOMCLR,NOPROTECT,NOCPD,NOBROWNOUT,NOIESO,NOFCMEN,NOLVP//,XT

struct adc_result
    {
    int8 value;
    int1 new_flag;
    } adc_conversion;

void init_io()    // Nustatomi įėjimai (1) ir išėjimai (0)
{
    SET_TRIS_A(0x01);
    SET_TRIS_B(0b00000011);
    SET_TRIS_E(0b11111110);
    OUTPUT_D(0x00);
}
```

```
void init_adc()
{
    SETUP_ADC_PORTS(sAN0 | VSS_VDD);
    SETUP_ADC(ADC_CLOCK_DIV_8);
    SET_ADC_CHANNEL(0);
    READ_ADC(ADC_START_ONLY);
}

#INT_RTCC
void timer0_interrupt_service()
{
    adc_conversion.value = READ_ADC(ADC_READ_ONLY);
    adc_conversion.new_flag = 1;
    READ_ADC(ADC_START_ONLY);
}
//=================================================================================================
void siusti (unsigned long x, int rysys)
{
      int i;
      if (rysys == 1){
            putc(x);
            delay_ms(10);
            putc(x>>8);
            delay_ms(10);
      } else {
            for(i = 0; i < 10; i++){
                  SiustiR(x, 1000);
                  delay_ms(10);
                  SiustiR(x >> 8, 1000);
            }
      }
}
//-------------------------------------------------------------------------------------------------
unsigned long gauti (int rysys)
{
      unsigned long in, in1, in2;
      in2 = 0;
      if (rysys == 1){
            in1 = getc(PC);
            delay_ms(10);
            in2 = getc(PC);
      } else {
            in1 = GautiR(1000);
            if ((in1 >> 8) != 13){
                  while ((in2 == in1) && ((in2 >> 8) != 13)){
                        in2 = GautiR(1000);
                  }
            }
      }
      if (((in1 >> 8) != 13) && (((in2 >> 8) != 13)))
            in = (in2<<8)|in1;
      else
```

```
            in = 0x0D00;
      return in;
}
//-------------------------- generuojamas ir siunčiamas raktas -------------------------------
int naujas_raktas (unsigned long& key1, unsigned long& key2, unsigned long& key3, unsigned
long& key4, int& key_sk, int rysys)
{
      unsigned long data, x;
      unsigned long key_n1, key_n2, key_n3, key_n4;
      int pavyko, sk;

      sk = 0;
      pavyko = 0;
      key_n1 = CreateKey();                        // generuojami raktai
      key_n2 = CreateKey();
      key_n3 = CreateKey();
      key_n4 = CreateKey();

      while(pavyko != 1 && sk < 3){
      x = Encrypt(0x0B00, key1, key2, key3, key4, key_sk, 0); // pranešimas apie siunčiamą raktą
      key_sk = key_sk_plius (key_sk);
      siusti(x, rysys);

      x = Encrypt(key_n1, SNKey1, SNKey2, SNKey3, SNKey4, 0, 1); // užšifruojmi ir siunčiami raktai
      siusti(x, rysys);
      x = Encrypt(key_n2, SNKey1, SNKey2, SNKey3, SNKey4, 0, 1);
      siusti(x, rysys);
      x = Encrypt(key_n3, SNKey1, SNKey2, SNKey3, SNKey4, 0, 1);
      siusti(x, rysys);
      x = Encrypt(key_n4, SNKey1, SNKey2, SNKey3, SNKey4, 0, 1);
      siusti(x, rysys);

      x = gauti(rysys);                            // laukiama pranešimo apie gautą raktą
      if (x != 3328)
            data = Decrypt(x, key1, key2, key3, key4, key_sk, 0);
      else
            data = x;
      sk++;

      if ((data >> 8) != 10 ){
            OUTPUT_D(0xF0);
      } else if ((data >> 8) != 13 ){
            OUTPUT_D(0xD0);
      } else {
            key1 = key_n1;
            key2 = key_n2;
            key3 = key_n3;
            key4 = key_n4;
            key_sk = 0;
            pavyko = 1;
            OUTPUT_D(0xA0);
      }
```

```
	}
	return pavyko;
}
//=============================================================================================
void main()
{
	int switch_count = 0;
	int rod = 0;
	unsigned long data, x;
	unsigned long key1, key2, key3, key4;
	int key_sk, dat_key;

	key_sk = 0;
	dat_key = 0;
	key1 =  0x5b32;
	key2 =  0x42c3;
	key3 =  0x2b9f;;
	key4 =  0xbe8b;

	init_io();
	init_adc();
	SETUP_TIMER_0(RTCC_INTERNAL | RTCC_DIV_256);
	ENABLE_INTERRUPTS(GLOBAL);
	ENABLE_INTERRUPTS(INT_RTCC);

//=============================================================================================
	OUTPUT_D(0x0f);
	while(1)
	{
	     if (INPUT_STATE(PIN_B0) == 0)             // Paspaudziamas mygtukas
	     {
	           if (switch_count < 8)               // Tikrinama ar tai tikras paspaudimas
	           {
	                 switch_count ++;              // Jei suskaičiuojami 8 paspaudimai,
	                                               // vadinasi tai tikras paspaudimas
	           if (switch_count == 8)
	           {
	                 if (rod == 0)                 // Jei tai pirmas paspaudimas
	                 {
	                       rod = 1;

//------------------- generuojamas ir siunčiamas naujas raktas -----------------------------
	if (key_sk >= 16 || key1 == 0x5b32){
	     naujas_raktas(key1, key2, key3, key4, key_sk, 1);     // generuojamas raktas
	}

//------------------ siunčiamos komandos ir laukiama atsakymų -----------------------------
	OUTPUT_D(0x20);                                          // Siunčiama komanda
	x = Encrypt(0x0200, key1, key2, key3, key4, key_sk, 0);
	key_sk = key_sk_plius (key_sk);
	siusti(x, 1);
```

```
      OUTPUT_D(0x00);
      x = gauti(1);
      data = Decrypt(x, key1, key2, key3, key4, key_sk, 0);
      OUTPUT_D(data>>8);
      key_sk = key_sk_plius (key_sk);
      switch( data >> 8 ) {
           case 12:                          // pranešimas apie pasibaigusį raktą
           {
                 naujas_raktas(key1, key2, key3, key4, key_sk, 1);     // generuojamas raktas
           }
           break;
           default :
           {
                 OUTPUT_D(data);
           }
      }
      } else {                          // Jei tai antras paspaudimas.
           OUTPUT_D(0x0f);
           rod = 0;
      }
      }
      }
      } else {          // Jei mygtukas nėra paspaudžiamas, skaitliukas nustatomas į 0.
           switch_count = 0;
      }
      }
}
```

Šalutinis įtaisas.

```
#include <16F887.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#use delay(clock=4000000)
#use rs232(baud=3200, parity=N, xmit=PIN_C6, rcv=PIN_C7, bits=8, stream=PC)

#use fast_io(A)
#use fast_io(B)

#FUSES INTRC,NOWDT,NOPUT,NOMCLR,NOPROTECT,NOCPD,NOBROWNOUT,NOIESO,NOFCMEN,NOLVP//,XT

struct adc_result
    {
    int8 value;
    int1 new_flag;
    } adc_conversion;

void init_io()    // Nustatomi įėjimai (1) ir išėjimai (0)
{
    SET_TRIS_A(0x01);
```

```
    SET_TRIS_B(0b00000011);
    SET_TRIS_E(0b11111110);
    OUTPUT_D(0x00);
}
void init_adc()
{
    SETUP_ADC_PORTS(sAN0 | VSS_VDD);
    SETUP_ADC(ADC_CLOCK_DIV_8);
    SET_ADC_CHANNEL(0);
    READ_ADC(ADC_START_ONLY);
}

#INT_RTCC
void timer0_interrupt_service()
{
    adc_conversion.value = READ_ADC(ADC_READ_ONLY);
    adc_conversion.new_flag = 1;
    READ_ADC(ADC_START_ONLY);
}
//=============================================================================================
void siusti (unsigned long x, int rysys)
{
      int i;
      if (rysys == 1){
            putc(x);
            delay_ms(10);
            putc(x>>8);
            delay_ms(10);
      } else {
            for(i = 0; i < 10; i++){
                  SiustiR(x, 1000);
                  delay_ms(10);
                  SiustiR(x >> 8, 1000);
            }
      }
}
//---------------------------------------------------------------------------------------------
unsigned long gauti (int rysys)
{
      unsigned long in, in1, in2;
      in2 = 0;
      if (rysys == 1){
            in1 = getc(PC);
            delay_ms(10);
            in2 = getc(PC);
      } else {
            in1 = GautiR(1000);
            if ((in1 >> 8) != 13){
                  while ((in2 == in1) && ((in2 >> 8) != 13)){
                        in2 = GautiR(1000);
                  }
            }
```

```
	}
	if (((in1 >> 8) != 13) && (((in2 >> 8) != 13)))
		in = (in2<<8)|in1;
	else
		in = 0x0D00;
	return in;
}
//-------------------------------------------------------------------------------------------
int gaunamas_raktas (unsigned long& key1, unsigned long& key2, unsigned long& key3, unsigned
long& key4, int& key_sk, int rysys)
{
	unsigned long key_n1, key_n2, key_n3, key_n4, x;
	int pavyko;

	pavyko = 0;
	key_n1 = gauti(rysys);
	key_n2 = gauti(rysys);
	key_n3 = gauti(rysys);
	key_n4 = gauti(rysys);

	if (key_n1 != 0 && key_n2 != 0 && key_n3 != 0 && key_n4 != 0 && key_n1 != 3328 && key_n2
!= 3328 && key_n3 != 3328 && key_n4 != 3328){
		x = Encrypt(0x0A00, key1, key2, key3, key4, key_sk, 0);// pranešimas kad raktas gautas
		siusti(x, rysys);
		OUTPUT_D(0xA0);
		key1 = Decrypt(key_n1, SNKey1, SNKey2, SNKey3, SNKey4, 0, 1);
		key2 = Decrypt(key_n2, SNKey1, SNKey2, SNKey3, SNKey4, 0, 1);
		key3 = Decrypt(key_n3, SNKey1, SNKey2, SNKey3, SNKey4, 0, 1);
		key4 = Decrypt(key_n4, SNKey1, SNKey2, SNKey3, SNKey4, 0, 1);
		pavyko = 1;
		key_sk = 0;
	} else {
		OUTPUT_D(0xF0);
		x = Encrypt(0x0F00, key1, key2, key3, key4, key_sk, 0);
		key_sk = key_sk_plius (key_sk);
		siusti(x, rysys);
	}
	return pavyko;
}
//===========================================================================================
void main()
{
	int bars = 0;
	int temp1 = 0;
	int led_display = 0;
	unsigned long data, x, key1, key2, key3, key4;
	int key_sk, dat_key;
	key_sk = 0;
	dat_key = 0;
	x = 0;
	key1 =  0x5b32;
	key2 =  0x42c3;
```

```
key3 =  0x2b9f;
key4 =  0xbe8b;

adc_conversion.value = 0;
adc_conversion.new_flag = 0;

init_io();
init_adc();
SETUP_TIMER_0(RTCC_INTERNAL | RTCC_DIV_256);
ENABLE_INTERRUPTS(GLOBAL);
ENABLE_INTERRUPTS(INT_RTCC);

while(1){
OUTPUT_D(0x00);
while (x == 0 || x == 3328){
      x = gauti(1);
}
data = Decrypt(x, key1, key2, key3, key4, key_sk, 0);
key_sk = key_sk_plius (key_sk);
switch( data >> 8 ) {
      case 12:                          // pranešimas apie pasibaigusį raktą
      {
           OUTPUT_D(0xc0);
           x = Encrypt(0x0C00, key1, key2, key3, key4, key_sk, 0);
           key_sk = key_sk_plius (key_sk);
           siusti(x, 1);
      }
      break;
      case 11:                          // gautas pranešimas apie siunčiamą raktą
      {
      gaunamas_raktas (key1, key2, key3, key4, key_sk, 1);
      }
      break;
      case 2:                           // prašymas atsiųsti matavimų rezultatus
      {
           if (adc_conversion.new_flag == 1){
                 bars = adc_conversion.value;
                 adc_conversion.new_flag = 0;
                 bars = (bars >> 5) + 1;
                 led_display = 0;
                 for (temp1 = 0; temp1 < bars; temp1++)
                 {
                       led_display = (led_display << 1) + 1;
                 }
           }
           data = led_display;
           OUTPUT_D(data);
           x = Encrypt(data, key1, key2, key3, key4, key_sk, 0);
           key_sk = key_sk_plius (key_sk);
           delay_ms(1000);
           siusti(x, 1);
      }
```

```
            break;
            default :
            {
            OUTPUT_D(0x0900);
                  x = Encrypt(0x0900, key1, key2, key3, key4, key_sk, 0);
                  key_sk = key_sk_plius (key_sk);
                  siusti(x, 1);
            }
            }
      }
}
```