KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS IR INFORMACINIŲ TECHNOLOGIJŲ SAUGOS STUDIJŲ PROGRAMA

GYTIS GRIGARAVIČIUS

KRIPTOGRAFINIŲ ALGORITMŲ ĮTAKA ENERGIJOS SUVARTOJIMUI MOBILIUOSE ĮRENGINIUOSE

Magistro baigiamasis darbas

Darbo vadovas

doc. dr. J. Toldinas

KAUNAS, 2013

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS IR INFORMACINIŲ TECHNOLOGIJŲ SAUGOS STUDIJŲ PROGRAMA

GYTIS GRIGARAVIČIUS

KRIPTOGRAFINIŲ ALGORITMŲ ĮTAKA ENERGIJOS SUVARTOJIMUI MOBILIUOSE ĮRENGINIUOSE

Magistro baigiamasis darbas

Recenzentas
prof. dr. Eligijus Sakalauskas

Vadovas
doc. dr. J. Toldinas

Atliko
IFNE-1/3 gr. stud.
Gytis Grigaravičius

KAUNAS, 2013

AUTORIŲ GARANTINIS RAŠTAS

DĖL PATEIKIAMO KŪRINIO

2013 - ..... - ..... d.

Kaunas

Autorius,_____

(vardas, pavardė)

_____,

patvirtina, kad Kauno technologijos universitetui pateiktas baigiamasis bakalauro (magistro) darbas (toliau vadinama – Kūrinys)_____

(kūrinio pavadinimas)

_____

pagal Lietuvos Respublikos autorių ir gretutinių teisių įstatymą yra originalus ir užtikrina, kad

1) jį sukūrė ir parašė Kūrinyje įvardyti autoriai;
2) Kūrinys nėra ir nebus įteiktas kitoms institucijoms (universitetams) (tiek lietuvių, tiek užsienio kalba);
3) Kūrinyje nėra teiginių, neatitinkančių tikrovės, ar medžiagos, kuri galėtų pažeisti kito fizinio ar juridinio asmens intelektinės nuosavybės teises, leidėjų bei finansuotojų reikalavimus ir sąlygas;
4) visi Kūrinyje naudojami šaltiniai yra cituojami (su nuoroda į pirminį šaltinį ir autorių);
5) neprieštarauja dėl Kūrinio platinimo visomis oficialiomis sklaidos priemonėmis.
6) atlygins Kauno technologijos universitetui ir tretiesiems asmenims žalą ir nuostolius, atsiradusius dėl pažeidimų, susijusių su aukščiau išvardintų Autorių garantijų nesilaikymu;
7) Autoriai už šiame rašte pateiktos informacijos teisingumą atsako Lietuvos Respublikos įstatymų nustatyta tvarka.

Autoriai

_____ _____

(vardas, pavardė) (parašas)

SANTRAUKA

Kriptografijoje egzistuoja labai didelis algoritmų kiekis. Vieni iš jų, kriptoanalizės atžvilgiu, yra labai stiprūs, tačiau itin imlūs skaičiavimams, kiti silpnesni, bet itin greiti ir reikalauja nedaug bei primityvių aritmetinių operacijų. Rinkoje naudojami nusistovėję, bei standartizuoti kriptografiniai algoritmai. Realiame gyvenime visada iškyla dilema kokį kriptografinį algoritmą pasirinkti vienokiam ar kitokiam uždaviniui spręsti. Pavyzdžiui, jeigu duomenys yra aktualūs tik keletą minučių ar sekundžių, tai bus visai nesvarbu, kad naudojamą kriptografinį algoritmą, pasitelkus tam tikrą skaičiavimo pajėgumą bus galima „nulaužti“ per keletą valandų. Egzistuoja ir tokios situacijos kai saugumas turi būti užtikrinamas labai ilgais laiko periodais, kaip pavyzdį čia būtų galima pateikti elektroninį parašą. Pastaruoju atveju yra itin svarbu, kad pasirašytas svarbus dokumentas, nebūtų pakeistas, tarkime 20 metų.

Duomenų saugos problema yra išspręsta, tačiau ši sauga lemia trumpesnį įrenginių darbą naudojant baterijoje sukauptą energiją. Kaip jau minėjome kriptografiniai algoritmai skiriasi savo atsparumu kriptoanalizei (saugumu) ir imlumu skaičiavimams. Pastarasis rodiklis tiesiogiai susijęs su energijos sąnaudomis reikalingomis duomenis šifruoti. Čia ir iškyla dilema – kokį kriptografinį algoritmą pasirinkti norint užtikrinti pageidaujamą saugos stiprumo lygį, bet šifravimui nesunaudoti visų turimos energijos atsargų. Šioje vietoje būtų galima pabrėžti ir tai, kad energijos taupymas naudingas ne tik mobiliems įrenginiams, bet ir stacionariems kompiuteriams, ypač duomenų centruose esantiems serveriams. Juk sumažinus serverio apkrovą tuo pačiu sumažėja ir energijos sąnaudos, kas įtakoja mažesnes sąskaitas už elektrą. Tarkime sumažinus naudojamą galią 10W per metus butų sutaupoma ~87,6kW elektros sąnaudų, o šio darbo rašymo metu, tai sudarytų beveik 40 litų. Jeigu tai padaugintume iš 100 serverių gautume gana solidžią 4000 litų sumą. Ir tai tik 10W mažesnėmis elektros energijos sąnaudomis...

Šio darbo pagrindiniai tikslai yra:

- Sukurti baterijos elgsenos modelį;
- Sukurti programinę įrangą reikalingą energijos sąnaudų matavimams atlikti;
- Atlikti eksperimentą išmatuojant įvairių kriptografinių algoritmų energijos sąnaudas;
- Patikrinti modelio atitikimą eksperimento rezultatams;
- Ištirti simetrinių ir asimetrinių šifrų bei maišos funkcijų įtaką energijos sąnaudoms;
- Įvertinti baterijos susidėvėjimą įtakojamą įkrovimo-iškrovimų ciklų.

SUMMARY

Under the term 'cryptography' there's a vast amount of various cipher algorithms. Some of these ciphers are very strong in terms of cryptanalysis, but very demanding in computing power. Others are weaker, but blazingly fast and require only few, primitive, arithmetic operations. In computing and security market some of the ciphers are standardized. In the real world there's always a dilemma which cipher to choose for one or another application. For e.g. if data which is about to be protected is relevant only for a few minutes or even seconds it would be completely irrelevant if cipher used for to protect that date could be broken in few days. On another hand there are applications when data protection will be relevant even for upcoming twenty years. Let's take a digital signature as an example – it is very important to be completely sure that digitally signed important document would be secure after couple of decades after it was signed.

Currently data protection problem is solved, but as a trade-off there's shorter life-time of mobile devices on which secure data is used or processed. As we've already mentioned before different cryptographic algorithms differ in their resistance to cryptanalysis and receptivity to calculations. Latter indicator is directly tied with energy consumptions required to protect data. Right here dilemma and arises – which cipher to use to give enough protection to precious data without depleting all the remaining battery charge. At this place it is relevant to emphasize that energy consumption is not only relevant to mobile devices, but for desktop computer or event servers stored in a datacenter. Decrease server's load automatically decreases its power consumption which in turn decreases electricity bills. Assume that one was able to decrease server's power consumption by 10W. Throughout a year it becomes 88kW of saved electrical power. When this paper was written savings would be about 40 Litas/annum. Let's multiple these numbers by 100 (servers) and it becomes savings of 4000 Litas/annum. All this by only reducing power consumption by 10 watts…

Aims of this paper:

- Created battery behavior model;
- Design and develop software for energy consumption measurements of different cryptographic algorithms;
- Perform and experiment and measure energy consumption of selected cryptographic algorithms;
- Validate model against actual data;
- Research how symmetrical and asymmetrical ciphers and hash functions influences battery state-of-charge;
- Evaluate battery wear during multiple charge-discharge cycles.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

TERMINŲ IR SANTRUMPŲ ŽODYNAS

API	Application Programming Interface. Aplikacijų kūrimo sąsaja
Bruteforce	Grubios jėgos ataka. Atakos tipas kurią vykdant perrenkami visi galimi (pvz. slaptažodžių variantai)
CBC	Cipher Block Chaining mode. Blokinių šifrų šifro blokų jungimo režimas
CFB	Cipher-FeedBack mode. Blokinių šifrų šifro grįžtamojo ryšio režimas
CLR	Common Language Runtime. Bendroji vykdančioji aplinka
CNG	Cryptography Next Generation. Microsoft sukurta ir palaikoma kriptografinė biblioteka
CSV	Comma-Separeted Values. Kableliu (nebūtina) atskirtos reikšmės
DWEE	Dynamic Weighted Energy-Efficiency benchmark. Dinaminis svoriai pagrįstas energetiškai efektyvus etaloninis testas
ECB	Electronic CodeBook mode. Blokinių šifrų elektroninės kodų knygos režimas
FIPS	Federal Information Processing Standard. Federalinis informacijos apdorojimo standartas, JAV
Hash	Santrauka
Hash function	Maišos (santraukų skaičiavimo) funkcija
IDE	Integrated Development Environment. Integruota programavimo aplinka
ISM	Industrial, Scientific and Medical. Nelicencijuojama radijo dažnių juosta skirta pramoniniams, moksliniam ir medicininiam tikslam (433MHz, 868MHz)
Kriptograma	Užšifruota informacija (tekstograma)
Lookup-table	Greitos paieškos lentelė. Indeksuota lentelė su iš anksto paruoštais (paskaičiuotais) duomenimis greitai paieškai
mod(ulo)	"modulo" operacija, kuri gražina liekaną po sveikų skaičių dalybos
NIST	National Institute of Standards and Technology. Nacionalinis standartų ir technologijų institutas, JAV
OFB	Output-FeedBack mode. Blokinių šifrų rezultato grįžtamojo ryšio režimas
Padding	Papildymas iki buferio (masyvo) galo. Technologija naudojama pilnam buferio užpildymui užtikrinti
PKI	Public Key Infrastructure. Viešojo rakto infrastruktūra
RAM	Random Access Memory. Laisvosios kreipties atmintis

ROM	Read-Only Memory. Tik skaitymui skirta atmintis
Raundas	Kriptografijoje nurodo vieną bloko šifravimo ciklą
RISC	Reduced Instruction Set Computer. Sumažinto instrukcijų kiekio kompiuteris. Procesorių architektūros tipas.
SoC	State-of-Charge. Įkrovos būsena
Šifrograma	Užšifruota informacija (tekstograma)
Tekstograma	Nešifruota informacija

ĮVADAS

Įvairūs mobilūs įrenginiai šiuo metu yra itin išpopuliarėję. Jų parametrai vis gerėja ir po truputį nyksta našumo riba tarp stalinių kompiuterių ir įvairių nešiojamųjų įrenginių įskaitant išmaniuosius telefonus, planšetinius ir nešiojamuosius kompiuterius. Yra vienas rodiklis kuris nesivysto taip sparčiai, kaip skaičiavimo pajėgumai – tai baterijų talpa. Čia ir iškyla pagrindinė problema – energijos taupymas siekiant prailginti įrenginio veikimo laiką.

Nešiojamieji įrenginiai yra padidintos rizikos zonoje dėl duomenų praradimo, dėl įrenginio netekimo, tame tarpe ir vagystės, atžvilgiu. Dirbant mobiliuoju įrenginiu duomenys, kaip taisyklė, yra perdavinėjami belaidžio ryšio tinklų (WiFi, Bluetooth, 3G, WiMax, LTE ir pan.) pagalba. Toks duomenų perdavimas yra itin nesaugus, nes bet kas esantis ryšio aprėpties zonoje gali perimti siunčiamus ir gaunamus duomenis ir juos panaudoti piktais kėslais. Šioms dviem problemoms spręsti yra sukurta daug įrankių ir priemonių pagrįstų kriptografiniais sprendimais, kurie užtikrina duomenų, esančių įrenginyje arba perduodamų belaidžiu ryšiu, saugą.

Kriptografijoje egzistuoja labai didelis algoritmų kiekis. Vieni iš jų, kriptoanalizės atžvilgiu, yra labai stiprūs, tačiau itin imlūs skaičiavimams, kiti silpnesni, bet itin greiti ir reikalauja nedaug bei primityvių aritmetinių operacijų. Rinkoje naudojami nusistovėję, bei standartizuoti kriptografiniai algoritmai. Realiame gyvenime visada iškyla dilema kokį kriptografinį algoritmą pasirinkti vienokiam ar kitokiam uždaviniui spręsti. Pavyzdžiui, jeigu duomenys yra aktualūs tik keletą minučių ar sekundžių, tai bus visai nesvarbu, kad naudojamą kriptografinį algoritmą, pasitelkus tam tikrą skaičiavimo pajėgumą bus galima „nulaužti“ per keletą valandų. Egzistuoja ir tokios situacijos kai saugumas turi būti užtikrinamas labai ilgais laiko periodais, kaip pavyzdį čia būtų galima pateikti elektroninį parašą. Pastaruoju atveju yra itin svarbu, kad pasirašytas svarbus dokumentas, nebūtų pakeistas, tarkime 20 metų.

Duomenų saugos problema yra išspręsta, tačiau ši sauga lemia trumpesnį įrenginių darbą naudojant baterijoje sukauptą energiją. Kaip jau minėjome kriptografiniai algoritmai skiriasi savo atsparumu kriptoanalizei (saugumu) ir imlumu skaičiavimams. Pastarasis rodiklis tiesiogiai susijęs su energijos sąnaudomis reikalingomis duomenis šifruoti. Čia ir iškyla dilema – kokį kriptografinį algoritmą pasirinkti norint užtikrinti pageidaujamą saugos stiprumo lygį, bet šifravimui nesunaudoti visų turimos energijos atsargų. Šioje vietoje būtų galima pabrėžti ir tai, kad energijos taupymas naudingas ne tik mobiliems įrenginiams, bet ir stacionariems kompiuteriams, ypač duomenų centruose esantiems serveriams. Juk sumažinus serverio apkrovą tuo pačiu sumažėja ir energijos sąnaudos, kas įtakoja mažesnes sąskaitas už elektrą. Tarkime sumažinus naudojamą galią 10W per metus butų sutaupoma ~87,6kW elektros sąnaudų, o šio darbo rašymo metu, tai sudarytų beveik 40 litų. Jeigu tai padaugintume iš 100 serverių gautume gana solidžią 4000 litų sumą. Ir tai tik 10W mažesnėmis elektros energijos sąnaudomis...

Šio darbo pagrindiniai tikslai yra:

- Sukurti baterijos elgsenos modelį;
- Sukurti programinę įrangą reikalingą energijos sąnaudų matavimams atlikti;
- Atlikti eksperimentą išmatuojant įvairių kriptografinių algoritmų energijos sąnaudas;
- Patikrinti modelio atitikimą eksperimento rezultatams;
- Ištirti simetrinių ir asimetrinių šifrų bei maišos funkcijų įtaką energijos sąnaudoms;
- Įvertinti baterijos susidėvėjimą įtakojamą įkrovimo-iškrovimų ciklų.

Magistrinio darbo tema buvo parašytas mokslinis straipsnis [1], kuris, darbo rašymo metu, yra priimtas spausdinimui žurnale „ELEKTRONIKA IR ELEKTROTECHNIKA“, ISSN 1392-1215, 2013.

1. KRIPTOGRAFINIŲ ALGORITMŲ IR JŲ ENERGIJOS SĄNAUDŲ ANALIZĖ

Kriptografijoje šifru vadinamas algoritmas naudojamas užšifruoti arba iššifruoti pranešimus. Užšifravus pranešimą (tekstogramą) gaunama kriptograma. Šifravimo algoritmai paprastai naudoja papildomą informaciją, kuri vadinama raktu. Raktai leidžia šifravimo algoritmų nelaikyti paslaptyje, saugus turi išlikti tik raktas. Naudojant skirtingus raktus šifruojant tą pačią tekstogramą gaunamos skirtingos šifrogramos.

Šifrai skirstomi į tris dideles grupes (žr. pav. 1):

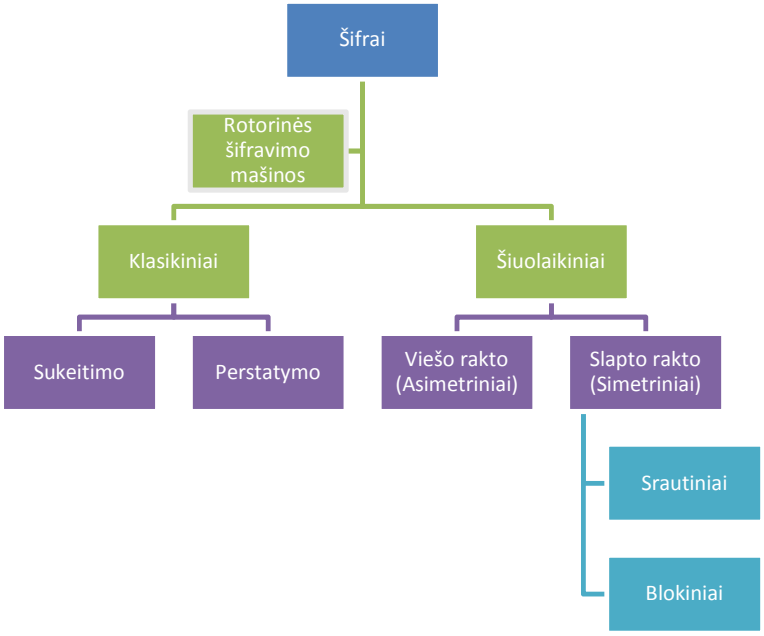
- Rotorinės šifravimo mašinos;
- Klasikiniai šifrai;
- Šiuolaikiniai šifrai.

Dvidešimtojo amžiaus pirmoje pusėje buvo išrastos elektro-mechaninės šifravimo mašinos, vadinamos rotorinėmis šifravimo mašinomis. Geriausiai žinoma rotorinė šifravimo mašina buvo Trečiojo Reicho sukurtas *Enigma* aparatas, kuris buvo naudotas Antrojo Pasaulinio karo metais. Pagrindinis šių mašinų skiriamasis bruožas buvo dantračių (rotorių) rinkinys užmautas ant bendros ašies. Kiekvieno rotoriaus abiejose pusėse buvo prijungti elektriniai kontaktai. Kontaktų tarpusavio sujungimai veikė kaip sukeitimų tinklas. Šio tinklo dinamiškumas buvo užtikrinamas rotorių pasukimu sulig kiekviena užšifruota raide. Šiais laikais rotorines šifravimo mašinas galima sutikti nebent muziejuose, tačiau jos davė pradžią šifravimo automatizavimui.

Skiriamos dvi grupės klasikinių šifrų: sukeitimo (*angl. substitution*) šifrai ir perstatymo (*angl. transposition*) šifrai. Sukeitimo šifrai yra bene paprasčiausi šifrai. Šifruojant kiekviena raidė (gali būti dvi, trys arba skirtingos kombinacijos) pakeičiama atitinkamu kiekiu kitų raidžių. Šifravimui rankiniu būdu yra naudojamos šifrų knygos, kuriose būna surašytos sukeitimų lentelės. Iššifravimas vykdomas atvirkštine tvarka atliekant sukeitimus pagal žinomas lenteles. Perstatymo šifrai tekstogramos simbolius ar jų grupes sukeičia vietomis su kitais simboliais ar jų grupėmis. Šifrogramos užšifruotos perstatymo metodu iššifruojamos atliekant perstatymus atvirkštine tvarka.

Šiuolaikiniai šifrai skirstomi į asimetrinius (viešojo rakto) ir simetrinius (slaptojo rakto). Asimetriniai algoritmai šifravimui ir iššifravimui naudoja du skirtingus raktus kurie yra matematiškai susieti. Vienas iš šių raktų gali būti viešai paskelbiamas (nors ir spaudoje). Tekstogramą užšifravus viešuoju raktu iššifruoti bus įmanoma tik naudojant privatųjį raktą. Būtent ši savybė asimetrinius šifrus ir išpopuliarino, tačiau jie turi ir trūkumų – šie šifrai yra keliomis eilėmis lėtesni už simetrinius ir nėra labai praktiški dideliems duomenų kiekiams šifruoti.

Simetriniai šifrai užšifravimui ir iššifravimui naudoja tą patį raktą, kuris turi būti laikomas paslaptyje. Simetriniai šifrai yra de-facto naudojami didelių duomenų masyvų ar srautų šifravimui. Savo ruožtu yra skiriamos dvi simetrinių šifrų grupės: srautiniai ir blokiniai. Srautiniai šifrai informaciją šifruoja po vieną bitą (arba baitą) ir neturi jokių apribojimų duomenų ilgiui. Tai leidžia šiuos šifrus panaudoti realaus laiko duomenų srautų šifravimui įvedant minimalius vėlinimus. Blokiniai šifrai operuoja fiksuoto ilgio duomenų blokais – jiems veikti reikia turėti duomenų bloką kuris ir bus šifruojamas ar iššifruojamas. Naudojant blokinius šifrus realaus laiko duomenims šifruoti sukuriamas vėlinimas, kurio metu duomenys kaupiami buferyje, šifruojami ir išsiunčiami tolesniam apdorojimui. Tačiau blokiniai šifrai labai gerai tinka didelės apimties failams šifruoti.



1 pav. Šifravimo algoritmų rūšys

1.1. Kriptografinių algoritmų analizė

Šiame darbe aprašytam tyrimui buvo pasirinkta po keturis asimetrinius ir simetrinius šifravimo algoritmus bei keturios maišos funkcijos (žr. 1 lentelę). Algoritmų tyrimui pasirinkimo kriterijai buvo šie:

- Platus naudojimas (sertifikatai, e. parašo sistemos, standartai, protokolai):
  - RSA, DSA, AES, MD5, SHA1, RC4
- Laisvai prieinamos bibliotekos (.NET, atviro kodo bibliotekos):
  - Visi pasirinkti šifrai
- Naudojami sudėtingesnių kriptoalgoritmų viduje tarpinėms operacijoms atlikti:
  - SHA1 ir SHA2 naudojamas DSA algoritme
- Artimi konkurentai:
  - Rijndael (AES) ir Serpent standartizavimo metu užėmė atitinkamai pirmą ir antrą vietas
- Senesnės ir naujesnės algoritmo modifikacijos:
  - SHA1 ir SHA2
- Standartizuoti:
  - Camelia, ECDSA, ElGamal

1 lentelė. Tiriamų kriptografinių algoritmų sąrašas

Tipas	Pavadinimas	Pilnas pavadinimas	Tipas
Simetrinio rakto algoritmai	AES	Advanced Encryption Standard	Simetrinio rakto blokinis šifras
	RC4	Ron Cipher 4, ARCFOUR	Simetrinio rakto srautinis šifras
	Camellia	Camellia	Simetrinio rakto blokinis šifras
	Serpent	Serpent	Simetrinio rakto blokinis šifras
Asimetrinio rakto algoritmai	RSA	Ron Rivest, Adi Shamir and Leonard Adleman	Asimetrinio rakto šifras ir skaitmeninio parašo algoritmas
	DSA	Digital Signature Algorithm	Skaitmeninio parašo algoritmas
	ECDSA	Elliptic Curve Digital Signature Algorithm	Eliptinių kreivių skaitmeninio parašo algoritmas
	ElGamal	ElGamal encryption system	Asimetrinio rakto šifras
Maišos funkcijos	MD5	Message Digest Algorithm	Kriptografinė maišos funkcija
	SHA1	Secure Hash Algorithm 1	Kriptografinė maišos funkcija
	SHA2/512	Secure Hash Algorithm 2/512	Kriptografinė maišos funkcija
	Whirlpool	Whirlpool	Kriptografinė maišos funkcija

1.1.1. Simetriniai šifrai

Simetriniai šifravimo algoritmai, dar vadinami slapto rakto šifrais, duomenų šifravimui ir iššifravimui naudoja vieną ir tą patį raktą. Šis raktas turi būti laikomas paslaptyje, iš čia ir kilo kriptografinių algoritmų šeimos pavadinimas – slapto rakto šifrai. Nepaisant šio didelio algoritmų trūkumo simetriniai algoritmai yra plačiausiai naudojami didelių duomenų kiekių šifravimui. Taip susiklostė todėl, jog simetriniai algoritmai yra daug kartų našesni už asimetrinius. Simetriniai algoritmai skirstomi į dvi pagrindines grupes: blokinius ir srautinius (žr. 1). Blokiniai ir srautiniai simetriniai šifrai savo veikimo principais yra visiškai skirtingi, tačiau naudojant tam tikrus algoritmus kiekvieną blokinį šifrą galima paversti srautiniu ir atvirkščiai. Pagrindinis skirtumas tarp šių dviejų šifrų šeimų yra aprašytas [2]:

> *„Blokiniai šifrai veikia naudodami fiksuotas transformacijas dideliems pradinių duomenų blokams, o srautiniai šifrai operuoja atskirais pradinių duomenų bitais naudodami laike kintamas transformacijas“*

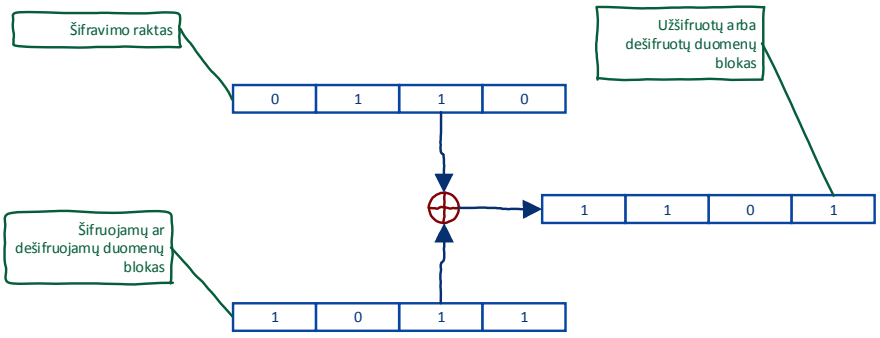
Blokiniai simetriniai šifrai operuoja fiksuoto ilgio duomenų blokais (pvz. 128, 192, 256 ir pan. bitų). Iš esmės jie nėra pritaikyti apdoroti bet kokio ilgio duomenų blokus (žr. pav. 2). Blokinių šifrų atveju ši problema sprendžiama pasitelkiant lygiavimą (*angl. padding*). Visas šifravimui skirtas duomenų masyvas padalijamas į dalis lygias šifravimo algoritmo bloko ilgiui, o paskutinę (nebūtinai) dalį papildant bet kokiais duomenimis. Šie papildomi duomenys iššifravus kriptogramą atmetami.

Blokiniai šifrai, naudojant tą patį raktą, identiškus pradinius duomenis užšifruoja visada į tą pačią kriptogramą. Tai ypač palengvina kriptoanalitikų darbą, nes tampa pakankamai nesudėtinga atsekti pasikartojančias sekas ir bent dalinai atkurti šifravimui naudojamą slaptą raktą. Siekiant pagerinti simetrinių šifravimo algoritmų saugą buvo sukurta eilė skirtingų jų darbo metodų (režimų). Šie metodai pritaikomi bet kokiems blokiniams šifravimo algoritmams,

nes jie aprašo kaip turi būti elgiamas su pradiniais ir užšifruotais ar iššifruotais blokais. Pagrinde, tai yra manipuliacijos su minėtais blokais kurių pasėkoje blokai tampa priklausomi vieni nuo kitų. Taip pat įvedama inicializacijos vektoriaus (IV) sąvoka, kuri padeda apsaugoti patį pirmąjį šifruojamų duomenų bloką, nes jis susiejamas su inicializacijos vektoriumi taip kaip sekantys blokai susiejami su prieš tai buvusiais. Dažniausiai naudojami metodai yra:

- ECB – elektroninės kodų knygos režimas (*angl. Electronic CodeBook mode*)
- CBC – šifro blokų jungimo režimas (*angl. Cipher Block Chaining mode*)
- CFB – šifro grįžtamojo ryšio režimas (*angl. Cipher-FeedBack mode*)
- OFB – rezultato grįžtamojo ryšio režimas (*angl. Output-FeedBack mode*)

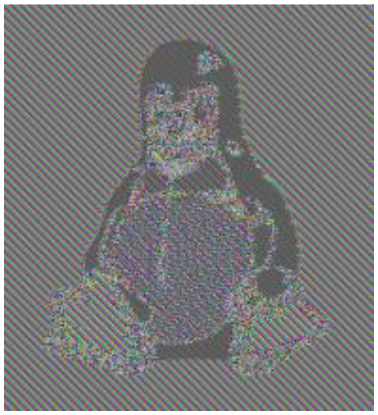
Praktikoje yra sutinkama ir daugiau įvairių blokinių šifrų darbo režimų, tačiau dauguma jų yra išvestiniai iš aukščiau paminėtų keturių režimų. Kaip keletą pavyzdžių paminėsime skaitliuko (*angl. Counter mode*), blokų jungimo (*angl. Block Chaining mode*), dauginamasis blokų jungimo (*angl. Propagating Cipher Block Chaining mode*), kripto blokų jungimo su kontroline suma (*angl. Cipher Block Chaining with Checksum mode*), netiesinės funkcijos grįžtamojo ryšio (*angl. Output FeedBack with a NonLinear Function*) režimai.



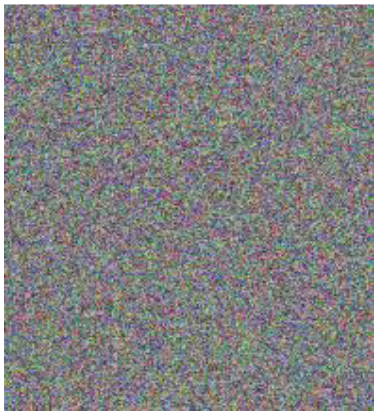
2 pav. Blokinio šifro struktūrinė schema

Elektroninės kodų knygos (ECB) režimas šifruoja kiekvieną bloką individualiai ir nenaudoja inicializacijos vektoriaus. Šį režimą pakankamai saugiu galima laikyti tik tuo atveju jeigu šifruojami blokai yra pakankamai skirtingi ir neturi ilgų pasikartojančių bitų sekų. Šifruojant pasikartojančius duomenų blokus, pvz. rastrinius grafinius paveikslus, tekstinę informaciją ir pan., ECB režime naudojami blokiniai šifrai iš esmės yra nesaugūs (žr. pav. 3, 4ir 5).

3 pav. Pradinis piešinys

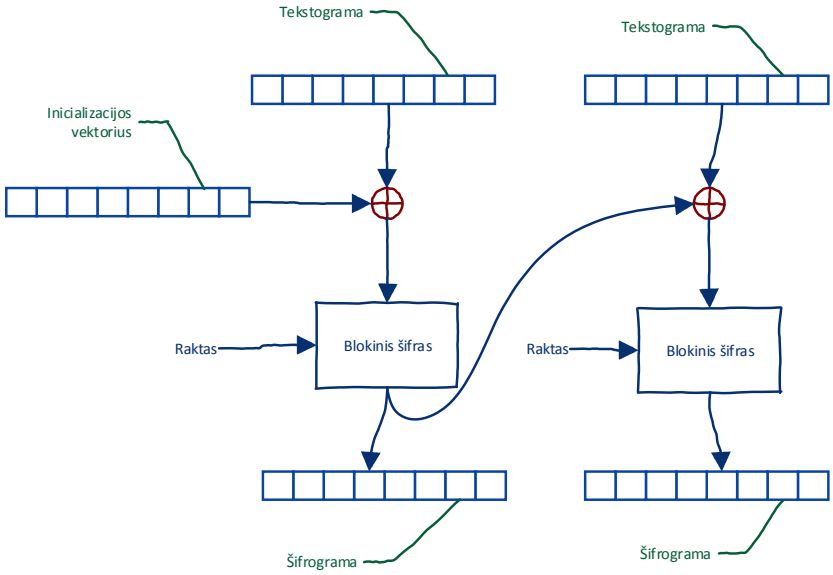


4 pav. Užšifruota naudojant ECB režimą



5 pav. Užšifruota naudojant CBC režimą

Pats paprasčiausias ir greičiausias būdas išvengti vienodų šifrogramų yra panaudoti šifro blokų jungimo (CBC) režimą. Šio režimo veikimo principas pavaizduotas pav. 6. Nuo elektroninės šifrų knygos režimo jis skiriasi tuo, kad pirmas blokas šifruojamas jį sudedant moduliu du su inicializacijos vektoriumi, o vėliau kiekvieno bloko šifrograma sudedama moduliu du su sekančio bloko tekstograma. Iššifravimas vykdomas atvirkščia tvarka.



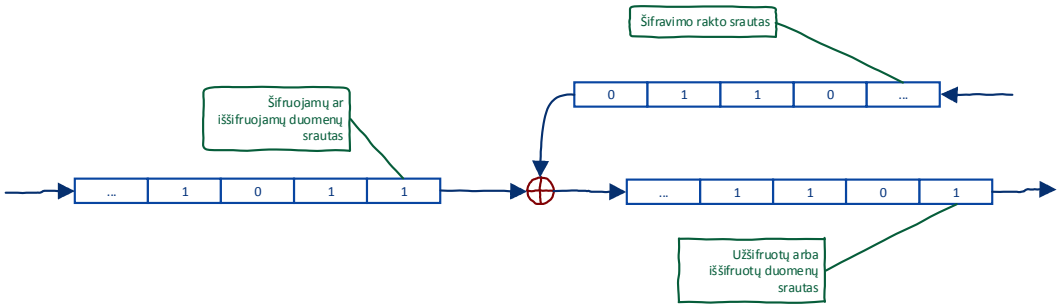
6 pav. Užšifravimo CBC režimu schema

Srautiniai simetriniai šifrai duomenis apdoroja po vieną bitą ir neturi griežtų apribojimų duomenų ilgiui (žr. pav.7). Paprasčiausias srautinio simetrinio šifravimo algoritmas sudarytas iš pseudo atsitiktinių skaičių generatoriaus, kuris naudojamas kaip rakto srauto generatorius ir įeinančių duomenų srauto. Šie du srautai po vieną bitą sumuojami moduliu du (*angl.* XOR). Rezultate gaunamas užšifruotų arba iššifruotų duomenų srautas. Minėtas pseudo atsitiktinių skaičių generatorius privalo būti deterministinis, t.y. jį inicializavus tuo pačiu vektorium jis turi generuoti identišką pseudo atsitiktinių skaičių seką. Jeigu egzistuotų idealus atsitiktinių skaičių generatorius srautiniai šifrai būtų praktiškai neįveikiami, tačiau visi egzistuojantys generatoriai yra pseudo atsitiktiniai todėl egzistuoja galimybė sukompromituoti srautinį šifrą. Siekiant gauti

skirtingas šifrogramas tos pačios tekstogramos šifravimui naudojami skirtingi raktai. Priklausomai nuo šifro rakto srauto generatoriai generuoja skirtingas sekas. Vienas iš būdų susieti šifravimo raktą su generatoriaus generuojama seka galima pasinaudojus maišos funkcija, kurios pradiniai duomenys yra šifravimo raktas, o maišos rezultatas naudojamas kaip generatoriaus inicializacijos vektorius.

Srautiniai šifravimo algoritmai itin gerai tinka naudoti realaus laiko duomenim, nes jų veikimui nereikalingas buferizavimas, kuris realaus laiko sistemose įneša nepageidaujamus vėlinimus. Kadangi srautiniai šifrai operuoja po vieną bitą, juos labai paprasta įgyvendinti aparatinės įrangos priemonėmis. Visai priešingai yra programinės įrangos atveju – procesoriai operuoja registrais, kurių ilgis bitais priklauso nuo procesoriaus architektūros. Šiuo atveju srautiniai šifrai kuriami taip, kad šifruotų ir iššifruotų žodžius (priklausomai nuo architektūros gali būti 8, 16, 32, 64 ir daugiau bitų). Priešingai, nei blokiniai šifrai, srautiniai šifrai labai paprastai aprašomi matematiškai, būtent dėl naudojamų pseudo atsitiktinių skaičių generatorių funkcijų [3].

Vienas plačiausiai praktikoje naudojamų srautinių šifrų yra RC4. Šį šifrą šiame darbe analizuosime ir ištirsime plačiau [4].



7 pav. Srautinio šifro struktūrinė schema

S-Dėžės. Daugelis blokinių ir srautinių šifrų savo darbe naudoja vadinamąsias S-Dėžes (*angl. S-Box*). Šis mechanizmas naudojamas sukeitimams atlikti. S-Dėžės naudojamos įvesti didesnį neapibrėžtumą šifrogramoje, vadinamą Šenono painiava [5]. Šis mechanizmas gautą m bitų ilgio duomenų bloką transformuoja į n bitų ilgio išeities duomenų bloką. Svarbu pabrėžti, kad nebūtinai m turi būti lygu n. Dažniausiai S-Dėžės įgyvendinamos naudojantis statinėmis paieškos lentelėmis (*angl. lookup-table*), nors kai kurie kriptoalgoritmai, kaip pvz. Blowfish ir Twofish, naudoja dinamines paieškos lenteles, kurios generuojamos priklausomai nuo šifravimui naudojamo rakto [6]. Pavyzdžiui DES kriptoalgoritmas naudoja 6x4 S-Dėžę. Pagal 6 įvesties bitus gaunami 4 išvesties bitai. Pirmas ir paskutinis įvesties bitai nurodo S-Dėžės eilutę, o keturi viduriniai – stulpelį. Gražinama, 4 bitų ilgio reikšmė, gražinama iš tos paieškos lentelės celės kurioje susikerta minėta eilutė ir stulpelis.

FL-Blokai. Kai kurie blokiniai šifrai (pvz. Camellia) naudoja vadinamuosius *FL-Blokus*. Šie blokai šifram suteikia papildomo netiesiškumo, ir tuo pačiu apsunkina kriptoanalizę algebriniais metodais. Matematiškai FL-Blokai yra funkcijos priklausančios nuo įvesties kintamojo X ir rakto kintamojo K. Pavyzdžiui Camellia-128 ir Misty1 kriptoalgoritmai naudoja tokias FL-Blokus aprašančias funkcijas [7], [8]:

$$Y_R = X_R \oplus [(X_L \cap K_L) \lll s] \quad (1)$$

$$Y_L = X_L \oplus (Y_R \cup K_R) \quad (2)$$

Šiose funkcijose, konstanta *s* yra priklausoma nuo algoritmo, pvz. Misty1 naudoja 0, o Camellia-128 – 1. Remiantis [7] šios dvi pavyzdinės funkcijos labai lengvai gali būti aprašytos kvadratinių lygčių sistema GF(2) (*angl. Galua Field*), kas leidžia šifrus analizuoti kaip lygčių sistemas.

Blokinių šifrų aprašymas polinomais. Daugelį blokinių šifrų, kurie naudoja S-Dėžes, FL-Blokus ir kitas priemones, galima aprašyti naudojant daugiamačius polinomus. Būdai kuriais kriptoalgoritmus galima aprašyti polinominėmis lygtimis, šiame darbe nenagrinėsime, o pasinaudosime egzistuojančiais darbais šioje srityje. Lygčių, kintamųjų ir polinomų narių skaičius leidžia lengviau įsivaizduoti ir interpretuoti kriptoalgoritmo sudėtingumą matematiškai. Remiantis [7] sudarėme 2 lentelę, kurioje pateikta lygčių ir kintamųjų, kuriais galima pilnai aprašyti mūsų darbe analizuojamus kriptoalgoritmus GF(2) lauke.

2 lentelė. Blokinių šifrų aprašymas GF(2) polinominėmis lygtimis

	AES-128	CAMELLIA-128	Serpent-128
Kintamieji	3296	3584	16640
Tiesinės lygtys	1696	1920	8320
Netiesinės lygtys	4600	4304	9360
Antro laipsnio nariai	12800	11520	13000
Pirmo laipsnio nariai	3296	3584	16640
Laisvieji nariai	9800	8880	11960

AES (RIJNDAEL) šifras. JAV, 1997 metais, NIST (National Institute of Standarts and Technology) paskelbė konkursą teikti paraiškas naujam simetriniam kriptografijos algoritmui, kuris buvo vadinamas AES (Advanced Encryption Standard). Šis naujasis algoritmas turėjo pakeisti DES standartą, kuris buvo naudojamas nuo 1976 metų ir jau buvo morališkai ir technologiškai pasenęs. Kandidatai į AES standartą turėjo atitikti iš anksto iškeltus reikalavimus. Pirmiausia, tai turėjo būti simetrinis algoritmas ir jis turėjo būti atsparus visoms tuo metu žinomoms atakų rūšims. Negana to algoritmas turėjo būti efektyvus ir našus skirtingose platformose, bei įgyvendinant jį aparatūrinėje įrangoje. Algoritmas neturėjo būti pernelyg sudėtingas ir privalėjo palaikyti skirtingus raktų ilgius (128, 192 ir 256 bitų). Koduojamo bloko ilgis turėjo būti 128 bitai. 1998 metais vykusioje AES2 konferencijoje penki algoritmai buvo paskelbti finalistais [9], [10], galiausiai 2001 metais NIST standartizavo Rijndael algoritmą, kaip FIPS PUB 197 standartą – AES.

AES-128, AES-192 ir AES-256 modifikacijos šiuo metu yra plačiausiai naudojamas blokinis šifras su simetriniu raktu [7]. Bandymų nulaužti šį šifrą buvo daug, tačiau nei vienas iš jų nebuvo sėkmingas, neskaitant kelių pavykusių bandymų nulaužti modifikuotą (silpnesnį) AES variantą. Įrodyta, kad galima atkurti AES-128 šifro slaptą raktą, kai naudojami aštuoni šifravimo raundai vietoje, pagal standartą, reikalaujamų dešimties [11]. Autorių pateikiamame darbe [12] sakoma, jog supaprastintą AES-128 šifrą galima sukompromituoti atlikus 2^{48} skaičiavimų. Darant prielaidą, kad supaprastinti AES variantai praktikoje nėra naudojami, minėtas metodas negali būti laikomas realiu pretendentu sukompromituoti šifro algoritmą. Vienintelis realus AES šifro sukompromitavimo būdas buvo paskelbtas 2011 metais, A. Bogdanov‘o, D. Khovratovich‘o ir Ch. Rechberger‘io, tačiau šis metodas praktikoje nepritaikomas, nes už raktų perrinkimo metodą (*angl. bruteforce*) jis greitesnis tik keturis kartus [12]. Šis metodas AES-128

raktą gali atkurti atlikus $2^{126.1}$ operacijų, AES-192 – $2^{189.7}$ operacijų, o AES-256 tik $2^{254.4}$ operacijų.

Lentelėje 3 pateikti, AES standarte [13] leistini pilno Rijndael algoritmo variantai, o lentelėje 2 – algoritmo 128 bitų modifikacijos sudėtingumas matematine išraiška.

3 lentelė. AES algoritmo modifikacijos

Šifro modifikacija	Rakto ilgis, bitai	Bloko ilgis, bitai	Šifravimo raundai
AES-128	128	128	10
AES-192	192	128	12
AES-256	256	128	14

RC4 srautinis simetrinio rakto kriptografinis algoritmas buvo sukurtas Ronaldo Linn Rivesto, tuo metu dirbusio RSA Laboratories Inc. įmonėje. Oficialiai šis algoritmas vadinamas „Ron Cipher 4“. RC4 kriptografinis algoritmas buvo RSA Laboratories komercinė paslaptis ir algoritmo aprašymo įmonė niekada nėra paviešinusi. 1994 metais algoritmo aprašymas buvo anonimiškai paskelbtas Cypherpunks susirašinėjimo grupėje [14], vėliau perskelbtas sci.crypt naujienų grupėse [15]. Po šių dviejų anonimiškų paviešinimų algoritmas labai greitai išplito internete ir netrukus paviešintas kodas buvo pripažintas originaliu, nes jo darbo rezultatai buvo identiški licencijuoto algoritmo bibliotekų rezultatams. RC4 algoritmo specifikacijai išplitus internete jis prarado savo, kaip komercinės paslapties esmę, ir šiuo metu RC4 tėra registruotas prekinis ženklas. Siekiant išvengti neteisėto prekinio ženklo naudojimo RC4 algoritmas daug kur vadinamas ARCFOUR. RC4 tapo vienu plačiausiai naudojamų simetrinio rakto srautinių algoritmų, jį galima sutikti tokiuose saugos protokoluose ir standartuose kaip WEP, WPA, SSL, TLS ir daugelyje kitų. Toks algoritmo populiarumas buvo nulemtas jo greičio ir paprastumo, efektyvaus įgyvendinimo programinėje ir aparatinėje įrangose, bei itin paprasto programavimo.

RC4 matematiškai aprašomas labai paprastai. Algoritmas veikia OFB režimu, t.y. jis nepriklauso nuo tekstogramos turinio. Jis naudoja simetrinę, 8x8 dydžio, S-Box lentelę, tai reiškia, kad 8 įvesties bitai gražins 8 išvesties bitus. RC4 kriptoalgoritmas naudoja du vidinius skaitiklius: *i* ir *j*, kurie pradžioje inicializuojami nuliais. Atsitiktinio baito, naudojamo srautiniam šifravimui generavimas vykdomas:

$$i = (i + 1) \bmod 256$$
$$j = (j + S_i) \bmod 256$$
$$swap^{1}\ S_i,\ S_j$$
$$t = (S_i + S_j) \bmod 256$$
$$K = S_t$$

Gautasis baitas *K* yra naudojamas šifravimui arba iššifravimui – operacijos sudėtis moduliu 2 (*XOR*) pagalba jis sudedamas su tekstogramos arba šifrogramos baitu. Dėl savo paprastumo RC4 kriptoalgoritmas yra apie 10 kartų greitesnis nei DES [15]. Pradinė S-Dėžės užpildymo operacija taip pat yra labai paprasta. Pirma ji užpildoma nuosekliai didėjančiomis reikšmėmis: $S_0=0,\ S_1=1,\ ...,\ S_{255}=255$, o laikinas tokio pačio dydžio masyvas *K* nuosekliai užpildomas šifravimo (iššifravimo) rakto reikšmėmis, pakartojant raktą tiek kartų kiek reikia, kad pilnai užpildyti 256 baitų masyvą. Būtent dėl šio S-Dėžės inicializavimo ir gaunamas maksimalus RC4 rakto ilgis lygus 2048 bitų.

Tada atliekamos tokios operacijos:

[1] *swap* pseudo operacija, sukeičia argumentų kintamųjų reikšmes vietomis.

```
for i = 0 to 255
begin
   j = (j + Si + Ki) mod 256
   swap Si, Sj
end
```

Aukščiau buvo pateiktas sutrumpintas, tačiau visus reikiamus veiksmus aprašantis algoritmas, kurį įgyvendinus rezultatas bus suderinamas su licencijuotu RC4 algoritmu. Nepaisant itin paprasto RC4 kriptoalgoritmo aprašymo jis yra pakankamai stiprus – teoriškai jis gali turėti 2^{1700} (256! * 256^2) vidinių būsenų, o tai yra neįtikėtinai didelis skaičius [3], [4]. Šio darbo autoriams yra tekę matyti pavyzdžių, kaip RC4 įgyvendinamas naudojant T-SQL kalbą, tačiau patikrinti ar sprendimas pilnai atitinka RC4 specifikaciją nebuvo galimybės.

Camellia simetrinis, blokinis, šifras buvo sukurtas bendromis Nippon Telegraph and Telephone Corporation (NTT) ir Mitsubishi Electric Corporation pajėgomis. Pirmą kartą kriptoalgoritmo specifikacija buvo viešai publikuota 2000 metais, vėliau 2001 metais, ji buvo papildyta ir ištaisytos pastebėtos klaidos [8]. Šifras buvo kuriamas atsižvelgiant į paprastą įgyvendinimą tiek programinėmis tiek ir aparatinėmis priemonėmis. Algoritmas naudoja 128 bitų ilgio blokus ir 128, 192 arba 256 bitų ilgio raktus (žr. 4 lentelę). Camellia kriptoalgoritmas tiek savo įgyvendinimu tiek ir našumu lygiuojasi su AES šifru [16]. Oficialiai Camellia šifras yra standartizuotas ISO/IEC bei NESSIE ir CRYPTREC projektų. Nors Camellia kriptoalgoritmas ir yra patentuotas, patento savininkai algoritmą leidžia naudoti ir su atviromis licencijomis, tai leido jam būti įtrauktam į OpenSSL, Mozilla NSS, FreeBSD, Crypto++, GnuTLS, PolarSSL bibliotekas, šifro praktinis panaudojimas yra dokumentuotas ir keliuose RFC dokumentuose. Tačiau nepaisant to, kad Camellia kripto algoritmas yra naudojamas daugelyje bibliotekų ir atsižvelgus į tai, kad šio šifro saugumas lygiuojamas kartu su AES šifru [17], [7], jis nėra labai populiarus tarp mokslininkų siekiančių išsiaiškinti kriptografinių algoritmų silpnas vietas.

4 lentelėje pateikti, Camellia specifikacijoje [8] aprašytos galimos šifro modifikacijos, o 2 lentelėje – algoritmo 128 bitų modifikacijos sudėtingumas matematine išraiška, sudarytas remiantis [7].

4 lentelė. Camellia algoritmo modifikacijos

Šifro modifikacija	Rakto ilgis, bitai	Bloko ilgis, bitai	Šifravimo raundai
Camellia-128	128	128	18
Camellia-192	192	128	24
Camellia-256	256	128	24

Serpent blokinis, simetrinis kriptografinis algoritmas [18] AES konkurso finale užėmė antrą vietą ir jį aplenkė tik Rijndael. Serpent šifras buvo kuriamas mokslininkų Ross Andersono, Eli Bihamo ir Lars Knudseno. Taip vadinama Serpent-0 modifikacija pirmą kartą buvo pristatyta penktojoje Fast Software Encryption konferencijoje 1998 metais Paryžiuje. Vėliau AES kripto konkursui autoriai modifikavo pirminę versiją ir vertinimui pateikė Serpent-1 modifikaciją. Kadangi Serpent algoritmas buvo siūlomas kandidatu į AES standartą, jis šifruoja 128 bitų ilgio blokus ir naudoja 128, 192 ir 256 bitų ilgio raktus. Serpent kripto algoritmas naudoja 32 raudus sukeitimo-perstatymo operacijų. Kiekvieno raundo metu 32 kartus panaudojama viena iš aštuonių 4x4 S-Dežių. Pastarosios operacijos gali būti atliekamos lygiagrečiai, tai leidžia išlygiagretinus šifravimo/iššifravimo operacijas pasiekti itin aukštą

našumą, tačiau tai įgalina Serpent kompromitavimui panaudoti itin daug kriptoalatinių darbų pritaikytų DES algoritmui [19], [17].

Visos žinomos atakos prieš Serpent šifrą skaičiavimų požiūriu yra netinkamos praktiniam panaudojimui. 2011 metais [20] aprašytos tuo metu žinotos galimos atakos prieš Serpent šifrą:

- 11 raundų bet kokio rakto ilgio Serpent šifro sukompromitavimui reikia 2^{116} žinomų tekstogramų ir užima $2^{107,5}$ laiko;
- 12 raundų Serpent-256 šifrui yra aprašomos dvi galimos atakos:
  - Pirmoji reikalauja žinoti 2^{118} tekstogramų ir $2^{228,8}$ laiko;
  - Antroji reikalauja žinoti 2^{116} tekstogramų ir $2^{237,5}$ laiko.

Iš aukščiau pateiktų reikalavimų, nei vienas iš žinomų metodų praktikoje naudoti nėra tikslingas, nes nedaug skiriasi nuo galimų raktų perrinkimo atakos (*angl. bruteforce attack*). Vienintelė XSL ataka [7] gali susilpninti Serpent šifrą pakankamai, kad taptų realiau jį sukompromituoti. Beje verta paminėti, kad [7] darbe buvo minima, jog Serpent algoritmą XSL ataka susilpnina mažiau nei Rijndael, kuriam jis pralaimėjo AES konkurso finale.

5 lentelėje pateiktos, Serpent šifro galimos modifikacijos, o 2 lentelėje – algoritmo 128 bitų modifikacijos sudėtingumas matematine išraiška, sudarytas remiantis [7].

5 lentelė. Serpent algoritmo modifikacijos

Šifro modifikacija	Rakto ilgis, bitai	Bloko ilgis, bitai	Šifravimo raundai
Serpent-128	128	128	32
Serpent-192	192	128	32
Serpent-256	256	128	32

1.1.2. Asimetriniai šifrai

1976 metais Whitfield Diffieis ir Martin Hellmanas visiems laikams pakeitė kriptografijos paradigmą [4]. Jie aprašė viešo rakto kriptografijos principus (*angl. Public Key Infrastructure, PKI*). Minėtų dviejų autorių darbuose buvo atskleista kaip galima kriptografijai panaudoti du skirtingus raktus – vieną viešą, kitą privatų (slaptą). Viešo rakto kriptografijos stiprioji pusė yra, kad abudu raktai yra matematiškai susieti tarpusavyje, tačiau paskaičiuoti privatųjį raktą iš viešojo yra labai sudėtinga. Viešasis raktas gali būti žinomas bet kam (iš čia ir kilęs jo pavadinimas) ir jo pagalba tekstogramą galima tik užšifruoti, norint ją iššifruoti reikia žinoti privatų raktą. Vienas iš naudingesnių PKI privalumų yra tas, kad išsprendžiama raktų dalijimosi problema – viešąjį raktą galima paskelbti nors ir Facebook socialiniame tinkle. Tačiau šie principai turi ir trūkumą – užtikrinti jų saugumą reikia naudoti pakankamai ilgus raktus ir reikia operuoti labai dideliais skaičiais, tai lemia pakankamai lėtą šifravimą ir iššifravimą. [4] teigiama, kad praktikoje asimetrinė kriptografija nėra tinkama, kaip simetrinio rakto pakaitalas. Jie naudojami ne tekstogramoms, bet simetriniams raktams šifruoti. Pagrindinė to priežastis yra ta, kad asimetrinės kriptografijos algoritmai yra lėti, mažiausiai 1000 kartų lėtesni nei simetriniai algoritmai. Apjungiant asimetrinės kriptografijos algoritmus (raktų šifravimui) su simetrinės kriptografijos (tektogramų šifravimui) algoritmais gaunama, taip vadinama hibridinė kriptosistema.

Kadangi asimetrinės kriptografijos algoritmai yra apsukami, t.y. užšifruoti galima ir privačiu raktu, o iššifruoti viešuoju, ją galima panaudoti kaip skaitmeninį parašą. Šiuo atveju paskaičiuojama pasirašomo dokumento santrauka (*angl. hash*) ir gauta trumputė tekstograma užšifruojama privačiu pasirašančiojo raktu ir gauta šifrograma prijungiama prie pasirašyto

dokumento. Vėliau bet kas turintis pasirašiusiojo viešąjį raktą gali iššifruoti santrauką, paskaičiuoti dokumento santrauką ir palyginti gautąją iššifravus su paskaičiuotąją. Sutampančios santraukos reiškia, kad dokumentas buvo pasirašytas asmens (ar sistemos) kurios viešuoju raktu buvo iššifruota parašo šifrograma. Papildomai, tai leidžia užtikrinti, kad po pasirašymo dokumentas nebuvo pakeistas, o šios funkcijos negali atlikti net ranka suraitytas parašas ant popieriaus lapo. Jeigu pasirašant dokumentą papildomai yra panaudojama ir laiko žymė, kada tai buvo atlikta, tai leidžia užtikrinti ir parašo neišsiginamumą.

Didžioji dauguma šiuo metu naudojamų asimetrinės kriptografijos metodų naudoja du skirtingus principus, kurie abudu susiveda į kol kas labai sunkiai išsprendžiamus matematinius uždavinius:

1) Didelių skaičių, kurie buvo gauti sudauginus du didelius pirminius skaičius, faktorizavimo (daugiklių suradimo) uždavinio;
2) Diskrečiojo logaritmo baigtiniame lauke uždavinys;

Dar yra ir trečiasis variantas naudojamas kriptografijoje – eliptinių kreivių kriptosistemos. Šį mechanizmą 1985 metais dirbdami atskirai pasiūlė Neal Koblitzas [21] ir V.S. Milleris [22]. Iš esmės, tai ta pati viešojo rakto kriptografija, tik jos įgyvendinimui naudojamos eliptinių kreivių matematinės savybės.

Kadangi visos šiuo metu naudojamos viešojo rakto kriptosistemos remiasi neišspręstais matematiniais uždaviniais, tiksliau uždaviniais kuriems spręsti nėra greitų algoritmų, jos laikomos saugiomis. Tačiau nėra aišku, kada gali įvykti proveržis matematikoje arba kompiuteriuose, pvz. bus galiausiai sukurtas kvantinis kompiuteris, ir tam įvykus visi šie algoritmai taps visiškai nesaugūs.

RSA šifras, bene populiariausias, viešo rakto kriptografinis algoritmas. RSA yra paremtas skaičių faktorizavimo problema, t.y. nėra našių algoritmų kurie galėtų pakankamai greitai išskaidyti didelį skaičių pirminiais dauginamaisiais. RSA kriptosistema buvo sukurta 1978 metais bendromis Rivest, Shamir ir Adleman jėgomis [23] ir iki šiol nėra nulaužta. Nors iš pirmo žvilgsnio RSA naudojami matematiniai pagrindai atrodo labai paprasti, tačiau jie yra itin imlūs skaičiavimams. Pavyzdžiui užšifravimas (2) ir iššifravimas (3) matematiškai aprašomi taip:

$$c = m^e (mod\ n) \tag{3}$$

$$m = c^d (mod\ n) \tag{4}$$

Nors atrodo ir paprasta pakelti skaičių laipsniu ir paimti šios operacijos liekaną dalinant iš kito sveiko skaičiaus, tačiau reikia nepamiršti, kad rekomenduojami raktų ilgiai [16] yra nuo 1024 bitų liktinėm sistemom iki 2432 bitų naujai kuriamom sistemom. Net ir paprastos matematinės operacijos su tokios eilės skaičiais yra sudėtingas uždavinys. Kaip jau ir minėjome, dėl šios priežasties asimetriniai kriptografiniai algoritmai yra labai imlūs skaičiavimams ir tuo pačiu energijos sąnaudoms. Tai yra pagrindinė problema, kodėl asimetriniai kriptografiniai algoritmai dideliems duomenų kiekiams šifruoti naudojami tik išimtinais atvejais.

DSA šifras yra FIPS elektroninio parašo standarto sudedamoji dalis. Savo prigimtimi DSA algoritmas yra Shnorr ir ElGamal elektroninio parašo sistemų variantas ir remiasi diskrečiojo logaritmo uždaviniu. Šis algoritmas elektroninio parašo standartui buvo pasiūlytas NIST organizacijos 1991 metais kaip FIPS 186 specifikacija [24]. Naudojimui šis standartas buvo priimtas 1993 metais. Vėliau standartas buvo nežymiai patobulintas 1996 metais (FIPS 186-1), vėliau jis buvo išplėstas 2000 metais (FIPS 186-2) ir 2009 (FIPS 186-3). Pradžioje David

W. Kravitzas DSA algoritmą buvo patentavęs JAV patentų biure kaip buvęs NSA organizacijos darbuotojas. Vėliau patentas buvo perduotas JAV Vyriausybei, ko pasekoje NIST organizacija šį patentą neatlygintinai leido naudoti visiem norintiems.

DSA algoritmas naudoja tokius parametrus [4]:

p = pirminis L bitų ilgio skaičius, čia L yra naudojamo rakto ilgis bitais;
q = 160 bitų ilgio skaičiaus $p - 1$ pirminis dauginamasis;
$g = h^{(p-1)/q} \bmod p$, čia h yra bet koks skaičius mažesnis už $p - 1$, tačiau toks, kad rezultatas būtų didesnis už vienetą;
x = skaičius mažesnis už q;
$y = g^x \bmod p$;

Papildomai algoritmas dar naudoja SHA-1 arba SHA-2 vienkryptę funkciją, pasirašomos žinutės santraukai paskaičiuoti – $H(m)$. Pirmi trys parametrai p, q ir g yra vieši ir gali būti naudojami bendrai daugelio šalių. Privatus raktas šiuo atveju bus x, o viešasis – y.

Žinutės pasirašymui yra sugeneruojamas atsitiktinis skaičius k, mažesnis už q. Pasirašant sugeneruojamas parašas:

$$r = (g^k mod\ p)\ mod\ q \tag{5}$$

$$s = \left(k^{-1}(H(m) + xr)\right)\ mod\ q \tag{6}$$

Atlikus šias operacijas, gautieji r ir s ir yra skaitmeninis parašas. Parašo patikrinimui skaičiuojama:

$$w = s^{-1}\ mod\ q \tag{7}$$

$$u_1 = (H(m) \cdot w)\ mod\ q \tag{8}$$

$$u_2 = (rw)\ mod\ q \tag{9}$$

$$v = \left((gu_1 \cdot yu_2)\ mod\ p\right)\ mod\ q \tag{10}$$

Jeigu gaunama, kad $v = r$, tai reiškia, jog parašas yra tikras.

DSA algoritme vienas svarbiausių parametrų saugumui užtikrinti yra atsitiktinis skaičius k. Jeigu keliems pasirašymams bus panaudota ta pati k reikšmė, bus galima atkurti privatų raktą [16]. Kai kuriais atvejais užtenka k panaudoti dviem parašam, kad būtų įmanoma praktiškai atkurti privatųjį raktą.

ECDSA šifras yra DSA algoritmo variacija kuri parašo generavimui naudoja eliptinių kreivių kriptografiją baigtiniuose laukuose. Eliptinių kreivių panaudojimo kriptografijai galimybes aprašė Neal Koblitz ir Victor S. Miller savo darbuose [21], [22]. Didžiausias eliptinių kreivių asimetrinės kriptografijos privalumas yra tas, kad manoma, jog rakto ilgis turi būti tik du kartus didesnis nei pageidaujamas saugos lygis. pavyzdžiui 80 bitų saugos lygmeniui įveikti, įsibrovėlis turėtų sugeneruoti 2^{80} elektroninių parašų, kad surasti naudojama privatųjį raktą. Tokio lygio saugai užtikrinti DSA algoritmas turėtų naudoti bent jau 1024 bitų ilgio raktą, o ECDSA algoritmui pakanka 160 bitų ilgio rakto [16].

ECDSA pasirašymas ir parašo tikrinimas yra ganėtinai panašus į DSA, tačiau remiasi sudėtingais matematiniais pagrindais ir šiame darbe jų nenagrinėsime. Tačiau norime pabrėžti, jog ECDSA algoritmo atveju naudojamas atsitiktinis parametras k yra toks pat kritiškas saugos

atžvilgiu kaip ir DSA algoritmo atveju. Pavyzdžiui korporacija Sony savo žaidimų kompiuterių Play Station 3 programinės įrangos, iki versijos 3.55, pasirašymui ECDSA algoritmu naudojo statinę *k* reikšmę. Tai leido piratams paskaičiuoti Sony naudojamą privatųjį raktą, ko pasekoje tapo realu konsolėje paleisti nelegalias žaidimų kopijas ir netgi įkelti neautorizuotą operacinę sistemą – Sony naudota apsauga buvo visiškai sukompromituota. Šiai spragai panaikinti Sony išleido naują OS versiją, naudojančią naujus raktu, o naujų žaidimų gamintojai privalėjo tikrintis OS versiją ir aptikus seną, sukompromituotą, versiją neleisti žaidimui veikti. Tačiau tai neapsaugojo nuo nelegalių senesnių žaidimų kopijų sėkmingo paleidimo naudojant senąją OS.

ElGamal viešojo rakto kriptografinė sistema yra Diffie-Hellman raktų apsikeitimo algoritmo veikiančio raktų perdavimo režime [25]. Kriptosistemos saugumas paremtas diskrečiojo logaritmo ir Diffie-Hellman uždaviniais. Ši kriptosistema pirmą kartą buvo aprašyta Taher ElGamalio 1984 metais [26]. DSA elektroninio parašo algoritmas yra ElGamal elektroninio parašo sistemos modifikacija. ElGamal kriptosistema naudojama tokiose sistemose, kaip GNU Privacy Guard, PGP ir kitose.

ElGamal kriptoalgoritmas inicializuojamas sugeneruojant raktų porą. Tam pasirenkamas didelis pirminis skaičius *p* ir sugeneruojami du atsitiktiniai pirminiai skaičiai *g* ir *x*, tokie, kad būtų mažesni už *p*. Tada paskaičiuojama:

$$y = g^x mod\ p \tag{11}$$

Iš čia gauname viešąjį raktą *y*, *g* ir *p*, kur *g* ir *p* gali būti laisvai naudojamas didelės vartotojų grupės. Privatusis raktas yra *x*.

Pasirašymas naudojantis ElGamal kriptosistema vykdomas taip:

- Sugeneruojamas viešojo rakto *p*, $g<p$, $y=g^x \bmod p$ ir privataus rakto $x<p$ pora;
- Pasirenkamas atsitiktinis skaičius *k*, toks, kad skaičiai *k* ir *p-1* būtų tarpusavyje pirminiai;
- Pasirašoma tekstograma:
  - $a = g^k \bmod p$
  - Parenkamas *b* toks, kad $M=(xa + kb) \bmod (p-1)$
- Parašas laikomas galiojančiu jeigu $y^a a^b \bmod p = g^M \bmod p$

Nedidelė ElGamal algoritmo modifikacija leidžia šifruoti tekstogramas. Tekstogramos *M* užšifravimas ir iššifravimas vykdomas taip:

- Sugeneruojamas viešojo rakto *p*, $g<p$, $y=g^x \bmod p$ ir privataus rakto $x<p$ pora;
- Pasirenkamas atsitiktinis skaičius *k*, toks, kad skaičiai *k* ir *p-1* būtų tarpusavyje pirminiai;
- Užšifruojama:

$$a = g^k\ mod\ p \tag{12}$$

$$b = y^k M\ mod\ p \tag{13}$$

- Iššifruojama:

$$M = \frac{b}{a^x} mod\ p \tag{14}$$

Iš pateiktų aukščiau matematinių išraiškų ElGamal, kaip ir kiti šiame darbe aprašyti viešojo rakto kriptografiniai algoritmai, atlieka itin elementarias aritmetines operacijas, tačiau operuojama itin dideliais skaičiais (512 ir daugiau bitų ilgio).

1.1.3. Maišos funkcijos

Vienkryptės maišos funkcijos literatūroje vadinamos įvairiai: suspaudimo funkcijos, santraukos funkcijos, pranešimo santraukos, skaitmeniniais pirštų atspaudais, kriptografinėmis kontrolinėmis sumomis, pranešimų vientisumo tikrinimo funkcijomis, manipuliacijų aptikimo kodu ir t.t. [25], [4]. Tačiau, kad ir kaip maišos funkcijos būtų vadinamos jos yra vienas iš kertinių šiuolaikinės kriptografijos pagrindų ir jos įeina į daugelį kriptografinių protokolų ir algoritmų. Pavyzdžiui dauguma elektroninio parašo algoritmų pirmiausia skaičiuoja pasirašomos tekstogramos maišą ir pasirašo tik tais ją – priešingu atveju kelių gigabaitų dydžio failo pasirašymas užtruktų amžinybę.

Maišos funkcija yra tokia matematinė funkcija kuriai padavus bet kokio ilgio argumentą jinai gražins fiksuoto ilgio rezultatą. Pati paprasčiausia, bet kriptografiniu požiūriu visiškai nesaugi, maišos funkcija būtų tokia kuri visus įvesties baitus sudeda moduliu du (*XOR* funkcija) ir grąžina vieno baito ilgio rezultatą. Nesunku nuspėti, kad tokios funkcijos atveju būtų labai nesudėtinga sugeneruoti eilę pradinių duomenų kurie gražintų tą patį rezultatą. Kriptografiškai saugi maišos funkcija turi tenkinti tris sąlygas:

- Fiksuoto ilgio – bet kokiai įvesties kombinacijai turi gražinti fiksuoto ilgio rezultatą;
- Vienkryptė – neturi būti paprasta iš maišos reikšmės paskaičiuoti pradinį pranešimą;
- Atspari kolizijoms – turi būti sudėtinga rasti skirtingas įvesties kombinacijas kurios duotų tą pačią maišos reikšmę.

Didžioji dauguma praktikoje naudojamų maišos funkcijų algoritmų yra vieši – nėra prasmės slėpti jų veikimo principus, nes saugumas užtikrinamas šių funkcijų vienkryptiškumu. Saugios maišos funkcijos garantuoja, kad pasikeitus vienam bitui įvesties duomenyse, maišos rezultate pasikeis bent jau pusė visų bitų reikšmių [4]. Taip pat jos užtikrina, kad įvesties duomenų paskaičiavimas turint maišos rezultatą būtų betikslis dėl itin sudėtingų ir daug laiko reikalaujančių skaičiavimų – praėjus laikui sugaištam bandant surasti pirminius duomenis iš jų jau nebebus jokios realios naudos.

MD5 maišos funkcija yra viena iš plačiausiai naudojamų santraukos skaičiavimo algoritmų. Pilnas kurios pavadinimas anglų kalboje yra „Message-Digest Algorithm“. Šis algoritmas skaičiuoja 128 bitų ilgio santraukas ir yra specifikuotas RFC-1321 dokumente [27]. Pirmą kartą buvo aprašytas ir paskelbtas Ronald Rivesto 1992 metais [28]. Dažniausiai MD5 santrauka užrašoma 32 skaitmenų šešioliktainiu skaičiumi, pvz.:

```
9e107d9d372bb6826bd81d3542a419d6.
```

Šiuo metu, tai turbūt nesaugiausias santraukos skaičiavimo algoritmas iš labiausiai naudojamų. Pirmos saugumo problemos buvo pastebėtos dar 1996 metais, t.y. praėjus tik 4 metam po Ron‘o Rivest‘o algoritmo viešo paskelbimo. Galiausiai 2004 metais buvo rasta kritinių saugumo problemų – buvo atrastas būdas, kaip sugeneruoti du failus kurių MD5 santraukos bus identiškos. Nuo to laiko MD5 nerekomenduotinas naudoti itin didelio slaptumo reikalaujančiuose uždaviniuose, kaip to įrodymas tapo 2008 metų gruodį pademonstruota galimybė suklastoti SSL sertifikatą naudojantį MD5 santraukų skaičiavimo funkciją [29]. Dėl šių priežasčių MD5 šiuo metu naudotinas tik kaip failų integralumo tikrinimo funkcija, kur saugos ir kolizijų aspektai visiškai neaktualūs.

ECRYPT 2 pateikiamos tokios išvados dėl MD5 naudojimo [16]:

- Kolizijas rasti nereikalingi dideli skaičiavimų pajėgumai, kai tekstogramos ilgis apribojamas 596 bitais;
- Viešo rakto sertifikatuose naudojančiuose MD5 maišos funkciją buvo aptikta kolizijų. Buvo pademonstruotas praktinis sertifikato suklastojimo pavyzdys;
- Slaptažodžių, saugomų kaip MD5 maišos rezultatai, atkūrimo atakos yra žinomos;
- Žinoma pradinės tekstogramos atstatymo ataka, kurios sudėtingumas $2^{124,4}$.

Vienkrypčių funkcijų SHA (Secure Hash Algorithm) šeima yra standartizuota FIPS standartuose. Iš viso yra trys standartizuotos SHA algoritmų versijos: SHA-0, SHA-1 ir SHA-2. Pirmosios dvi versijos skaičiuoja 160 bitų santrauką, o antroji – 256 arba 512 bitų. SHA-2 atveju dar egzistuoja sutrumpintos maišos variacijos kurios skaičiuoja 224 ir 384 bitų ilgio maišas. SHA-0 versija nėra naudojama, nes joje yra rasta kolizijų ir ji skaitoma nesaugia. SHA-1 nuo SHA-0 skiriasi tik santraukos rezultato išplėtimu ir yra laikoma ankstesnio standarto išplėtimu. 2012 metų spalio mėnesį NIST organizacija paskelbė SHA-3 algoritmo konkurso nugalėtoją – Keccak maišos funkcija [30], kuri šio darbo rašymo metu dar nebuvo standartizuota.

Šiuo metu šis algoritmas yra standartizuojamas, tačiau nėra planų juo pakeisti SHA-2 algoritmą, nes kol kas dar nėra aptikta jo saugumo spragų. NIST SHA-3 konkursą organizavo labiau tam, kad būtų sukurtas naujas standartas, iš esmės besiskiriantis nuo egzistuojančių, jeigu SHA-2 būtų sukompromituotas. Netgi senajame SHA-1 algoritme kol kas nėra rasta įrodytų silpnumų ir jis laikomas saugiu. Teoriškai yra aprašytas būdas leidžiantis sukompromituoti SHA-1 maišos funkciją atlikus 2^{51} sudėtingumo skaičiavimų [31]. SHA-1 algoritmas, šio darbo rašymo metu, yra plačiausiai naudojamas santraukos reikšmių skaičiavimui. Jis naudojamas tokių protokolų įgyvendinimui, kaip: TLS, SSL, PGP, SSH, S/MIME ir IPSec. Taipogi SHA-1 yra naudojama išeities kodo kontrolės sistemose: Git, Mercurial, Monotone, kaip failų skaitmeniniai pirštų antspaudai (angl. *fingerprints*), pagal kuriuos šie failai yra versijuojami.

2011 metų vasario mėnesio duomenimis [32] NIST organizacija yra patvirtinusi daugiau nei 1400 SHA-1 algoritmo implementacijų. Vienas įdomesnis faktas apie SHA-1 yra, tas, kad šis algoritmas buvo naudojamas Nintendo kompanijos žaidimų apsaugai nuo nesankcionuoto kopijavimo, tačiau įgyvendinime buvo įvelta klaida kuria pasinaudojo žaidimų piratai [33]. Klaidos esmė, buvo ta, kad tikrinant santraukas buvo naudojama C kalbos funkcija *strncmp*, kuri tikrina simbolių eilutę iki pirmo nulinio baito ir jį radus gražina teigiamą atsakymą. Atitinkamai suformavus santraukas buvo galima priversti OS tikrinti tik pirmus 8 santraukos bitus – telieka tik 2^8 kombinacijos ir perrinkti tokį kiekį duomenų, kad suformuoti reikiamus pradinius duomenis yra trivialus uždavinys.

SHA-2 algoritmų šeima buvo patvirtinta, kaip FIPS standartas 2001 metais. Šią šeimą sudaro keturios algoritmo versijos kurios skaičiuoja santrauką 256, 224, 512 ir 384 bitų ilgiais. Kaip buvo minėta ankstesniame skyrelyje SHA-1, teoriškai jau yra sukompromituotas [31], todėl šiuo metu NIST rekomenduoja naudoti SHA-2. Kaip ir SHA-1 atveju, SHA-2 yra gana plačiai naudojamas. Labiausiai žinomi protokolai, kuriuose, galima pasirinkti SHA-2 naudojimą yra: TLS, SSL, PGP, SSH, S/MIME, IPSec ir BitCoin. Kaip matome, pirmąjį ir antrąjį SHA algoritmus naudoti galima su tais pačiais protokolais, tereikia tik pasirinkti tinkamą algoritmą kiekvienai situacijai individualiai. SHA-1 ir SHA-2 algoritmai yra naudojami ir DSA skaitmeninio parašo sistemoje. SHA-2/512 atveju algoritmo veikimo greitis labai priklauso nuo procesoriaus architektūros, kaip teigiama [34] 64 bitų architektūros kompiuteriuose SHA-2/512 skaičiuojamas žymiai greičiau, nei SHA-2/256.

Whirlpool santraukų skaičiavimo algoritmas nėra taip plačiai naudojamas, kaip aukščiau aprašytieji SHA-1/2 ir MD5 algoritmai, tačiau jis 2004 metais buvo standartizuotas

International Organization of Standartization (ISO) ir International Electrotechnical Commision (IEC), kaip jungtinis ISO/IEC 10118-3 standartas. Taip pat jis yra rekomenduojamas NESSIE projekto. WHIRPOOL algoritmą 2000 metais sukūrė Vincent Rijmeno (vienas iš Rijndael kriptoalgoritmo autorių) ir Paulo S. L. M. Baretto duetas. Pagrindinis jo skirtumas nuo ankstesniuose skyreliuose aprašytųjų algoritmų yra tas, kad santraukos funkcijai skaičiuoti yra naudojamas taip vadinamas *s*-box šifravimo algoritmas, todėl pradiniame pranešime pasikeitus tik vienam bitui, santrauka pakinta kardinaliai. WHIRPOOL skaičiuoja 512 bitų ilgio santraukas. Šio santraukų skaičiavimo algoritmo autoriai, aprašymo dokumente, pateikia ir bazines skaičiavimo funkcijas parašytas C ir Java programavimo kalbomis [35]. Kadangi Whirlpool maišos algoritmas nėra toks populiarus kaip MD5 ir SHA šeima nėra ir tiek daug mokslinių darbų bandančių rasti jo silpnumus. Jeigu Whirlpool būtų naudojamas sertifikatuose ir elektroninio parašo algoritmuose situacija su jo analize būtų visai kitokia.

1.2. Šifrų raktų ilgių analizė

Siekiant užtikrinti pakankamą saugą ir neperlenkti lazdos su per stipria sauga reikia teisingai parinkti kriptoalgoritmų naudojamus raktų ilgius. Per trumpas raktas nesuteiks pakankamos saugos nuo tikėtinų įsilaužimų, o per ilgas lėtins sistemos darbą ir, mobilių įrenginių atveju, be reikalo eikvos įrenginio baterijoje sukauptą energiją ir trumpins jo darbo laiką.

Lentelėje 6 pateiktos ECRYPT II metinėje 2010-2011 ataskaitoje [16] pateiktos rekomendacijos naudotinų šifravimo raktų ilgiam norint pasiekti atitinkamą saugos lygį, o lentelėje 7 – minimalūs rekomenduotini raktų ilgiai apsaugai nuo skirtingus skaičiavimo ir finansinius resursus turinčių įsilaužėlių. Rakto ilgis pateikiamas simetrinių šifrų raktų ilgių ekvivalentu. Norint palyginti simetrinių ir asimetrinių šifrų raktų ilgius reikia įvesti santykinį rakto ilgį. Pagal ECRYPT II rekomendaciją, rakto ilgio baziniu vienetu pasirinktas simetrinių blokinių šifrų rakto ilgis. Asimetrinių algoritmų raktų ilgiai, remiantis algoritmo sudėtingumu, perskaičiuojami taip, kad atitiktų blokinių raktų ilgius. Šis ekvivalentiškumas pateiktas lentelėje 8.

6 lentelė. Saugos lygmenų ir raktų ilgių suvestinė

Saugos lygis	Rakto ilgis, bitais	Saugos aprašas	Komentaras
1	32	Realaus laiko atakos atliekamos individulių programišių	Tinkama tik autentifikacijos žymėms ir kitiem trumpalaikės saugos reikalaujantiem duomenim šifruoti
2	64	Itin trumpalaikė apsauga nuo mažų organizacijų atakų	Neturėtų būti naudojamas konfidencialumui užtikrinti kuriant naujas sistemas
3	72	Trumpalaikė apsauga nuo vidutinio dydžio organizacijų, vidutinio ilgio apsauga nuo mažų organizacijų	
4	80	Itin trumpalaikė apsauga nuo valstybinių organizacijų, ilgalaikė apsauga nuo mažų organizacijų	Mažiausias rekomenduotinas bendro pobūdžio saugos lygis (apsauga mažiau nei 4 metams)
5	96	Liktinių sistemų standartinis saugos lygis	Apsauga apytiksliai 10 metų
6	112	Apsauga vidutinio laikotarpiui	Apsauga apytiksliai 20 metų
7	128	Ilgalaikė apsauga	Rekomenduojamas saugos lygis, apsauga apytiksliai 30 metų

8	256	Apsauga „numatomai ateičiai“	Gera apsauga nuo kvantinių kompiuterių galimybių, išskyrus atvejį jeigu Shor‘o algoritmas [36] pasitvirtintų.

7 lentelė. ECRYPT2 rekomenduojami raktų ilgiai pagal įsilaužėlių tipus

Įsilaužėlio tipas	Finansinis biudžetas	Prieinama aparatinė įranga	Minimalus saugos lygis, bitais
Programišius	0	PC	53
	< 400 USD	PC / FPGA	58
	0	Piktybiška PĮ, *botnet‘ai*	73
Maža organizacija	10 000 USD	PC / FPGA	64
Vidutinė organizacija	300 000 USD	FPGA / ASIC	68
Didelė organizacija	10 000 000 USD	FPGA / ASIC	78
Vyriausybinė organizacija	300 000 000 USD	ASIC	84

8 lentelė. Saugos lygio ekvivalentas tarp simetrinių ir asimetrinių šifrų

Simetrinio šifro rakto ilgis[2], bitais	RSA[3]	DLOG[4]	EC[5]
48	480	480	96
56	640	640	112
64	816	816	128
80	1248	1248	160
112	2432	2432	224
128	3248	3248	256
160	5312	5312	320
192	7936	7936	384
256	15424	15424	512

1.3. Kriptoalgoritmų energijos sąnaudų analizė

Mokslininkų tandemas Antonio Vincenzo Taddeo ir Alberto Ferrante sprendė problemą [6] kaip realiame laike pasirinkti tinkamą kripto algoritmą atsižvelgiant į saugai keliamus reikalavimus ir sistemos turimus likutinius resursus. Savo darbe autoriai pateikia metodiką, pagal kurią galima realiame laike pasirinkti kriptografinį algoritmą. Minėta metodika remiasi trimis kriptoalgoritmo veiksniais: atminties, laiko bei energijos sąnaudomis. Energijos sąnaudos gali būti tiek iš anksto paskaičiuotos tiek ir nustatomos realiu laiku. Pirmasis variantas tinkamas greitai ir paprastai metodikos integracijai, o antrasis – apsimokymui darbo metu. Išbandydami savo metodiką mokslininkai rėmėsi iš anksto pamatuotomis kriptografinių algoritmų resursų sąnaudomis. Kaip matosi iš resursų sąnaudų lentelės 9, šio tyrimo metu buvo naudojami itin tikslūs matavimo įrankiai ir energijos sąnaudos gautos mikro džiaulių tikslumu.

[2] Bazinis šifro rakto ilgis
[3] Šifravimo algoritmas paremtas skaičių faktorizavimo problema
[4] Šifravimo algoritmai paremti diskrečiojo logaritmo problema
[5] Eliptinių kreivių šifravimo algoritmas

Tyrimo pabaigoje autoriai padarė išvadas:

- Optimalaus kriptografinio algoritmo parinkimas yra tipinis sprendimo priėmimo uždavinys;
- Pasirinktas Analitinis Hierarchinis Procesas kaip pirminis veiksnių svorių įvertinimo algoritmas;
- Suformuluojamas ir išsprendžiamas „kuprinės uždavinys“ – pasirenkamas optimaliausias algoritmas atsižvelgiant į poreikius ir galimybes;
- Tęsiant darbą reikia sukurti sistemos prototipą ir išbandyti pasiūlytą metodą realiomis sąlygomis.

9 lentelė. Kriptografinių algoritmų resursų sąnaudos

Šifras	Rakto ilgis	Bloko dydis	Duomenų atminties	Programos atmintis	Inic. laikas	Laikas	Inic. Energija	Energija
	Bitai	Baitai	Baitai	Baitai	µs	µs/baitas	µJ	µJ/baitas
3DES-CBC	168 + 24	80	10244	142272	31,52	1,08	363,91	12,27
ARC4	128	–	8192	73424	16,27	0,08	248,99	1,7
AES-CBC	256	80	20932	123488	61,61	0,17	1094,75	2,28

Mokslininkai Krzysztof Piotrowski, Peter Langendoerfer, Steffen Peter, tuo metu dirbę IHP Microelectronics, savo darbe [37] analizavo viešojo rakto infrastruktūros (*angl. PKI*) įtaką belaidžių daviklių tinklų (*angl. WSN*) elementams. Viešojo rakto infrastruktūra belaidžių daviklių tinkluose naudojama atskirų daviklių ir šliuzų tarpusavio autentifikacijai. Autoriai tyrimui atlikti panaudojo daviklius su dviejų tipų mikrokontroleriais: ATMEL ATmega128L ir Texas Instruments MSP430F1611. Šie procesoriai yra RISC architektūros, tačiau ATmega128L yra 8 bitų architektūros, o MSP430F1611 – 16 bitų. Tyrime buvo panaudoti:

- MICA2DOT – ATmega128L @ 4MHz;
- MICA2/MICAz – ATmega128L @ 7,37MHz;
- TelosB – MSP430F1611 @ 8MHz

Autorių atlikto energijos sąnaudų tyrimo suvestinė pateikta lentelėje 10. Pasirašymo ir parašo patikrinimo operacijų kiekis, kurį davikliai atliktų maitinami iš dviejų AA beterijų (MICA2/MICAz ir TelosB) bei cr2354 baterijos (MICA2DOT) pateiktas lentelėje 11. Tyrimo metu autoriai papildomai išskyrė ir energijos kiekį reikalingą užtikrinti kliento-serverio funkcionalumą, bei duomenų perdavimą ISM dažnių (433MHz ir 868MHz) siųstuvais.

10 lentelė. Energijos sąnaudos naudojant PKI belaidžių daviklių tinkluose

Kripto sistema	MICA2DOT		MICA2/MICAz		TelosB	
	Pasirašymas	Tikrinimas	Pasirašymas	Tikrinimas	Pasirašymas	Tikrinimas
RSA-1024	304,00 mWs 22,03 s	11,90 mWs 0,86 s	359.87 mWs 12.04 s	14.05 mWs 0.47 s	68.97 mWs 5.66 s	2.70 mWs 0.22 s
ECC-160	22,82 mWs 1,65 s	45,09 mWs 3,27 s	26.96 mWs 0.89 s	53.42 mWs 1.77 s	6.26 mWs 0.52 s	12.41 mWs 1.02 s
RSA-2048	2302,70 mWs 166,86 s	53,70 mWs 3,89 s	2725.66 mWs 91.18 s	63.55 mWs 2.13 s	523.10 mWs 42.89 s	12.20 mWs 1.00 s
ECC-224	61,54 mWs	121,98 mWs	72.85 mWs	144.40 mWs	16.93 mWs	33.55 mWs

	4,46 s	8,84 s	2.41 s	4.78 s	1.39 s	2.76 s

11 lentelė. PKI operacijų kiekis kol bus išnaudota visa energija

Kripto sistema	Daviklis	Pasirašymas	Patikrinimas
RSA-1024	MICA2DOT	12105	310078
	MICA2/MICAz	18757	480427
	TelosB	97867	2500000
ECC-160	MICA2DOT	161586	81542
	MICA2/MICAz	250371	126357
	TelosB	1078275	543916
RSA-2048	MICA2DOT	1598	68547
	MICA2/MICAz	2476	106216
	TelosB	12904	553279
ECC-224	MICA2DOT	59791	30166
	MICA2/MICAz	92656	46745
	TelosB	398701	201192

Atlikę tyrimą autoriai priėjo prie išvadų:

- Energijos sąnaudos e. parašo perdavimui nėra aktualus veiksnys naudojant PKI belaidžių daviklių tinkluose;
- Siekiant sumažinti energijos sąnaudas WSN tinkluose, davikliuose reikia panaudoti specializuotus kriptografinius koprocesorius;
- Energijos sąnaudos PKI įgyvendinimui nėra toks svarbus faktorius kaip buvo tikimasi, ypač jeigu davikliai komunikuoja retai;
- Jeigu aktualus PKI našumas, būtina naudoti specializuotus koprocesorius;
- 16 bitų architektūros mikrokontroleriai žymiai našesni dirbant su asimetriniais kripto algoritmais;
- Belaidžių daviklių atveju iškyla maitinimo šaltinio įtampos kritimo problema.

Dar viena mokslininkų grupė, kurią sudarė Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan ir Niraj K. Jha, savo darbe pavadintame „Saugos protokolų energijos sąnaudų analizė“ [38], atliko išsamią plačiai paplitusių kriptografinių algoritmų energijos sąnaudų analizę. Kaip ir ankstesniuose išanalizuotuose darbuose, autoriai pasitelkė papildomą įrangą kurios pagalba matavo kripto algoritmų sunaudojamą energijos kiekį. Tyrimui jie pasirinko SSL protokolo steką, ir ištyrė kriptografinius algoritmus kurie įeina į šio protokolo sudėtį. Tiriami algoritmai buvo suskaidyti į tris grupes: simetriniai ir asimetriniai kriptografiniai algoritmai ir maišos funkcijos. Papildomai buvo tiriama kokią įtaką energijos sąnaudoms padaro skirtingi kriptoalgoritmų darbo režimai, raktų ilgiai ir kiti konfigūruojami parametrai.

Tyrimui atlikti buvo naudojamas delninis kompiuteris, atsižvelgiant į straipsnio publikavimo datą, tai buvo pasirinktas vienas mobiliausių to meto mikrokompiuteris, kuriam, kaip žinia, energijos sąnaudos yra kritinis veiksnys nepertraukiamo darbo užtikrinimui. Buvo naudojamas Compaq iPAQ H3670 SA-11000 delninis kompiuteris, kurio veikimą užtikrino StrongARM mikroprocesorius, dirbantis 206MHz taktiniu dažniu. Kompiuteryje buvo įdiegta 64MB operatyviosios atmintis (*angl. RAM*) ir 16MB pastoviosios atminties (*angl. ROM*). Prie tinklo įrenginys prisijungdavo naudodamas Cisco Aironet 350 serijos belaidžio tinklo adapterį. Delninukui energiją tiekė vidinė ličio polimerų 950mAh talpos baterija ir jame buvo įdiegta Linux Famillіar operacinė sistema. SSL protokolo veikimui užtikrinti buvo naudojama atvirojo kodo OpenSSL biblioteka. Tiksliem energijos sąnaudų matavimam atlikti buvo naudojamas

laboratorinis maitinimo blokas į kurio maitinimo grandinę nuosekliai įjungta etaloninė varža. Įtampos kritimas, proporcingas delninio kompiuterio naudojamai srovei, šioje varžoje buvo matuojamas naudojantis SCB-68 įvesties/išvesties bloką, kuris per duomenų įvesties plokštę, matavimo rezultatus perduodavo į LabVIEW programinę įrangą.

Atlikę skirtingų kripto algoritmų energijos sąnaudų matavimus autoriai gavo rezultatus, kurie pateikti žemiau. Lentelėje 12 parodytos skirtingų maišos funkcijų energijos sąnaudos mikro džiauliais vienam baitui apdoroti. Devynių blokinių šifrų inicializacijos ir šifravimo/iššifravimo energijos sąnaudos pavaizduotos pav. 8. Šie rezultatai gauti simetriniams šiframs veikiant elektroninės kodų knygos (*angl. ECB*) režimu, t.y. ta pati tekstograma, naudojant tą patį raktą visada bus užšifruojama į tą pačią šifrogramą.

Tyrime naudotų asimetrinių kriptografinių algoritmų rakto generavimo, pasirašymo ir parašo patikrinimo energijos sąnaudos parodytos lentelėje 13. Čia norime atkreipti dėmesį į tai, kad RSA ir DSA algoritmai dirbo su 1024 bitų ilgio raktu, o ECDSA tik su 163 bitų raktu. Atsižvelgus į ECRYPT II ataskaitoje [16] pateiktus raktų ilgio ekvivalentus, kuriuos parodėme lentelėje 8, matosi, kad RSA/DSA 1248 ilgio raktas atitinka 160 bitų ilgio raktą naudojamą ECDSA algoritme. Būtent tai autoriai ir paminėjo savo straipsnyje ir saugos atžvilgiu buvo pasirinkti itin panašaus saugos lygio raktų ilgių atitikmenys. Deja autoriai savo darbe nepaminėjo, kokio ilgio duomenų blokas yra pasirašomas. Taip pat labai įdomūs duomenys yra pateikti lentelėje 14. Čia mokslininkai atliko AES šifro energijos sąnaudų analizę naudojant visus tris galimus rakto ilgius, bei keturis dažniausiai naudojamus blokinių šifrų darbo režimus.

12 lentelė. OpenSSL bibliotekoje naudojamų maišos funkcijų energijos sąnaudos

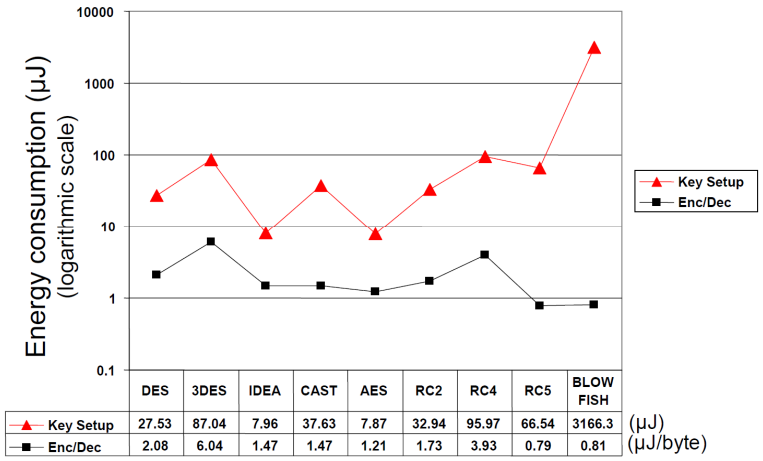
Algoritmas	MD2	MD4	MD5	SHA	SHA1	HMAC
Energijos sąnaudos, (µJ/B)	4,12	0,52	0,59	0,75	0,76	1,16

13 lentelė. OpenSSL bibliotekoje naudojamų skaitmeninio parašo algoritmų energijos sąnaudos

Algoritmas	Rakto ilgis, *bitais*	Rakto generavimas, *mJ*	Pasirašymas, *mJ*	Parašo patikrinimas, *mJ*
RSA	1024	270,13	546,5	15,97
DSA	1024	293,20	313,6	338,02
ECDSA	163	226,65	134,2	196,23

14 lentelė. AES šifro skirtingų raktų ilgių ir darbo režimų energijos sąnaudos

Kripto algoritmas	Inicializacija, *µJ*	ECB, *µJ/B*	CBC, *µJ/B*	CFB, *µJ/B*	OFB, *µJ/B*
AES-128	7,83	1,21	1,62	1,91	1,62
AES-192	7,87	1,42	2,08	2,30	1,83
AES-256	9,92	1,64	2,29	2,31	2,05

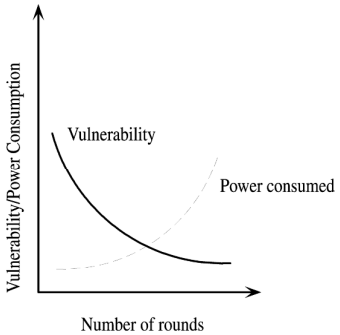


8 pav. OpenSSL bibliotekoje naudojamų simetrinių šifrų energijos sąnaudos

Darbo pabaigoje autoriai daro išvadas:

- Pagal energijos sąnaudas, mažėjančia tvarka algoritmų tipai išsidėsto taip: asimetriniai, simetriniai, maišos funkcijos;
- Asimetrinių algoritmų energijos sąnaudos *labai* priklauso nuo naudojamo rakto ilgio, tačiau to paties negalima pasakyti apie simetrinius kripto algoritmus;
- Energijos sąnaudos, tarp to paties tipo kriptografinių algoritmų varijuoja labai smarkiai;
- Saugos protokolų vykdymas gali būti dinaminis ir priklausyti nuo pradinių sąlygų (pvz. užtikrinti pakankamą saugos lygį optimaliai naudojant energiją).

Mokslininkai R. Chandramouli, S. Bapatla ir K. P. Subbalakshmi darbe pavadintame „Battery Power-Aware Encryption“ [39] pateikia pasiūlymą kaip galima dinamiškai koreguoti energijos sąnaudas naudojant kriptografinius algoritmus siekiant kompromiso tarp energijos sąnaudų ir pageidaujamo saugos lygio užtikrinimo. Grafiškai šį kompromisą autoriai atvaizdavo pav. 9. Grafiko abscisių ašyje atvaizduojamas šifravimo raundų skaičius (kuo daugiau raundų, tuo šifras atsparesnis kriptoanalizei), o ordinačių ašyje – pažeidžiamumas (kuo didesnė reikšmė, tuo mažiau saugus šifras). Ištisine linija pavaizduotas pažeidžiamumo tikimybė (saugos mažėjimas), o punktyrine – energijos sąnaudos. Pažvelgus į kreives matosi, jog jos abi kinta pagal eksponentę. Tai leidžia daryti išvadą, kad nėra tikslinga naudoti „super stiprią“ kriptoalgoritmo modifikaciją, nes galiausiai pažeidžiamumo tikimybė mažės lėtai, o energijos sąnaudos kils labai greitai.

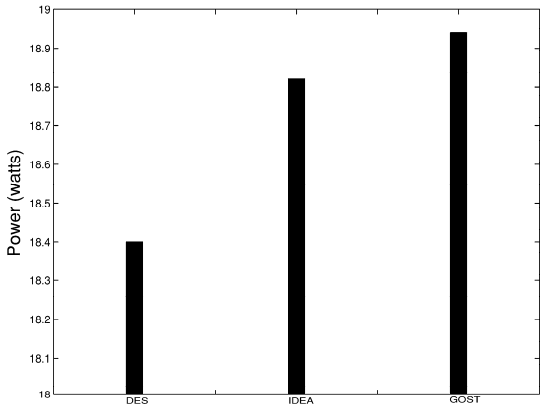


9 pav. Saugos užtikrinimo ir energijos sąnaudų kompromisas

Analizuojamame darbe yra nagrinėjami keturi kriptografiniai algoritmai: DES, IDEA, GOST ir RC4. Autoriai iškelia du tikslus:

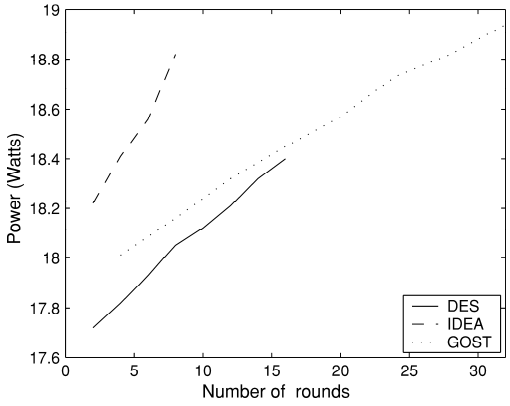
- Pamatuoti ir sumodeliuoti analizuojamų kriptografinių algoritmų energijos sąnaudas;
- Sumažinti pažeidžiamumą (padidinti saugą) neviršijant pasirinktų energijos sąnaudų suvaržymų.

Šiame darbe energijos sąnaudos buvo matuojamos naudojant išorinį maitinimo šaltinį, GPIB duomenų įvesties sąsają ir LabVIEW programinę įrangą. [38] darbo autoriai matavimo platformai buvo pasirinkę delninį kompiuterį, o šiame darbe autoriai pasirinko nešiojamąjį kompiuterį. Visas eksperimentas buvo atliktas naudojantis Sony Vaio nešiojamąjį kompiuterį valdomą 700MHz taktiniu dažniu dirbančio Pentium 3 procesoriaus. Kompiuteryje veikė Red Hat Linux 2.4.8 operacinė sistema. Siekdami eliminuoti energijos sąnaudas būtinas kompiuterio ir OS funkcionavimui, autoriai pirma išmatavo realų energijos suvartojimą, kai kompiuteryje neveikia jokia papildoma programinė įranga išskyrus operacinę sistemą ir jos vidinius modulius. Vėliau atlikus kriptografinių algoritmų energijos sąnaudų matavimus iš jų išminusuojamos „laisvos eigos“ energijos sąnaudos. Tokiu būdu gaunama kiek energijos suvartoja tik kriptoalgoritmai. Gauti pirminiai duomenys parodyti pav. 10. Čia matome nešiojamojo kompiuterio naudojamą galią (įtampos ir srovės sandauga) vatais šifruojant duomenis skirtingais kripto algoritmais. Norėtume pabrėžti, kad savo darbe mokslininkai operuoja ne energijos sąnaudomis, o vidutine vartojama galia. Tai nėra didelė problema, nes galią galima perskaičiuoti į atliktą darbą (sunaudotą energiją).



10 pav. Nešiojamojo kompiuterio naudojama galia šifruojant skirtingais kriptografiniais algoritmais

Tęsdami savo eksperimentą, autoriai išmatuoja nešiojamojo kompiuterio vartojamą galią kriptografiniam algoritmam naudojant skirtingus raundų kiekius. Šie rezultatai pavaizduoti pav. 11. Čia matome, jog GOST šifras energijos sąnaudų kreivė turi mažiausią posvyrį, tai reiškia, kad didinant raundų kiekį energijos sąnaudos didės lėčiausiai. Greičiausiai energijos sąnaudos augs IDEA šifro atveju. Jeigu kriptografinis algoritmas naudos iki 20 raundų, tai energijos požiūriu optimaliausias bus DES šifras.



11 pav. Vartojamos galios priklausomybė nuo šifravimo raundų skaičiaus

Pasinaudodami statistinės regresijos metodu, mokslininkai apskaičiuoja tieses geriausiai atitinkančias pav. 11 atvaizduotas kreives:

$$P_{DES} = 0{,}0486r + 17{,}7335$$
$$P_{IDEA} = 0{,}0975r + 18{,}015$$
$$P_{GOST} = 0{,}03321r + 17{,}90204$$

Gautos tiesės, realias kreives atitinka su standartiniu nuokrypiu mažesniu nei 0,05. RC4 šifro atveju viskas tapo sudėtingiau, nes jo galios priklausomybės nuo rakto ilgio grafikas nėra tiesinis. Gauta kreivės lygtis, kuri realius matavimus atitinka su standartiniu nuokrypiu 0,0797:

$$P(k) = 15{,}85e^{-3{,}311/k}$$

Šiuo atveju, autoriai panaudojo matavimus atliktus su k = 32, 64, 128, 192 ir 256.

Toliau darbe yra analizuojama, kaip matematiškai aprašyti kriptografinio algoritmo pažeidžiamumą, bei siūlomi sprendimai kaip išspręsti „kuprinės uždavinį“. Šis uždavinys buvo minimas ir [6] darbe. Kadangi šie sprendimai, energijos sąnaudų prognozavimas ir optimalaus kriptografinio algoritmo parinkimas su mūsų darbu nėra tiesiogiai susiję, tai šią dalį mūsų analizėje praleisime ir pereisime prie darbe minimų išvadų.

- Atlikus realių energijos sąnaudų matavimus nustatyta, kad:
    - DES, IDEA ir GOST kriptografinių algoritmų energijos sąnaudos priklausomai nuo naudojamo rakto ilgio kinta tiesiškai;
    - Iš tirtų algoritmų, tik RC4 energijos sąnaudų priklausomybė nuo rakto ilgio kito ne pagal tiesinį dėsnį;
    - GOST kripto algoritmas pasižymėjo mažiausiu energijos sąnaudų kiekio padidėjimu didėjant raundų skaičiui.

1.4. Išvados

- Duomenų saugos užtikrinimui naudojami įvairūs simetriniai, asimetriniai šifravimo algoritmai ir maišos funkcijos;
- Išanalizuotuose mokslinėse publikacijose, daugelyje atvejų, kriptografinių algoritmų energijos sąnaudos įvertinamos naudojant tikslią laboratorinę matavimo įrangą pateikiant rezultatus mikro džiauliais;
- Realiose sąlygose matavimai naudojant laboratorinę matavimo įrangą gali būti naudojami, tačiau jie neįvertina sistemų aparatūrinės ir programinės įrangos veiklų kurios įtakoja energijos sąnaudas;
- Informacijos saugos lygiai priklauso nuo naudojamų šifravimo raktų ilgio, todėl šis parametras yra svarbus tiriant energijos sąnaudas;
- Sėkmingam energijos sąnaudų tyrimui reikalingas baterijos elgsenos modelio sudarymas ir vartojamos energijos matavimo programinės įrangos sukūrimas veikiančios taikomųjų programų lygmenyje.

2. ENERGIJOS SĄNAUDŲ MODELIAVIMAS

Nepakankamai ilgai veikiančios baterijos buvo ir išlieka problema mobiliųjų įrenginių vartotojams. Vis dar jaučiamas didelis atotrūkis tarp energijos resursų poreikio ir jų kiekio kurį šiuolaikinės baterijos gali sukaupti (ribotas energijos tankis). Šiuo metu vidutinis aktyviai naudojamo mobiliojo įrenginio darbo laikas naudojantis tik baterijos tiekiama energija yra mažiau nei dvi dienos. Baterijos energijos tankio problemą išspręsti nėra taip paprasta, nors mokslininkai deda didžiules pastangas, o įmonės investuoja dideles lėšas. Į šią problemą galima pažvelgti ir iš kitos pusės – numatyti mobilaus įrenginio baterijos iškrovimo charakteristikas taikomųjų programų lygmenyje atsižvelgiant į naudojamą programinę įrangą, įrenginio darbo režimus ir energijos valdymo profilius. Pasiekus šį tikslą vartotojas galėtų pats nuspręsti bei pasirinkti kaip likutinės energijos požiūriu efektyviausiai atlikti reikiamas užduotis. Toks prognozavimas galimas tik tuo atveju kai turimas pakankamai tikslus baterijos elgsenos modelis atsižvelgiantis į vidinius (elektrocheminius) ir išorinius (procesoriaus apkrova, atminties naudojimas, vaizdų generavimas ir pan.) faktorius kurie įtakoja baterijos įkrovos lygį (*angl. State-of-Charge, SoC*).

Pagrindiniai faktoriai įtakojantys baterijos gyvavimo laiką yra baterijos talpa ir iškrovimo greitis. Šis greitis įtakojamas trijų netiesinių veiksnių, iš kurių du yra apibrėžiami baterijos elektrocheminėmis savybėmis. Apkrovos-talpos efektas (*angl. rate-capacity effect*) stebimas, kai baterija iškraunama pastoviai. Iškrovimas didele srove nulemia, jog baterija atiduos mažiau energijos kol pilnai išsikraus, negu atiduotų iškraunama maža srove. Atsistatymo efektas (*angl. recovery effect*) stebimas, kaip baterijos likutinės energijos atsistatymas bateriją iškraunant maža srove arba jos visai neiškraunant ilgesnį laiko tarpą. Trečiasis netiesinis procesas lemiantis baterijos įkrovos pasikeitimus yra programinis energijos sąnaudų planavimas taikomųjų programų arba operacinės sistemos lygmenyse. Minėtas procesas apima, pavyzdžiui, procesoriaus taktinio dažnio ir/arba maitinimo įtampos dinaminį keitimą, periferinių įrenginių atjungimą kurie nėra naudojami, ekrano ryškumo keitimą ir pan. Šių trijų veiksnių visuma laike, atsižvelgus į vartotojo elgseną ir atliekamas operacijas, nulemia kaip greitai baterija bus iškrauta. Kai kuriais atvejais lemiamu faktoriumi baterijos faktiniai talpai gali būti ne vidutinė iškrovimo srovė, bet ribinė. Tačiau visais atvejais faktinė baterijos talpa priklausys nuo to kaip baterija yra iškraunama, nes elektrocheminiai procesai, vykstantys baterijoje, trunka tam tikrą laiką ir jie negali atkartoti momentinių energijos pareikalavimų.

2.1. Baterijos elgsenos modelis

Netiesinę bateriją aprašančią sistemą galima aprašyti lygčių sistema:

$$x_{k+1} = f(x_k, u_k) + w_k \tag{15}$$

$$y_k = g(x_k, u_k) + v_k \tag{16}$$

čia x yra sistemos būsena, u – sistemos įvestis, w – nepamatuotas vidinių procesų triukšmas, kuris įtakoja sistemos būseną, y sistemos išvesties reikšmė, v matavimo triukšmas ir k – diskretus laiko momentas kuriame aprašoma sistema.

Baterijos būsena gali būti aprašoma diferencialine vėlinimo lygtimi:

$$x(t) = f\big(x(t), x(t-\tau)\big) \tag{17}$$

čia τ yra proceso vėlinimas.

Kurdami baterijos elgsenos modelį, mes remiamės tokiomis prielaidomis:

1. Baterijos būseną nulemia du faktoriai: procesoriaus apkrova ir fizinės atminties naudojimas;
2. Dėl lėtų elektrocheminių procesų vykstančių baterijoje baterijos įkrovos lygis pakinta vėliau nei pasikeičia procesoriaus arba atminties sunaudojimas.

Mes formuluojame savo baterijos įkrovos pokyčio vėlinimo modelį kaip:

$$c(t) = f\big(c(t), c(t-\tau_c), l(t-\tau_l), m_a(t-\tau_a), m_r(t-\tau_r)\big) \tag{18}$$

čia $c(t)$ yra baterijos įkrovos lygis (SoC), $l(t)$ – procesoriaus apkrovos lygis, $m_a(t)$ – fizinės atminties išskyrimas, $m_r(t)$ – fizinės atminties atlaisvinimas, τ_c – baterijos iškrovimo vėlinimas, τ_l – procesoriaus apkrovos vėlinimas, τ_a – atminties išskyrimo vėlinimas, τ_r – atminties atlaisvinimo vėlinimas.

Baterijos iškrovimas yra tolydus procesas, kai tuo tarpu energiją naudojantys procesai (procesoriaus vykdomos operacijos, atminties išskyrimo ir atlaisvinimo operacijos) yra diskretūs. Baterijos įkrovos lygis taip pat matuojamas diskrečiais laiko tarpais. Tuo pačiu procesoriaus apkrova galima laikyti momentine, o operacijas susijusias su atminties išskyrimu ir atlaisvinimu galima sužinoti tik per tam tikrą laiko tarpą. Norint šias reikšmes atvaizduoti kaip tolydines reikia įvesti filtravimą.

Tegul l_t bus procesoriaus apkrovimo lygis procentais laiko momentu t, m_t – naudojamos atminties kiekis laiko momentu t, c_t – baterijos įkrovos lygis procentais tuo pačiu laiko momentu t ir funkcija $h(x)$ bus žingsnio funkcija lygi:

$$h(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases} \tag{19}$$

Fizinės atminties išskyrimai laiko intervale $(t_0, t_0 + \Delta t)$ apskaičiuojami taip:

$$m_a(t_0, \Delta t) = \sum_{t=t_0}^{t_0+\Delta t} h(m_t - m_{t-1}) \tag{20}$$

Fizinės atminties atlaisvinimai laiko intervale $(t_0, t_0 + \Delta t)$ apskaičiuojami taip:

$$m_r(t_0, \Delta t) = \sum_{t=t_0}^{t_0+\Delta t} h(m_{t-1} - m_t) \tag{21}$$

Tegul,

$$f_s(X, t_0, w) = \frac{1}{w} \sum_{t=t_0}^{t_0+w-1} x_t \tag{22}$$

būna filtravimo funkcija (mes naudosime slenkantį vidurkį, MVA) laiko eilutėje $x_t \in X$, kur w bus slenkančio vidurkio lango dydis. Vidutinė procesoriaus apkrova lange w_l bus:

$$\overline{l(t)} = f_s(l_t, t_0, w_l) \tag{23}$$

Vidutinis išskiriamos fizinės atminties kiekis lange w_a bus:

$$\overline{m_a(t)} = f_s(m_a, t_0, w_a) \quad (24)$$

Vidutinis atlaisvinamos fizinės atminties kiekis lange w_r bus:

$$\overline{m_r(t)} = f_s(m_r, t_0, w_r) \quad (25)$$

Vidutinė baterijos įkrova lange w_c bus:

$$\overline{c(t)} = f_s(c_t, t_0, w_c) \quad (26)$$

Vėlinančio modelio pritaikymas energijos etaloniniam matavimui. Efektyvus energijos naudojimas yra kritinis faktorius kuriant baterija maitinamus įrenginius. Dažniausiai kompiuterių etaloniniai testavimai *(angl. benchmarks*) atliekami, pasitelkiant specializuotą programinę įrangą, kuri įvertina kompiuterio našumą atlikdama griežtas ir vienodas operacijas. Įvertinimas (metrika) paprastai pateikiamas skaitine reikšme, kuri yra specifinė naudojamai etaloninio testavimo programinei įrangai. Skirtinguose kompiuteriuose atlikus vienodus etaloninius testus naudojant tą pačią programinę įranga, juos vėliau galima palyginti našumo požiūriu.

Šiame darbe mes siūlome savo metriką, paremtą mūsų pasiūlytu baterijos iškrovimo vėlinimo modeliu. Mažesnė mūsų metrikos reikšmė reiškia geresnius rezultatus energijos suvartojimo požiūriu. Šią metriką galima aprašyti lygtimi:

$$P_b = \frac{s_c}{s_{perf}} \left(c(t_{start}) - \frac{\sum_{t=t_{start}}^{t_{end}} c(t)}{\Delta t(t_{end} - t_{start})} \right) \quad (27)$$

čia t_{start} yra etaloninio testo pradžios laikas, t_{end} – testo pabaigos laikas, Δt – parametrų atskaitų fiksavimo intervalas, s_{perf} – etaloninio testo rezultato skaitinė reikšmė ir s_c yra konstanta leidžianti rezultatą atvaizduoti į norimą reikšmių intervalą. Jeigu $c(t)$ yra apskaičiuojama procentais, tai s_c reikėtų naudoti lygų 1000.

2.2. Programinės priemonės

Programinės įrangos reikalingos eksperimentui atlikti kūrimui buvo pasirinkta:

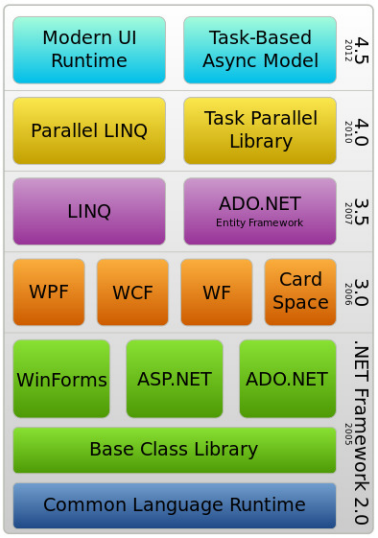
- Microsoft .NET Framework 3.5 karkasas
- BouncyCastle 1.7 kriptografinių algoritmų biblioteka
- Programavimo kalba C#
- Microsoft Visual Studio 2012 integruota programavimo aplinka

Programavimo kalba C# sukurta ir vystoma Microsoft korporacijos. Ši kalba yra labai panaši į C++ ir Java. Pirma kartą ji buvo išleista 2001 metais, kartu su Microsoft .NET Framework karkasu. Šios kalbos kūrimui vadovavo Anders Hejlsberg. C# programavimo kalba buvo kuriama kaip paprasta, moderni, bendro naudojimo objektiškai orientuota programavimo kalba. Kaip ir Java, C# yra pilnai objektiškai orientuota programavimo kalba, t.y. kiekvienas tipas sukonstruojamas į objektą. C# kalba parašytas išeities kodas yra kompiliuojamas į tarpinę kalbą, vadinamą *MSIL (Microsoft Intermediate Language)*, kuri labai panaši į x86 architektūros procesorių asemblerio kalbą. Vėliau, vykdymo metu, reikiami metodai yra kompiliuojami į procesoriaus palaikomą mašininį kodą ir vykdomi pagrindinio procesoriaus. Šis procesas vadinamas *Just-In-Time* kompiliavimas. .NET karkasas turi integruotą modulį, *NGEN*, kuris

foniniame režime kompiliuoja visus sistemoje registruotus .NET modulius ir programas, kad išvengti pastovaus *JIT* kompiliavimo, tokiu būdu padidinamas programų našumas ir pagreitėja paleidimas. *JIT* kompiliavimas suteikia privalumą programą leidžiant skirtingose architektūrose. Kaip pavyzdys gali būti kompiliuojant išnaudojami papildomo procesoriaus komandų rinkiniai, tokie kaip SSE, SSE2, 3DNow! ir pan. .NET Framework karkasas leidžia vykdyti ir nevaldomą kodą (*angl. unsafe code*, tiesiogines procesoriaus komandas), todėl ir C# programavimo kalbos metodais galima paprastai iškviesti bet kokią operacinės sistemos funkciją.

.NET Framework karkasas yra Microsoft korporacijos vystomas programinės įrangos kūrimo ir vykdymo karkasas kuris veikia Windows operacinėje sistemoje. Egzistuoja ir jo atviro kodo analogas – MONO karkasas, tačiau jis savo funkcionalumu ir galimybėmis atsilieka 2-3 versijom nuo naujausios Microsoft leidžiamos versijos. Pagrindiniai .NET karkaso privalumai yra du:

- Milžiniška klasių biblioteka (*angl. class library*);
- Suderinamumas su skirtingomis programavimo kalbomis.



12 pav. .NET karkaso versijos ir funkcionalumas

.NET karkasui parašytos programos vykdomos valdomoje aplinkoje, o ne tiesiogiai procesoriuje. Ši valdoma aplinka vadinama Bendrinė Kalbų Vykdymo Aplinka (*angl. Common Language Runtime, CLR*); ji teikia tokias paslaugas kaip sauga, atminties valdymas ir klaidų valdymas. Minėta klasių biblioteka kartu su vykdančiąja aplinka ir sudaro .NET karkasą. Pagrindinius ir stambiausius klasių bibliotekos funkcionalumus, išskirstytus pagal karkaso versijas, pateikėme pav. 12. Bazinė .NET klasių biblioteka savyje turi kodą skirtą dirbti su:

- Vartotojo sąsajomis;
- Duomenų bazių valdymo sistemomis;
- Kriptografija;
- Žiniatinklio taikomosiomis programomis;
- Skaitiniais algoritmais;
- Tinklinėmis technologijomis.

Microsoft teigimu .NET karkasas turėtų būti naudojamas kuriant visas naujas taikomąsias programas. Žinoma tai nereiškia, kad pavyzdžiui tvarkyklių kūrimui irgi reikėtų

naudoti ši karkasą, tai nepavyks netgi jeigu ir norėtumėme, nes tvarkyklės veikia kitame procesoriaus lygmenyje nei operacinės sistemos dalis kurį vykdo valdomą kodą (*angl. managed code*).

Microsoft Visual Studio yra integruota programavimo aplinka skirta programuotojams, darbui su .NET Framework karkasu, tame tarpe ir programų kūrimui C# kalba. Žinoma ši aplinka neapsiriboja tik .NET karkasu, jos pagalba galima programuoti ir C++, Visual Basic, o atsisiuntus atitinkamų plėtinių – daugeliu programavimo kalbų. Pilnas Visual Studio paketas yra pakankamai brangus produktas, todėl pavieniams programuotojams ir smulkioms įmonėms yra siūloma nemokama, apribotų galimybių versija – *Express Edition*. Nemokama versija gali būti laisvai naudojama tiek atviro kodo, nemokamai programinei įrangai kurti tiek ir komercinių produktų kūrimui.

BouncyCastle yra atvirojo kodo kriptografinių algoritmų ir protokolų biblioteka. Ją galima naudoti .NET Framework karkase veikiančiomis ir JAVA programavimo kalbomis. Ši biblioteka palaiko labai platų spektrą kriptografinių algoritmų ir protokolų. Yra įgyvendinti visi plačiausiai naudojami simetriniai ir asimetriniai šifrai, saugos protokolai ir, ką labai vertina programuotojai, visa tai įgyvendinta naudojant bendrines sąsajas, o tai labai palengvina ir pagreitina programavimą. BouncyCaste biblioteka buvo kuriama JAVA programavimo kalbos pagrindu, tačiau vėliau buvo papildomai kuriama ir palaikoma C# versija. Iki šiol bibliotekos pagrindu yra JAVA dalis, o C# dalis truputį atsilieka vystyme, tačiau tai nedaro didelės įtakos galimybėms BouncyCastle naudoti kuriant programas.NET karkaso pagrindu. Egzistuoja ir dar viena BouncyCastle atšaka – SpongyCastle. Kadangi Android operacinėje sistemoje, dėl vardų erdvių (*angl. namespaces*) konflikto neįmanoma naudoti BouncyCastle JAVA versijos, tai vystytojai nusprendę sukurti atskirą versiją kurioje vienintelis pakeitimas yra pervadintos vardų erdvės.

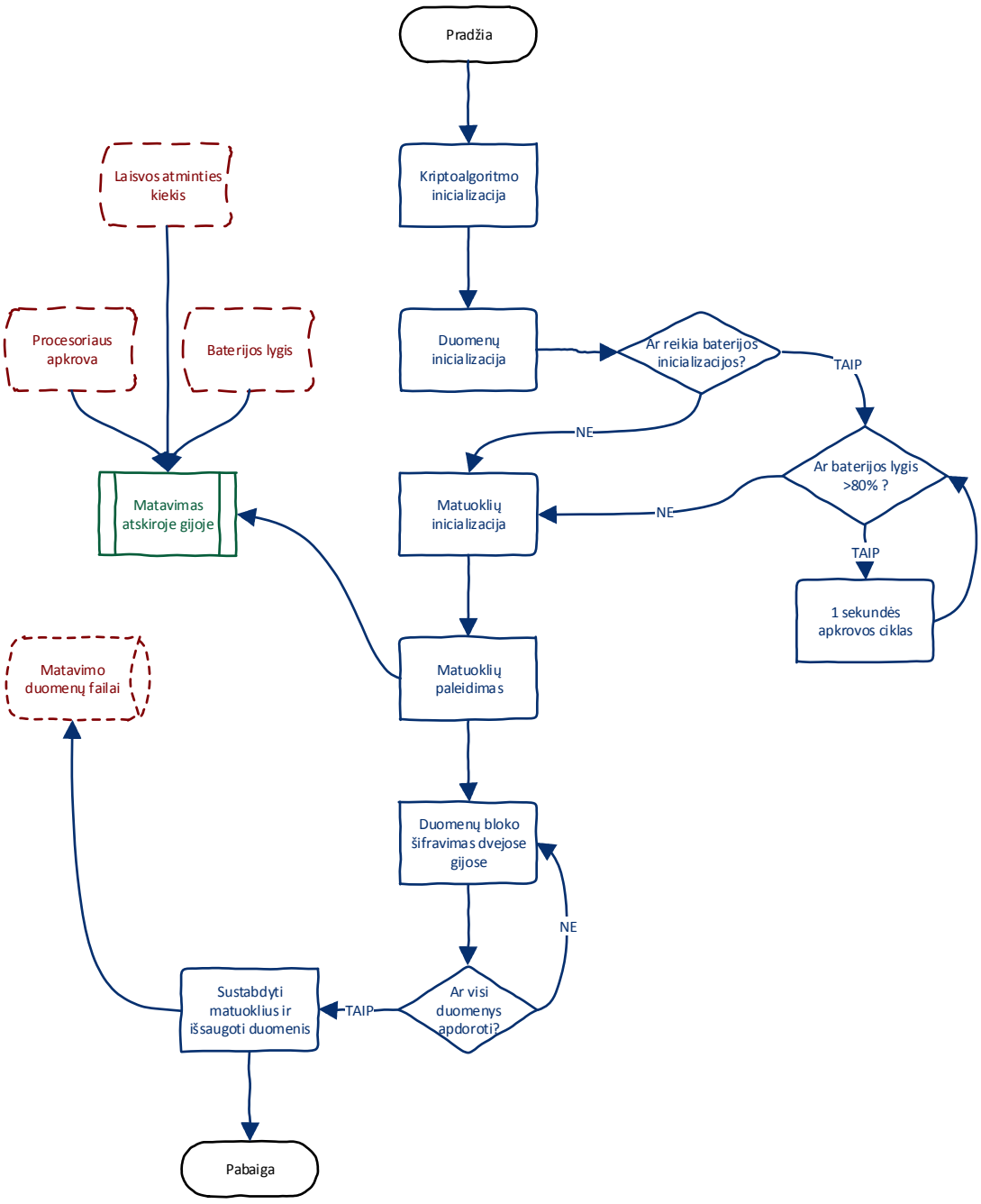
2.3. Programinė įranga baterijos energijos sąnaudų įvertinimui

Kriptografinių algoritmų energijos sąnaudų tyrimui atlikti buvo suprojektuota ir sukurta speciali programinė įranga. Programinei įrangai kurti buvo pasirinktas Microsoft .NET Framework 3.5 karkasas ir C# programavimo kalba. Programinės įrangos kūrimui ir derinimui buvo naudota Microsoft Visual Studio 2012 integruota programavimo aplinka (*angl. IDE*). Microsoft technologijos buvo pasirinktos todėl, kad jos dominuoja vartotojiškuose kompiuteriuose ir užima didelę dalį korporatyvinės rinkos. Eksperimentinis tyrimas buvo atliekamas naudojantis senesne, tačiau vis dar labai plačiai paplitusia, operacine sistema Microsoft Windows XP. Šioje OS veikia tik dalis visų .NET Framework karkaso palaikomų kriptografinių algoritmų – neveikia *Cryptography API: Next Generation* (*trump. CNG*) API naudojantys algoritmai. *CNG* palaikymas Windows operacinėse sistemose atsirado tik nuo Windows Vista versijos. Dėl šio apribojimo dalis eksperimente naudotų kriptografinių algoritmų buvo iš Microsoft .NET karkaso, o dalis iš BouncyCastle 1.7 kriptografinės bibliotekos parašytos C# programavimo kalba (žr. 48 psl. esančią lentelę 15).

Eksperimentui atlikti sukurtos programinės įrangos veikimo algoritmas pavaizduotas pav. 13. Iš algoritmo schemos matosi, kad programinės įrangos darbo ciklą sudaro tokie etapai:

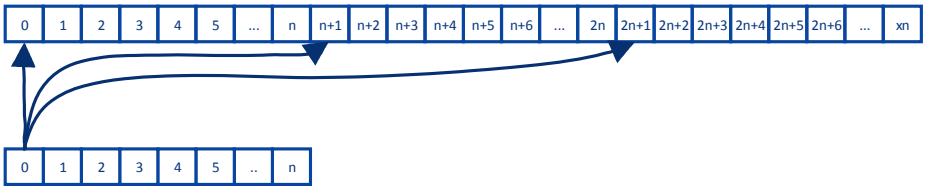
- Inicializacija:
    - Kriptografinio algoritmo inicializavimas;
    - Duomenų apdorojimui paruošimas ;

- Jeigu reikia, vykdoma baterijos inicializacija (pilnai apkrovus procesorių laukiama kol baterija išsikraus iki 80%);
- Inicializuojami ir startuojami kompiuterio parametrų matavimo komponentai – matuokliai;
- Vykdomas duomenų apdorojimas (šifravimas) dvejose gijose, tol kol bus apdoroti visi numatyti duomenys;
- Sustabdomi matuokliai ir išsaugomi jų sukaupti duomenys.



13 pav. Eksperimente naudotos programinės įrangos veikimo algoritmas

Duomenys apdorojimui paruošiami tokiais etapais: pirmiausia nuskaitomas etaloninis duomenų failas, tada suformuojamas konfigūracijoje nurodyto ilgio baitų masyvas ir jis ciklišk ai užpildomas etaloniniais duomenimis (žr. pav. 14).



14 pav. Ciklinis masyvo pildymas etaloniniais duomenimis

Duomenų apdorojimo metu, kiekvieną sekundę yra skaitomi ir atmintyje kaupiami tokie duomenys:

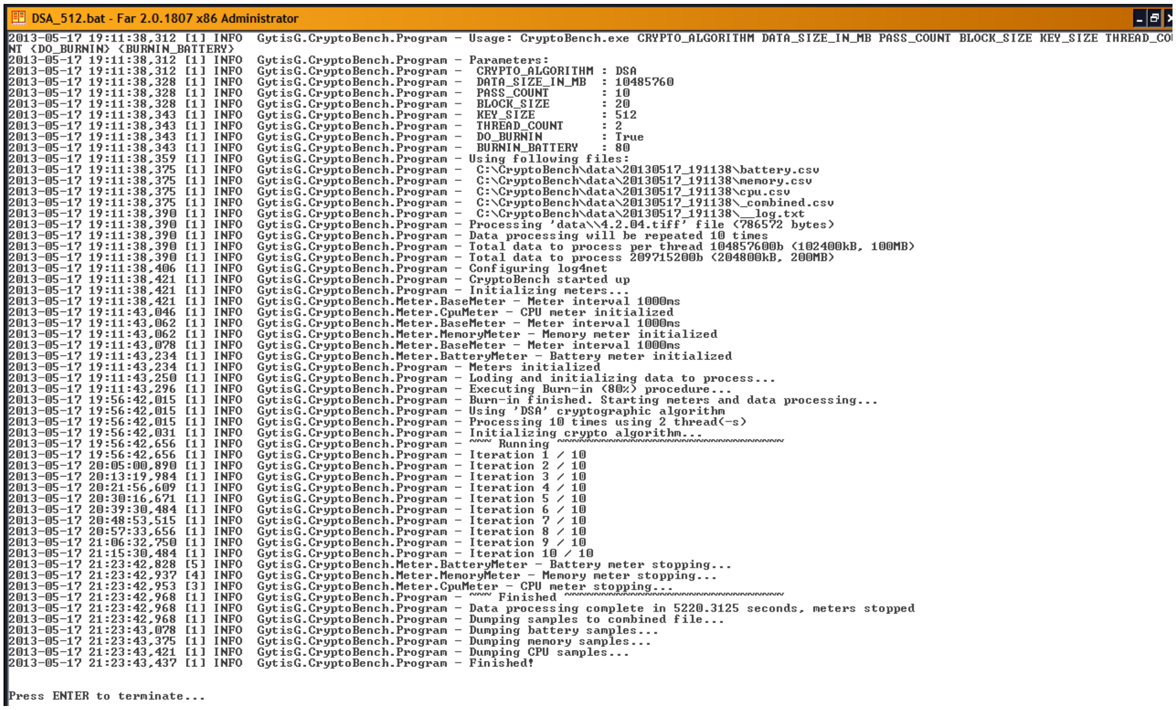
- Pagrindinio procesoriaus apkrova procentais;
- Baterijos įkrovos likutinė vertė procentais;
- Laisvos fizinės atminties kiekis baitais.

Eksperimento ciklui pasibaigus sukuriamas įvykių žurnalo failas, kuriame išsaugomi konfigūracijos duomenys, pagrindiniai programinės įrangos atlikti žingsniai, ir galutinė eksperimento statistika. Visa eksperimento metu sukaupta informacija vienu metu įrašoma į CSV formato failą. Šis failas vėliau tiesiogiai užkraunamas į Microsoft Excel tolesniam duomenų apdorojimui. CSV failo, atidaryto su Excel programa, ištraukos pavyzdys (ECDSA 256 bitų ilgio rakto šifravimo duomenys) pateiktas pav. 15.

	A	B	C	D	E	F	G	H
1				ECDSA-256				
2	Rel.Time	Battery	Power	MemoryTotal	MemoryFree	CPU		
3	0	80	FALSE	1600241664	945852416	99		
4	1	80	FALSE	1600241664	947466240	98		
5	2	80	FALSE	1600241664	948924416	100		
6	3	80	FALSE	1600241664	949551104	97		
7	4	80	FALSE	1600241664	949395456	98		
8	5	80	FALSE	1600241664	949403648	97		
9	6	80	FALSE	1600241664	949338112	98		
10	7	80	FALSE	1600241664	949272576	97		
11	8	80	FALSE	1600241664	948547584	99		
12	9	80	FALSE	1600241664	949272576	97		
13	10	80	FALSE	1600241664	948801536	97		
14	11	80	FALSE	1600241664	948547584	98		
15	12	80	FALSE	1600241664	954724352	98		
16	13	80	FALSE	1600241664	954802176	95		
17	14	80	FALSE	1600241664	954810368	96		
18	15	80	FALSE	1600241664	954810368	96		
19	16	80	FALSE	1600241664	954806272	97		
20	17	80	FALSE	1600241664	953774080	97		
21	18	80	FALSE	1600241664	953868288	96		
246	5243	35	FALSE	1600241664	951300096	94		
247	5244	35	FALSE	1600241664	951300096	94		
248	5245	35	FALSE	1600241664	951267328	96		
249	5246	35	FALSE	1600241664	951255040	97		
250	5247	35	FALSE	1600241664	951279616	97		
251	5248	35	FALSE	1600241664	951279616	97		
252	5249	35	FALSE	1600241664	951279616	96		
253	5250	35	FALSE	1600241664	951271424	95		
254	5251	35	FALSE	1600241664	951312384	97		
255	5252	35	FALSE	1600241664	951304192	93		
256	5253	35	FALSE	1600241664	951623680	98		
257	5254	35	FALSE	1600241664	951525376	95		
258	5255	35	FALSE	1600241664	951566336	96		
259	5256	35	FALSE	1600241664	951549952	97		
260	5257	35	FALSE	1600241664	951521280	99		
261	5258	35	FALSE	1600241664	951545856	80		
262	5259	35	FALSE	1600241664	951545856	50		
263	5260	35	FALSE	1600241664	951545856	50		
264	5261	35	FALSE	1600241664	951541760	50		
265	5262	35	FALSE	1600241664	951517184	50		
266	5263	35	FALSE	1600241664	951549952	50		
267	5264	35	FALSE	1600241664	951541760	50		
268	5265	35	FALSE	1600241664	951541760	50		
269								

15 pav. CSV duomenų failas atidarytas su Excel programa

Programinės įrangos, baigusios DSA-512 algoritmo matavimus, ekrano kopija pavaizduota pav. 16. Visa informacija kuri yra matoma ekrane yra išsaugoma, jau minėtame, programinės įrangos veiklą atspindinčiame įvykių žurnalo faile.



```
DSA_512.bat - Far 2.0.1807 x86 Administrator
2013-05-17 19:11:38,312 [1] INFO  GytisG.CryptoBench.Program - Usage: CryptoBench.exe CRYPTO_ALGORITHM DATA_SIZE_IN_MB PASS_COUNT BLOCK_SIZE KEY_SIZE THREAD_COU
NT <DO_BURNIN> <BURNIN_BATTERY>
2013-05-17 19:11:38,312 [1] INFO  GytisG.CryptoBench.Program - Parameters:
2013-05-17 19:11:38,312 [1] INFO  GytisG.CryptoBench.Program -  CRYPTO_ALGORITHM : DSA
2013-05-17 19:11:38,328 [1] INFO  GytisG.CryptoBench.Program -  DATA_SIZE_IN_MB  : 10485760
2013-05-17 19:11:38,328 [1] INFO  GytisG.CryptoBench.Program -  PASS_COUNT       : 10
2013-05-17 19:11:38,328 [1] INFO  GytisG.CryptoBench.Program -  BLOCK_SIZE       : 20
2013-05-17 19:11:38,343 [1] INFO  GytisG.CryptoBench.Program -  KEY_SIZE         : 512
2013-05-17 19:11:38,343 [1] INFO  GytisG.CryptoBench.Program -  THREAD_COUNT     : 2
2013-05-17 19:11:38,343 [1] INFO  GytisG.CryptoBench.Program -  DO_BURNIN        : True
2013-05-17 19:11:38,343 [1] INFO  GytisG.CryptoBench.Program -  BURNIN_BATTERY   : 80
2013-05-17 19:11:38,359 [1] INFO  GytisG.CryptoBench.Program - Using following files:
2013-05-17 19:11:38,375 [1] INFO  GytisG.CryptoBench.Program -  C:\CryptoBench\data\20130517_191138\battery.csv
2013-05-17 19:11:38,375 [1] INFO  GytisG.CryptoBench.Program -  C:\CryptoBench\data\20130517_191138\memory.csv
2013-05-17 19:11:38,375 [1] INFO  GytisG.CryptoBench.Program -  C:\CryptoBench\data\20130517_191138\cpu.csv
2013-05-17 19:11:38,375 [1] INFO  GytisG.CryptoBench.Program -  C:\CryptoBench\data\20130517_191138\_combined.csv
2013-05-17 19:11:38,390 [1] INFO  GytisG.CryptoBench.Program -  C:\CryptoBench\data\20130517_191138\__log.txt
2013-05-17 19:11:38,390 [1] INFO  GytisG.CryptoBench.Program - Processing 'data\\4.2.04.tiff' file (786572 bytes)
2013-05-17 19:11:38,390 [1] INFO  GytisG.CryptoBench.Program - Data processing will be repeated 10 times
2013-05-17 19:11:38,390 [1] INFO  GytisG.CryptoBench.Program - Total data to process per thread 104857600b (102400kB, 100MB)
2013-05-17 19:11:38,390 [1] INFO  GytisG.CryptoBench.Program - Total data to process 209715200b (204800kB, 200MB)
2013-05-17 19:11:38,406 [1] INFO  GytisG.CryptoBench.Program - Configuring log4net
2013-05-17 19:11:38,421 [1] INFO  GytisG.CryptoBench.Program - CryptoBench started up
2013-05-17 19:11:38,421 [1] INFO  GytisG.CryptoBench.Program - Initializing meters...
2013-05-17 19:11:38,421 [1] INFO  GytisG.CryptoBench.Meter.BaseMeter - Meter interval 1000ms
2013-05-17 19:11:43,046 [1] INFO  GytisG.CryptoBench.Meter.CpuMeter - CPU meter initialized
2013-05-17 19:11:43,062 [1] INFO  GytisG.CryptoBench.Meter.BaseMeter - Meter interval 1000ms
2013-05-17 19:11:43,062 [1] INFO  GytisG.CryptoBench.Meter.MemoryMeter - Memory meter initialized
2013-05-17 19:11:43,078 [1] INFO  GytisG.CryptoBench.Meter.BaseMeter - Meter interval 1000ms
2013-05-17 19:11:43,234 [1] INFO  GytisG.CryptoBench.Meter.BatteryMeter - Battery meter initialized
2013-05-17 19:11:43,234 [1] INFO  GytisG.CryptoBench.Program - Meters initialized
2013-05-17 19:11:43,250 [1] INFO  GytisG.CryptoBench.Program - Loding and initializing data to process...
2013-05-17 19:11:43,296 [1] INFO  GytisG.CryptoBench.Program - Executing Burn-in (80%) procedure...
2013-05-17 19:56:42,015 [1] INFO  GytisG.CryptoBench.Program - Burn-in finished. Starting meters and data processing...
2013-05-17 19:56:42,015 [1] INFO  GytisG.CryptoBench.Program - Using 'DSA' cryptographic algorithm
2013-05-17 19:56:42,015 [1] INFO  GytisG.CryptoBench.Program - Processing 10 times using 2 thread(-s)
2013-05-17 19:56:42,031 [1] INFO  GytisG.CryptoBench.Program - Initializing crypto algorithm...
2013-05-17 19:56:42,656 [1] INFO  GytisG.CryptoBench.Program - ~~~ Running ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
2013-05-17 19:56:42,656 [1] INFO  GytisG.CryptoBench.Program - Iteration 1 / 10
2013-05-17 20:05:00,890 [1] INFO  GytisG.CryptoBench.Program - Iteration 2 / 10
2013-05-17 20:13:19,984 [1] INFO  GytisG.CryptoBench.Program - Iteration 3 / 10
2013-05-17 20:21:56,609 [1] INFO  GytisG.CryptoBench.Program - Iteration 4 / 10
2013-05-17 20:30:16,671 [1] INFO  GytisG.CryptoBench.Program - Iteration 5 / 10
2013-05-17 20:39:30,484 [1] INFO  GytisG.CryptoBench.Program - Iteration 6 / 10
2013-05-17 20:48:53,515 [1] INFO  GytisG.CryptoBench.Program - Iteration 7 / 10
2013-05-17 20:57:33,656 [1] INFO  GytisG.CryptoBench.Program - Iteration 8 / 10
2013-05-17 21:06:32,750 [1] INFO  GytisG.CryptoBench.Program - Iteration 9 / 10
2013-05-17 21:15:30,484 [1] INFO  GytisG.CryptoBench.Program - Iteration 10 / 10
2013-05-17 21:23:42,828 [5] INFO  GytisG.CryptoBench.Meter.BatteryMeter - Battery meter stopping...
2013-05-17 21:23:42,937 [4] INFO  GytisG.CryptoBench.Meter.MemoryMeter - Memory meter stopping...
2013-05-17 21:23:42,953 [3] INFO  GytisG.CryptoBench.Meter.CpuMeter - CPU meter stopping...
2013-05-17 21:23:42,968 [1] INFO  GytisG.CryptoBench.Program - ~~~ Finished ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
2013-05-17 21:23:42,968 [1] INFO  GytisG.CryptoBench.Program - Data processing complete in 5220.3125 seconds, meters stopped
2013-05-17 21:23:42,968 [1] INFO  GytisG.CryptoBench.Program - Dumping samples to combined file...
2013-05-17 21:23:43,078 [1] INFO  GytisG.CryptoBench.Program - Dumping battery samples...
2013-05-17 21:23:43,375 [1] INFO  GytisG.CryptoBench.Program - Dumping memory samples...
2013-05-17 21:23:43,421 [1] INFO  GytisG.CryptoBench.Program - Dumping CPU samples...
2013-05-17 21:23:43,437 [1] INFO  GytisG.CryptoBench.Program - Finished!

Press ENTER to terminate...
```

16 pav. Programinės įrangos darbo rezultatai

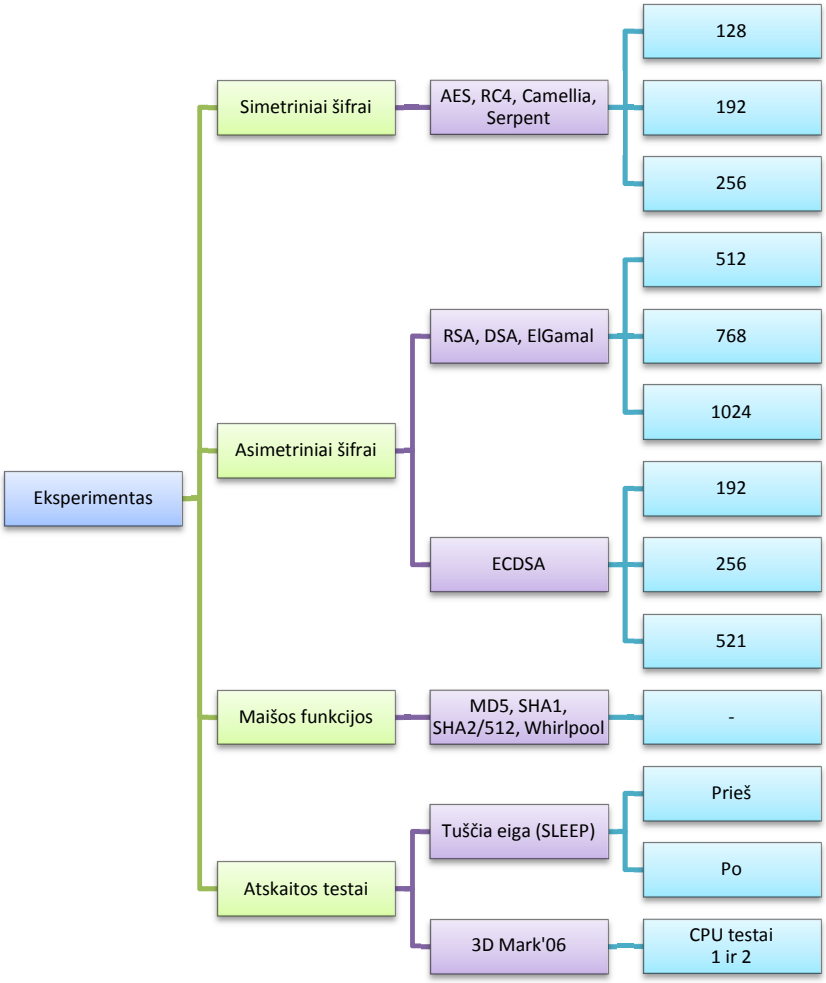
Eksperimentinio tyrimo medis pavaizduotas pav. 17. Eksperimento metu atlikti ir kontroliniai matavimai – du kartus buvo vykdomas energijos sąnaudų matavimas tuščia eiga ir 3DMark'06 CPU testas. Energijos sąnaudų matavimas tuščia eiga vykdytas siekiant patikrinti baterijos susidėvėjimą. Kiekvienas iš pav. 17 parodytų matavimų atliktas tris kartus, skaičiavimam buvo naudojamos vidutinės trijų matavimų reikšmės. Iš viso atlikta:

- 4 simetriniai kriptoalgoritmai su trim skirtingais raktų ilgiais – 12 bandymų;
- 4 asimetriniai kriptoalgoritmai su trim skirtingais raktų ilgiais – 12 bandymų;
- 4 maišos funkcijos bandymai;
- 2 bandymai tuščia eiga;
- 3DMark'06 CPU bandymas;

Iš čia gauname, jog viso atlikta:

$$n = 3(12 + 12 + 4 + 2 + 1) = 93$$

matavimo ciklų. Galima teigti, jog atlikta apytiksliai tiek pat pilnų įkrovimo-iškrovimo ciklų (nors baterija nevisais atvejai buvo pilnai iškraunama).



17 pav. Eksperimentinio tyrimo medis

2.4. Energijos sąnaudų matavimo metodika

Eksperimentui atlikti buvo pasirinktas šifravimas ir maišos rezultatų skaičiavimas. Apdorojamas buvo viešai prieinamas ir jau tapęs standartiniu tokio tipo eksperimentuose pradinių duomenų failas: spalvota Lenos nuotrauka (žr. pav. 18) [40]. Pasirinktas failas yra 512x512 taškų dydžio, 24 bitų spalvų gylio TIFF formato piešinys. Failo dydis baitais yra 786572. Failo maišos reikšmės:

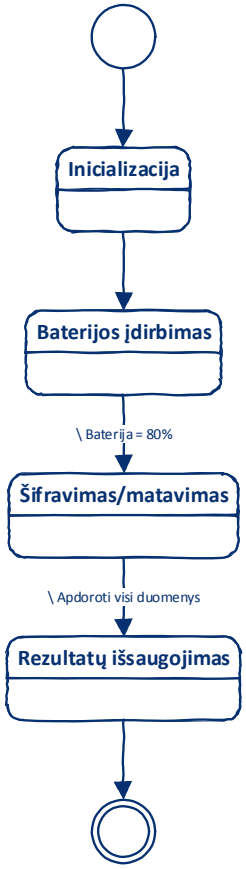
- **MD5**
  `7278246CF26B76E0CA398E7F739B527E`
- **SHA1**
  `E647D0F6736F82E498DE8398ECCC48CF0A7D53B9`
- **SHA2/512**
  `2CB9D7DF53EB8640DC48D736974F472A98D9C7186DE7A972490455F5F3ED29DF`
  `C5B75C95CCB3ED4596BC2BFC4B1E52CF4D76BCEE27D334DD155BB426617392DC`
- **Whirlpool**
  `26E888D647C42781B6C43C7A79ADE65A0FD741518506556C168B4A012D37F62C`
  `736CBC65BF58C0F5183E78F1B4CEAC02223BCDCDB8BCE8C66B0C8951F76B1ABB`

18 pav. Lena.tiff (4.2.04.tiff)

Visi matavimai buvo atliekami į atmintį užkrovus minėtą pradinių duomenų failą, jį pakartojus tiek kartų, kad būtų užpildytas visas numatytas atminties buferis (žr. lentelę 15). Buferis talpino visus duomenis kurie buvo šifruojami, taip buvo stengiamasi išvengti diskinių operacijų reikalingų pakrauti duomenis į atmintį ir tuo pačiu nebuvo vykdomos atminties išskyrimo ir atlaisvinimo operacijos. Siekiant išlaikyti eksperimento pastovumą, buvo pasirinkta šifruoti 20 baitų ilgio (160 bitų) blokais, nes toks yra maksimalus DSA algoritmo palaikomas šifruojamų (pasirašomų) duomenų masyvo ilgis. Šis sprendimas dar buvo sustiprintas ir to, kad labai dažnai praktikoje yra šifruojami duomenų paketai, kurie dažnai būna pakankamai trumpi. Dėl tos pačios priežasties, bei todėl, kad eksperimente buvo tiriamos ir vienkryptės maišos funkcijos, duomenys nebuvo iššifruojami, t.y. eksperimente buvo dirbama tik su viena hipotetinės paskirstytosios sistemos dalimi.

Siekiant išvengti pilnos ir baigiančios išsikrauti baterijos talpos netiesiškumo [41], eksperimentas buvo atliekamas tokia seka (žr. diagramą pav. 19):

1) Pilnai pakraunama baterija;
2) Paleidžiama ir inicializuojama matavimo PĮ;
3) Generuojama dirbtinė apkrova, kol baterijos įkrovos lygis nukrenta iki 80%;
4) Pradedamas šifravimas ir kompiuterio parametrų matavimai;
5) Šifravimas baigiamas apdorojus visus numatytus duomenis;
6) Išsaugomi matavimų rezultatai ir baigiamas darbas.



19 pav. Programinės įrangos būsenų diagrama

Vykdant eksperimentą duomenų kiekis skirtas apdorojimui buvo parenkamas eksperimentiniu būdu – pirmas šifravimo ciklas kiekvienam algoritmui ir rakto ilgiui buvo vykdomas prie tuo metu esančio baterijos lygio ir apdorojamas 1MB duomenų su (ECDSA atveju 10kB). Pagal tai kiek baterijos įkrovos buvo sunaudota šiam pirmam ciklui ekstrapoliacijos metodu paskaičiuojama kiek įmanoma apdoroti duomenų, kad baterija nuo 80% išsikrautų iki 20%. Kaip matysis eksperimento rezultatuose, skaičiavimas ne visada buvo itin tikslus. Atlikus šiuos paruošiamuosius darbus buvo pilnai įkraunama baterija ir atliekamas, aukščiau aprašytas, eksperimentas. Eksperimentas kiekvienam kriptografiniam algoritmui ir rakto ilgiui buvo kartojamas tris kartus ir tyrimui naudoti suvidurkinti rezultato duomenys.

Pirmas eksperimento ciklas buvo atliktas matuojant baterijos išsikrovimą be apkrovos. Tarp eksperimento ciklų kompiuteris nebuvo naudojamas kitiem darbam. Baigus kriptografinių algoritmų eksperimentinį tyrimą, buvo atliekamas 3DMark'06 etaloninis energijos sąnaudų matavimas. Paskutinis eksperimento ciklas buvo atliktas identiškas pirmam – be apkrovos. Pirmas ir paskutinis ciklai skirti tam, kad būtų galima įvertinti baterijos susidėvėjimą, nes viso eksperimento metu buvo atlikti 93 pilni baterijos įkrovimo/iškrovimo ciklai.

2.5. Tiriami kriptografiniai algoritmai

Išanalizavus literatūrą ir kitų mokslininkų vykdytus tyrimus, eksperimentui atlikti buvo pasirinkta po keturis simetrinius, asimetrinius (viešojo rakto) ir maišos kriptografinius algoritmus (žr. lentelę 15). Kriptoalgoritmų raktų ilgiai buvo pasirinkti atsižvelgus į ECRYPT2

rekomendacijas, kurių siūloma laikytis iki 2015 metų [16]. Pagal atrinktus raktų ilgius matome, kad tyrimui naudojami algoritmai atitinka ECRYPT2 rekomendacijų saugos lygius nuo 7 (ilgalaikė apsauga) iki 8 (apsauga „numatomai ateičiai“). Išimtis yra RSA, DSA ir ElGamal kriptoalgoritmai, kurie tiriami su raktų ilgiais atitinkančiais 1 – 3 saugos lygius. Toks pasirinkimas nulemtas tuo, kad RSA algoritmas skaitmeniniuose sertifikatuose dažniausiai naudojamas su 1024 bitų ilgio raktu. Siekiant išlaikyti tyrimo nuoseklumą kitų asimetrinių šifrų raktų ilgiai buvo pasirinkti atsižvelgiant į RSA.

15 lentelė. Tyrimui naudoti kriptografiniai algoritmai

Algoritmas	Tipas	Rakto ilgis	Saugos lygis pagal ECRYPT2	Biblioteka	Duomenų kiekis, MB
AES	Simetrinis, blokinis	128	7	BouncyCastle	128000
		192	7-8		128000
		256	8		128000
RC4	Simetrinis, srautinis	128	7	BouncyCastle	26000
		192	7-8		26000
		256	8		26000
Camellia	Simetrinis, blokinis	128	7	BouncyCastle	128000
		192	7-8		128000
		256	8		98304
Serpent		128	7	BouncyCastle	65536
		192	7-8		65536
		256	8		65536
RSA	Asimetrinis	512	1-2	.NET 3.5	200
		768	2		100
		1024	3		50
DSA		512	1-2	.NET 3.5	200
		768	2		200
		1024	3		100
ECDSA		192	5	BouncyCastle	2
		256	7		1
		521	8+		0,2
ElGamal		512	1-2	BouncyCastle	10
		768	2		4
		1024	3		2
MD5	Maišos funkcija	–	–	BouncyCastle	131072
SHA1		–	–		65536
SHA2/512		–	–		20480
Whirlpool		–	–		8192

2.1. Išvados

- Pasiūlytas baterijos elgsenos modelis kuris sudarytas iš trijų pagrindinių komponentų:
  - *Apkrovos-talpos efektas* stebimas, kai baterija iškraunama pastoviai;
  - *Atsistatymo efektas* stebimas, kaip baterijos likutinės energijos atsistatymas bateriją iškraunant maža srove arba jos visai neiškraunant ilgesnį laiko tarpą;
  - *Programinis energijos sąnaudų planavimas* – procesas apima pavyzdžiui procesoriaus taktinio dažnio ir/arba maitinimo įtampos dinaminį keitimą, periferinių įrenginių atjungimą kurie nėra naudojami, ekrano ryškumo keitimą ir pan.

- Sukurta ir matematine lygtimi aprašyta metrika, kuri paremta pasiūlytu baterijos iškrovimo vėlinimo modeliu. Metrika leidžia įvertinti baterijos elgseną dinaminėje apkrovoje naudojant etaloninio testo rezultatų skaitines reikšmes.
- Sukurta programinė įranga baterijos energijos sąnaudų matavimui taikomųjų programų lygmenyje įvertinant sistemos apkrovimą;
- Pasiūlyta kriptografinių algoritmų tyrimo metodika.

3. EKSPERIMENTINIS ENERGIJOS SĄNAUDŲ TYRIMAS

Remiantis skyriuje 2.5 pasiūlytu kriptografinių algoritmų tyrimo medžiu atlikti tyrimai naudojant techninę įrangą:

- Nešiojamasis kompiuteris DELL Latitude D420
  - Intel Core Duo (Yonah) 1,20GHz (MMX, SSE, SSE2, SSE3, VT-x)
  - 12,1“ WXGA TFT LCD
  - Intel GMA 945GM integruota grafinė posistemė, 224MB bendro naudojimo atminties
  - 1GB DDR2-266 SDRAM 2xDIMM (512MB integruota)
  - Toshiba 60GB 4200RPM 1,8“ kietasis diskas
  - Intel PRO/Wireless 3945ABG
  - Nežinomo gamintojo, neoriginali, 9000mAh LiIo baterija
  - Microsoft Windows XP Professional SP3
  - Microsoft .NET 4

Tikslios eksperimente naudotos techninės įrangos charakteristikos gautos naudojant CPU-Z ir GPU-Z programas pateiktos priede 6.2 pav. nuo 37 iki 44. Siekiant eksperimentą atlikti realiomis sąlygomis kompiuterio ir operacinės sistemos nustatymai buvo tokie:

- Minimalus ekrano ryškumas
- Pasirinktas „Allways on“ energijos profilis
- Monitoriaus ir kietojo disko išjungimas bei sistemos užmigimo (*angl. stand-by*) funkcijos atjungtos
- Visos vartotojo lygio programos išjungtos
- Antivirusinė programinė įranga (AVG Free Edition 2013) palikta veikianti

3.1. Kriptografinių algoritmų energijos sąnaudų eksperimentinis tyrimas

Atliktų tyrimų rezultatai apibendrinti ir pateikti suvestinėje (žr. lent. 16).

16 lentelė. Bendra eksperimento suvestinė

Algoritmas	Tipas	Rakto ilgis	Duomenų kiekis, MB	Bendro duomenų kiekio kartojimai	Blokas ,baitai	Gijos	Viso apdorota, MB	Darbo laikas, s	Vidutinė CPU apkrova, %	Baterija, %			Laikas 1MB šifruoti, s	mAh/MB	Laikas 1% baterijos suvartojimui, s	Laikas bloko šifravimui, ms
										Nuo	Iki	Išnaudota				
Empty-Pries	**-**	-	-	-	-	-	-	15016	0,4	100	20	80	-	-	187,70	-
Empty-Po	**-**	-	-	-	-	-	-	13904	0,4	100	20	80	-	-	173,80	-
3D Mark'06	**-**	-	-	-	-	-	-	7574	90,7	80	10	70	-	-	108,20	-
RSA	**Asimetrinis**	512	10	10	20	2	200	4776	99,5	80	42	38	23,88	0,19000	125,68	0,4555
		768	5	10	20	2	100	6424	99,5	80	28	52	64,24	0,52000	123,54	1,2253
		1024	5	5	20	2	50	6645	99,3	80	26	54	132,90	1,08000	123,06	2,5349
DSA	**Asimetrinis**	512	10	10	20	2	200	4750	99,4	80	43	37	23,75	0,18500	128,38	0,4530
		768	10	10	20	2	200	8362	99,2	80	11	69	41,81	0,34500	121,19	0,7975
		1024	10	5	20	2	100	6569	99,3	80	27	53	65,69	0,53000	123,94	1,2529
ECDSA	**Asimetrinis**	192	1	1	20	2	2	5476	95,6	80	33	47	2738,00	23,50000	116,51	52,2232
		256	0,1	5	20	2	1	5266	95,7	80	35	45	5266,00	45,00000	117,02	100,4410
		521	0,1	1	20	2	0,2	5563	96,2	80	33	47	27815,00	235,00000	118,36	530,5290
ElGamal	**Asimetrinis**	512	1	5	20	2	10	5222	99,4	80	37	43	522,20	4,30000	121,44	9,9602
		768	1	2	20	2	4	6804	99,4	80	23	57	1701,00	14,25000	119,37	32,4440
		1024	1	1	20	2	2	7953	99,4	80	7	73	3976,50	36,50000	108,95	75,8457
Aes	**Simetrinis**	128	256	250	20	2	128000	5172	98,4	80	34	46	0,04	0,00036	112,43	0,0008
		192	256	250	20	2	128000	5686	98,0	80	27	53	0,04	0,00041	107,28	0,0008
		256	256	250	20	2	128000	6424	98,2	80	21	59	0,05	0,00046	108,88	0,0010
RC4	**Simetrinis**	128	100	130	20	2	26000	7324	99,5	80	20	60	0,28	0,00231	122,07	0,0054
		192	100	130	20	2	26000	7279	99,3	80	20	60	0,28	0,00231	121,32	0,0053
		256	100	130	20	2	26000	7344	99,3	80	22	58	0,28	0,00223	126,62	0,0054
Camellia	**Simetrinis**	128	256	250	20	2	128000	5390	97,6	80	31	49	0,04	0,00038	110,00	0,0008
		192	256	250	20	2	128000	6420	98,1	80	20	60	0,05	0,00047	107,00	0,0010

		256	256	192	20	2	98304	5060	98,7	80	35	45	0,05	0,00046	112,44	0,0010
Serpent	**Simetrinis**	128	256	128	20	2	65536	6211	99,1	80	24	56	0,09	0,00085	110,91	0,0018
		192	256	128	20	2	65536	6221	98,8	80	22	58	0,09	0,00089	107,26	0,0018
		256	256	128	20	2	65536	6338	99,3	80	23	57	0,10	0,00087	111,19	0,0018
MD5	**Santrauka**	-	256	256	20	2	131072	7772	99,1	80	8	72	0,06	0,00055	107,94	0,0011
SHA1	**Santrauka**	-	256	128	20	2	65536	6353	99,5	80	23	57	0,10	0,00087	111,46	0,0018
SHA512	**Santrauka**	-	256	40	20	2	20480	7269	99,1	80	16	64	0,35	0,00313	113,58	0,0068
Whirlpool	**Santrauka**	-	256	16	20	2	8192	7491	99,4	80	12	68	0,91	0,00830	110,16	0,0174

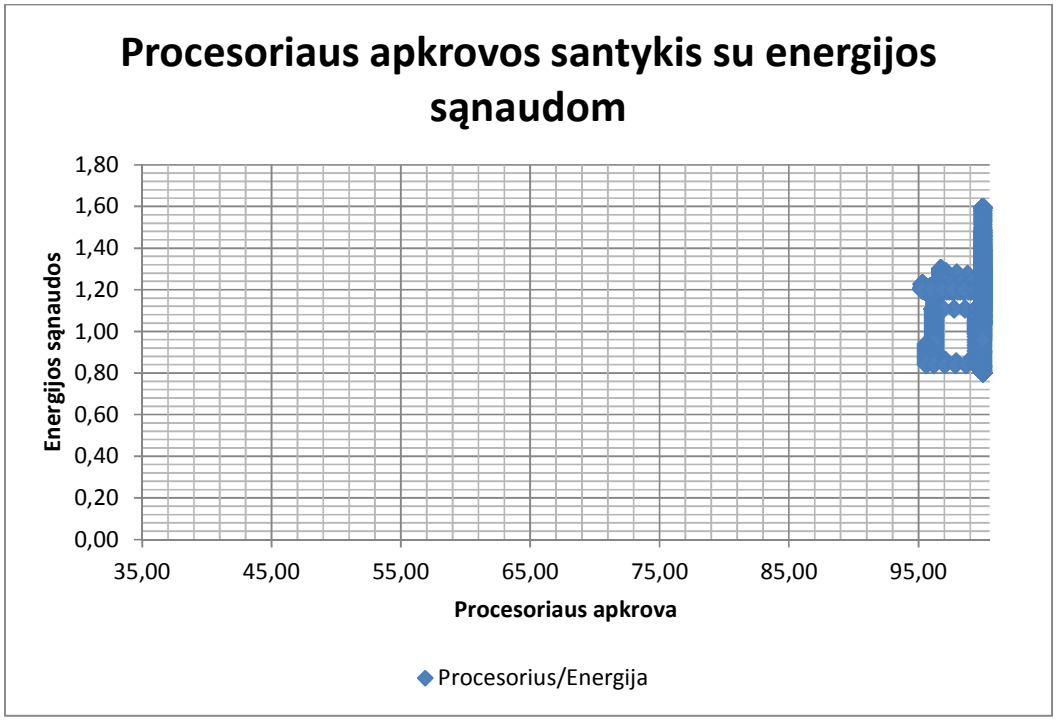
Eksperimento suvestinėje (žr. 16 lentelę) pavaizduotų stulpelių reikšmės yra:

- **Algoritmas** – (kripto) algoritmas naudotas matavimams atlikti;
- **Tipas** – kriptografinio algoritmo tipas;
- **Rakto ilgis** – šifro rakto ilgis;
- **Duomenų kiekis** – pradinių duomenų masyvo ilgis;
- **Bendro duomenų kiekio kartojimai** – kiek kartų buvo pakartotas duomenų bloko šifravimas;
- **Blokas** – viena iteracija šifruojamo bloko ilgis;
- **Gijos** – kiek gijų buvo naudojama lygiagrečiam tų pačių duomenų šifravimui;
- **Viso apdorota** – bendras užšifruotų duomenų kiekis;
- **Darbo laikas** – laikas sekundėmis, kiek kompiuteris užtruko šifruodamas duomenis;
- **Vidutinė CPU apkrova** – vidutinė procesoriaus apkrova procentais šifravimo metu;
- **Baterija nuo** – kompiuterio baterijos įkrovos reikšmė pradedant šifravimą;
- **Baterija iki** – baterijos įkrovos reikšmė procentais baigus šifravimą;
- **Baterija išnaudota** – kiek procentų baterijos buvo sunaudota šifravimui;
- **Laikas 1MB šifruoti** – vidutinis laikas sekundėmis kurį kompiuteris užtruko šifruodamas vieną megabaitą duomenų;
- **mAh/MB** – vidutinis sunaudotos energijos kiekis miliampervalandėmis kurį kompiuteris užtruko šifruodamas vieną megabaitą duomenų;
- **Laikas 1%** – vidutinis laikas sekundėmis per kurį baterija išsikrovė vienu procentu (90mAh);
- **Laikas bloko šifravimui** – vidutinis laikas milisekundėmis kurį kompiuteris užtruko šifruodamas 160 bitų (20 baitų) duomenų bloką.

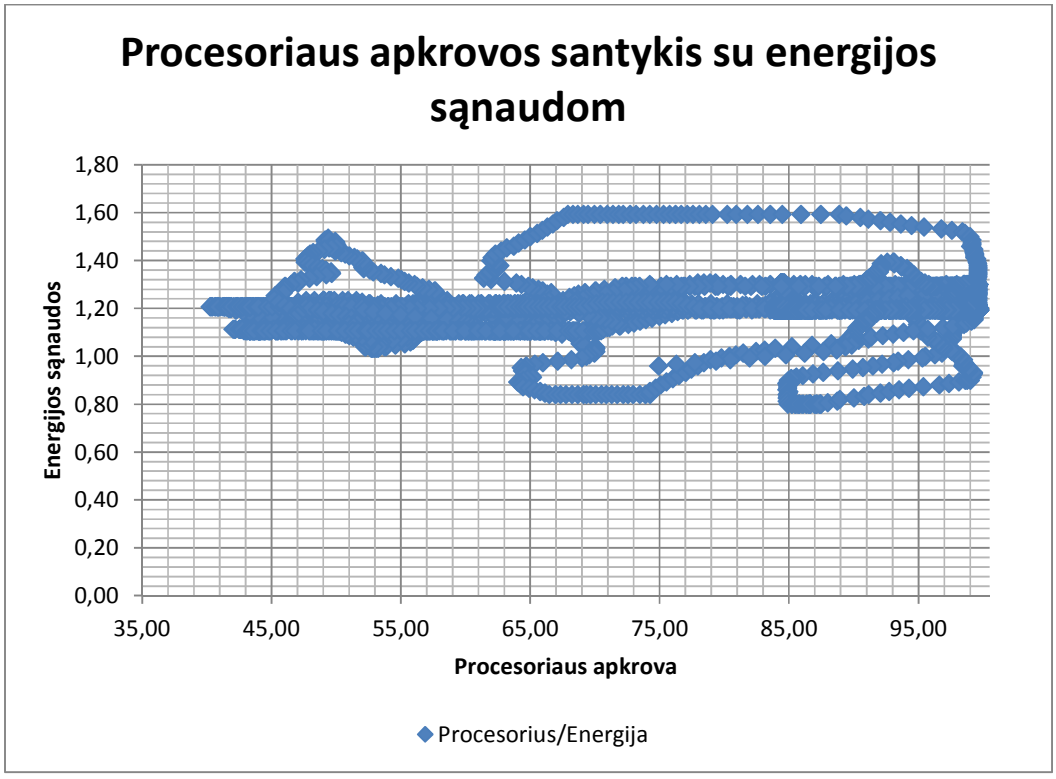
Pav. 20 parodytas baterijos energijos sąnaudų įvertinimas naudojant mūsų pasiūlytą vėlinantį energijos sąnaudų modelį [1] šifruojant ElGamal-1024 šifru. Analizuojant grafiką pastebėta, jog energijos sąnaudų dinamiškumas atvaizduojamas labai siaurame procesoriaus apkrovos diapazone. Dėl šios priežasties nutarta išplėsti eksperimentą tam, kad įvertinti energijos sąnaudų kitimą platesniame dinaminiame diapazone. Šiam tikslui pasiekti pasiūlyta naudoti 3Dmark'06 kaip etaloninę energijos sąnaudų įvertinimo priemonę. Bandymui buvo pasirinktas CPU 1-2 testas kartojamas vieną kartą 3DMark programoje, tačiau tris kartus kaip atskiras eksperimentas.

Atlikus abu eksperimentus (žr. pav. 20 ir 21) pastebėjome jog [1]:

- Baterija priklauso vėlinančioms sistemoms. Eksperimentiškai nustatytos vėlinimo reikšmės yra:
  - Procesoriaus vėlinimas: 340s
  - Fizinės atminties išskyrimo vėlinimas: 340s
  - Fizinės atminties atlaisvinimo vėlinimas: 180s
- Nenustatyta stiprios koreliacijos tarp procesoriaus apkrovos ir energijos sąnaudų;
- Vėlinimas tarp energiją vartojančių įvykių ir baterijos įkrovos pokyčių apsunkina baterijos įkrovos prognozavimą;
- Pasiūlytas vėlinimo modelis gali būti naudojamas kaip teorinis pagrindas energijos sąnaudų etaloniniams matavimam atlikti;
- Mes pasiūlėme formulę (39) kurios pagalba galima paskaičiuoti energinį našumą pasitelkus 3DMark'06 programą ir atlikus energijos sąnaudų matavimus.



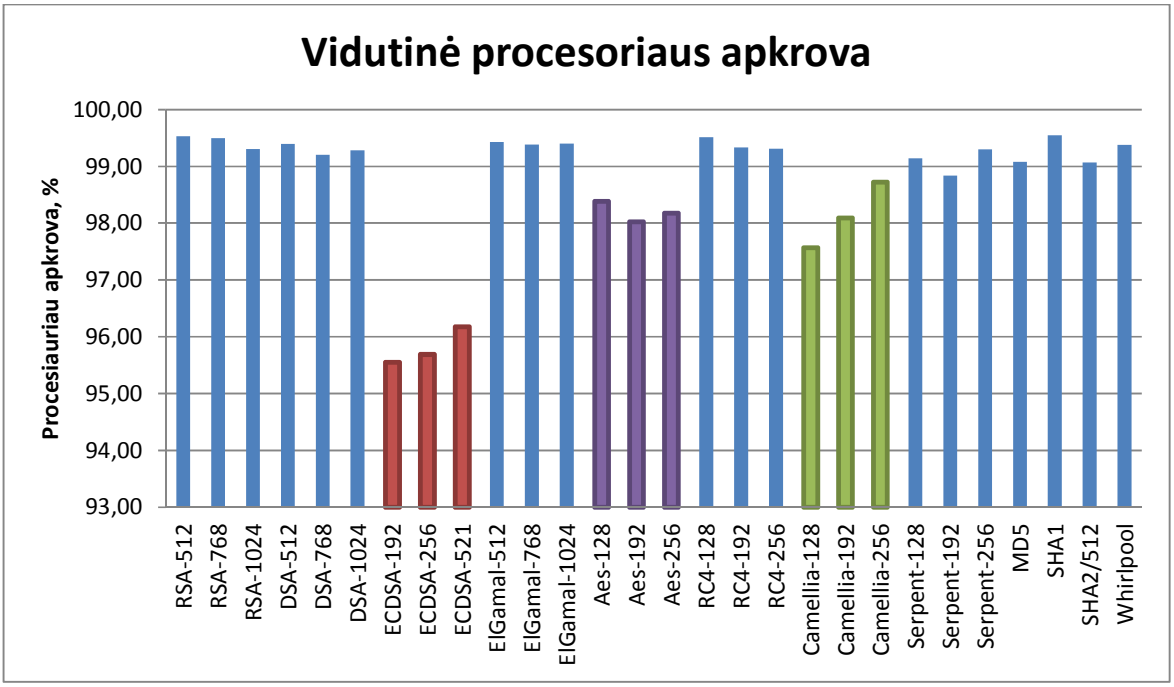
20 pav. Procesoriaus apkrovos santykis su energijos sąnaudomis (ElGamal-1024)



21 pav. Procesoriaus apkrovos santykis su energijos sąnaudomis (3DMark‘06)

Atlikus eksperimentus su kriptografiniais algoritmais paskaičiavome vidutinę kiekvieno iš šifrų generuojamą apkrovą centriniam procesoriui (pav. 22). Iš grafikų matome, kad:

- Atliekant tyrimą su išrinktais kriptografiniais algoritmais nustatyta, kad vidutinė procesoriaus apkrova yra tarp 95% ir 100%;
- Tyrimo rezultatai parodė, kad mažiausią procesoriaus apkrovą diapazone tarp 95% ir 96% pasiekta su ECDSA kriptoalgotitmu. AES algoritmo apkrova tarp 98% ir 99%, o Camellia tarp 97% ir 99%;
- Visi kiti tirti kriptografiniai algoritmai procesorių apkrovė diapazone nuo 99% iki 100%.

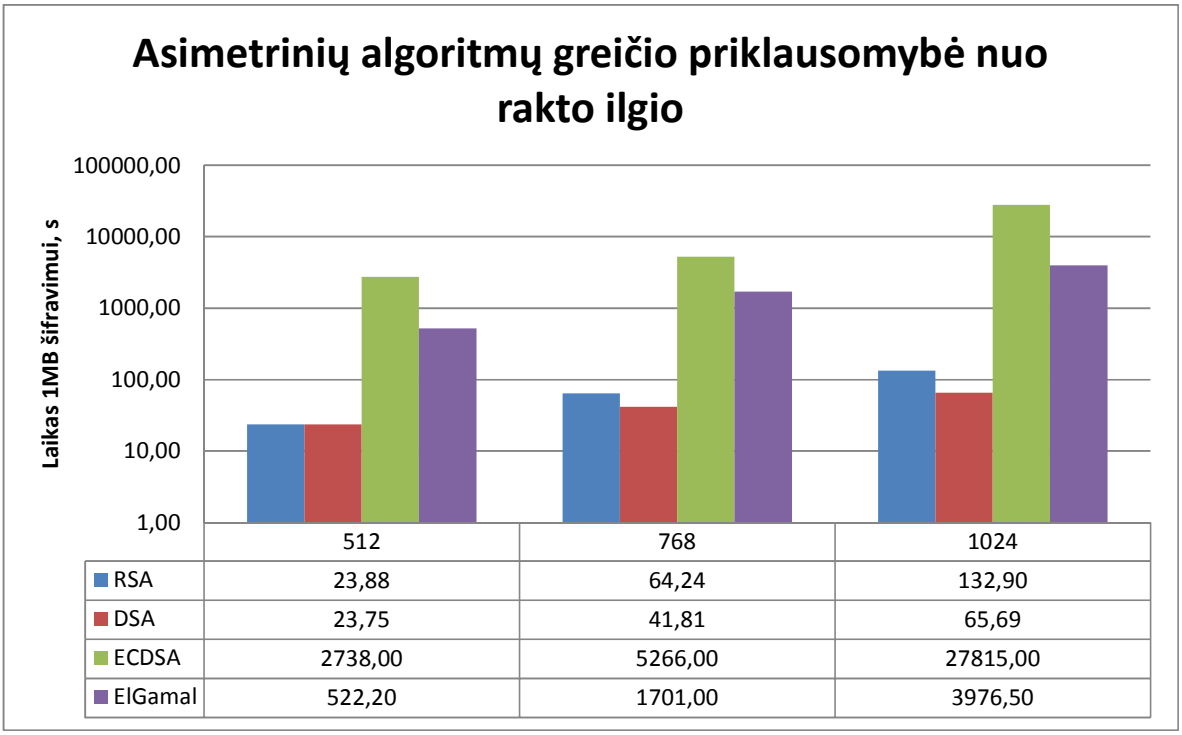


22 pav. Vidutinė procesoriaus apkrova

Pav. 23 parodyta asimetrinių kriptografinių algoritmų darbo greičio (laikas reikalingas 1MB užšifruoti) priklausomybė nuo rakto ilgio. Kaip matosi iš grafiko visi keturi tirti asimetriniai šifrai parodė eksponentinę greičio priklausomybę nuo rakto ilgio:

- Ilgėjant šifravimo raktui ECDSA algoritmo šifravimo greitis sulėtėja 10,2 karto nuo 2738 sekundžių (esant 512 bitų raktui) vienam megabaitui šifruoti iki 27815 sekundžių (esant 1024 bitų raktui).
- Ilgėjant šifravimo raktui ElGamal algoritmo šifravimo greitis sulėtėja 7,6 karto nuo 522,2 sekundės (esant 512 bitų raktui) vienam megabaitui šifruoti iki 3976,5 sekundės (esant 1024 bitų raktui).
- Geriausiai pasirodė DSA ir RSA kriptografiniai algoritmai kurių šifravimo greitis sulėtėjo atitinkamai 2,8 ir 5,6 karto:
    - 512 bitų ilgio raktas atitinkamai 23,75s ir 23,88s;
    - 768 bitų ilgio raktas – 41,81s ir 64,24s;
    - 1024 bitų ilgio raktas – 65,69s ir 132,9s.

- DSA kriptografinis algoritmas naudojantis 1024 bitų ilgio raktą parodė greičiausią šifravimo laiką kuris sudarė 65,69 sekundės, o lėčiausiai – ECDSA kurio šifravimo laikas tomis pačiomis sąlygomis sudarė 7h 43min ir 35sek.

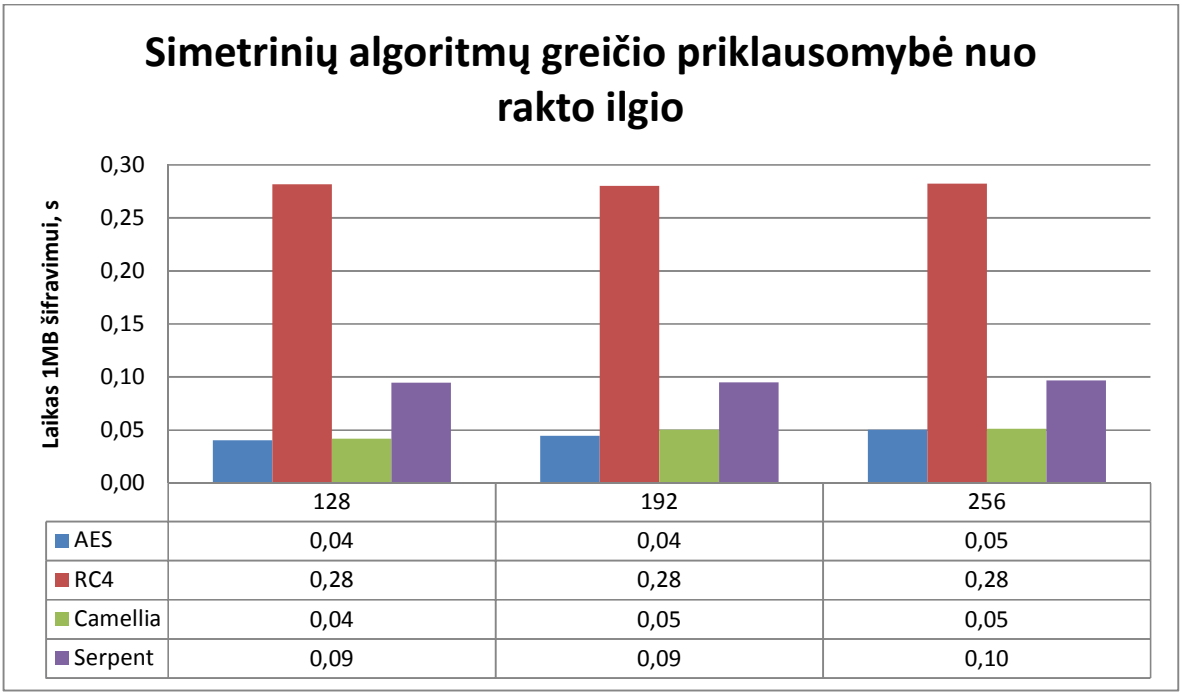


	512	768	1024
RSA	23,88	64,24	132,90
DSA	23,75	41,81	65,69
ECDSA	2738,00	5266,00	27815,00
ElGamal	522,20	1701,00	3976,50

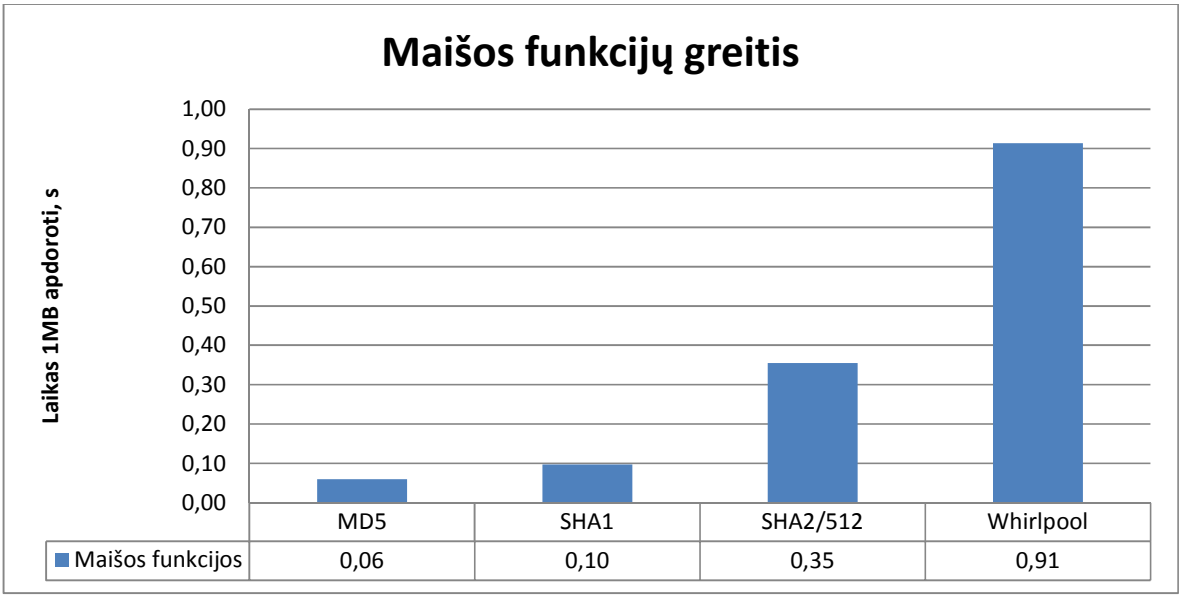
23 pav. Asimetrinių algoritmų greičio priklausomybė nuo rakto ilgio

Simetriniai šifrai, priešingai nei asimetriniai kriptoalgoritmai, kintant rakto ilgiui šifruoja beveik vienodu greičiu. Pažvelgus į pav. 24 greičio pokytis beveik nematomas. Pagal eksperimento duomenis, pateiktus lentelėje 16, matome, kad kintant rakto ilgiui 1MB šifravimo greitis pasikeičia 10-20 milisekundžių. Išimtis yra tik RC4 srautinis šifras kuris parodė vienodą, 0,28s, 1MB šifravimo laiką naudojant 128, 192 ir 256 bitų raktų ilgius.

Greičiausias blokinis šifras – AES (40ms – 50ms vienam megabaitui informacijos šifruoti). Nuo jo Camellia šifras atsiliko 10ms, tačiau tik naudojant 192 bitų ilgio raktą. Trečias pagal greitį yra Serpent šifras, kuris už AES ir Camellia yra du kartus lėtesnis. Lėčiausias iš tirtų simetrinių ir vienintelis tirtas srautinis šifras RC4 kuriam prireikė 280ms užšifruoti vieną megabaitą informacijos.

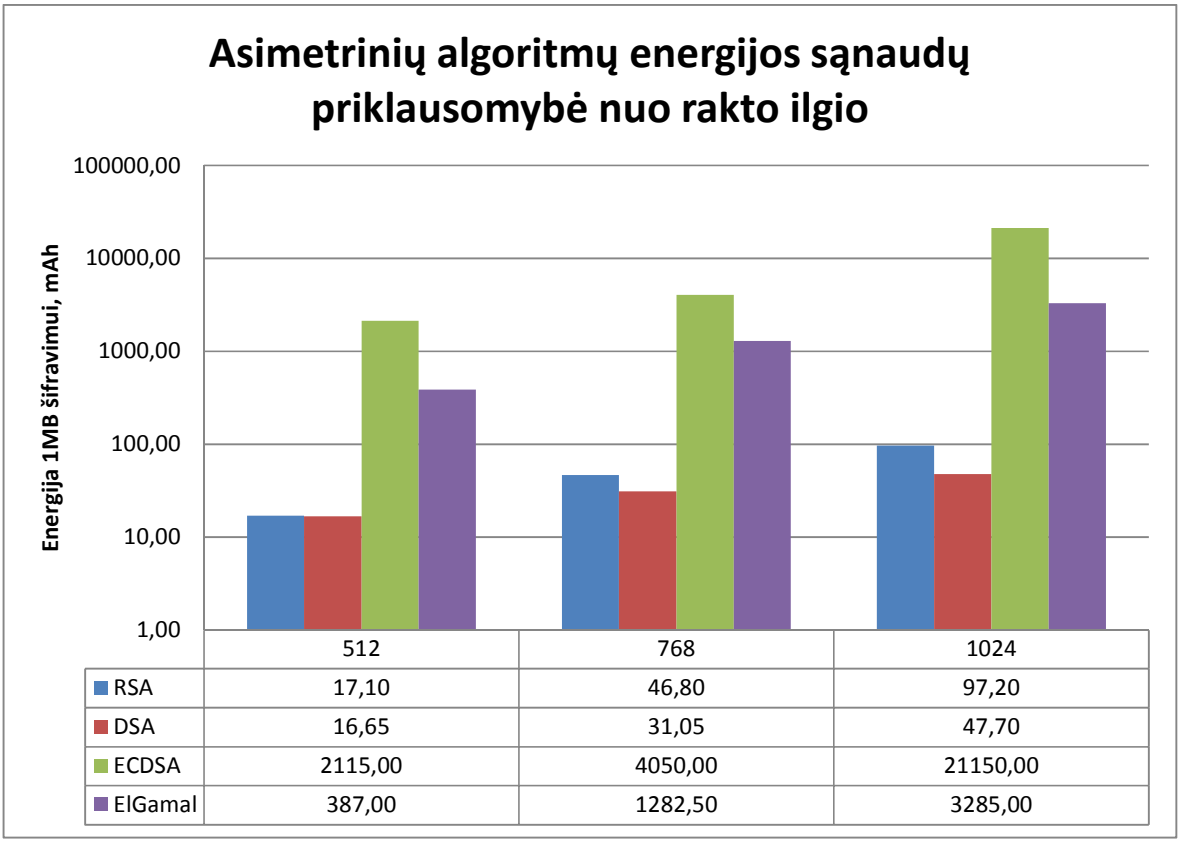


24 pav. Simetrinių algoritmų greičio priklausomybė nuo rakto ilgio



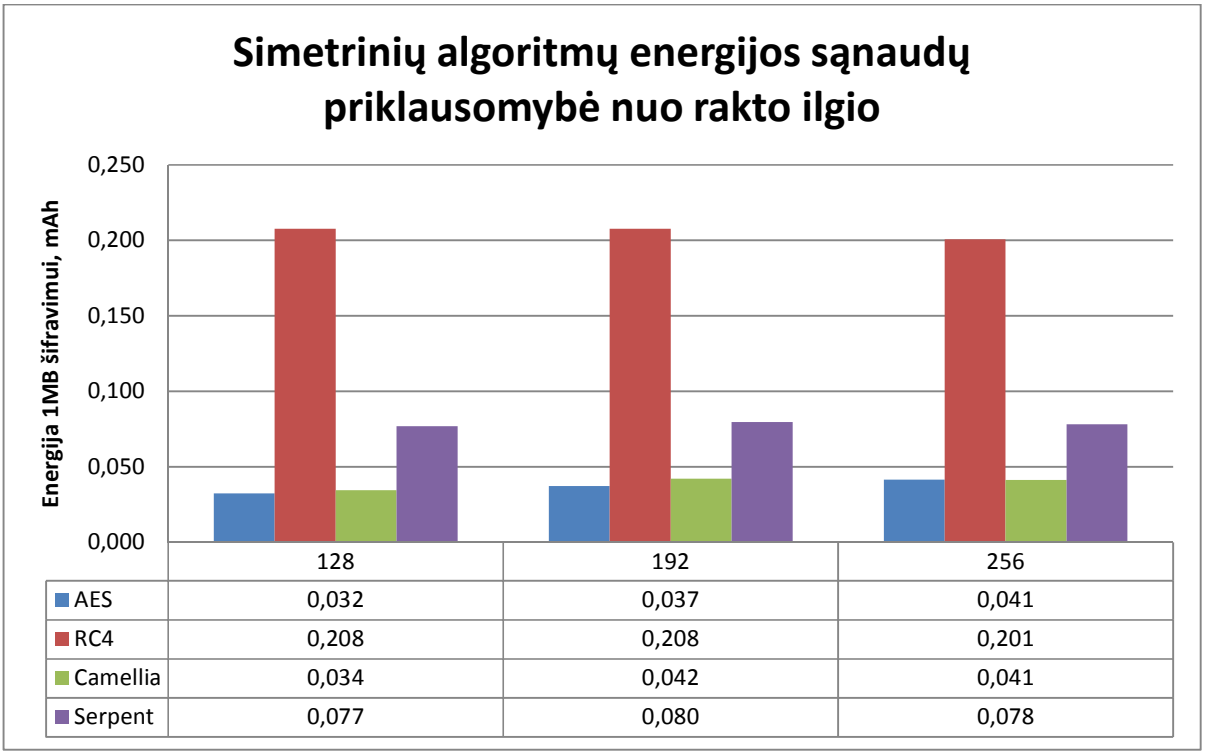
25 pav. Maišos funkcijų greitis

Darbo greičiu skyrėsi visos eksperimente tirtos maišos funkcijos (žr. pav. 25). Greičiausia tirta maišos funkcija yra MD5, kuriai paskaičiuoti 1MB duomenų santrauką prireikė 60ms, toliau rikiuojasi SHA1 (100ms vienam megabaitui) ir SHA2/512 (350ms vienam megabaitui). Lėčiausia maišos funkcija – Whirlpool, kuri už SHA1 yra lėtesnė daugiau nei 9 kartus, o už MD5 net 15 kartų.

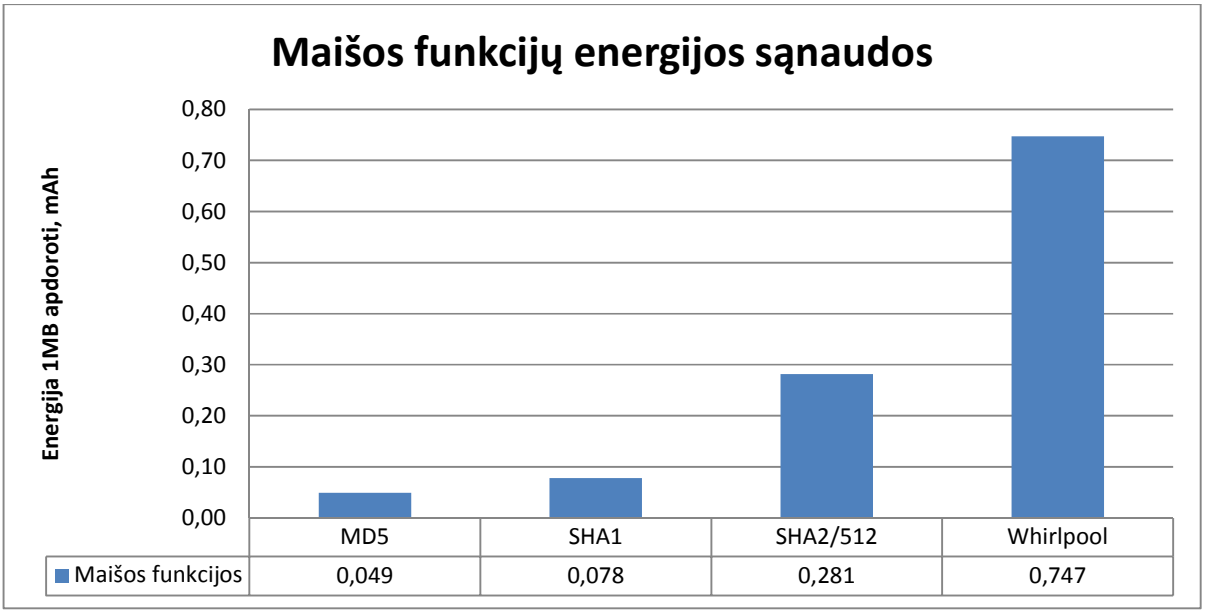


26 pav. Asimetrinių algoritmų energijos sąnaudų priklausomybė nuo rakto ilgio

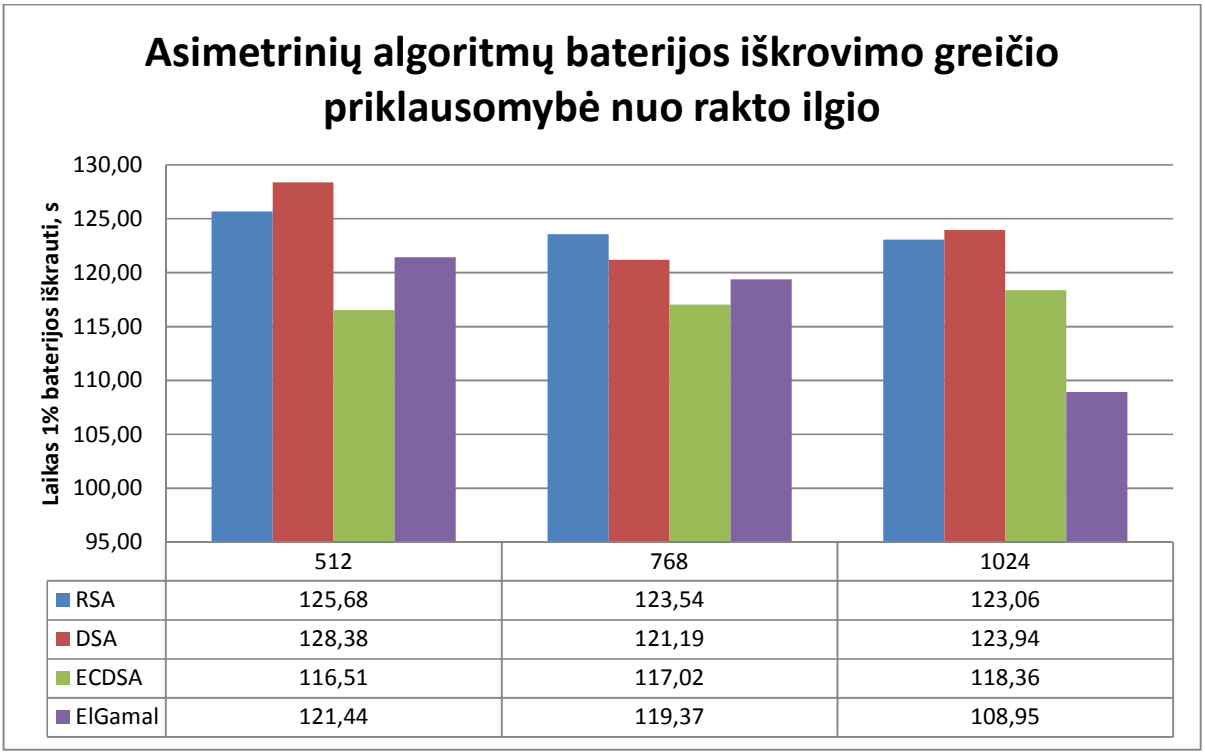
Sulyginus kriptoalgoritmų darbo greičio (pav. 23, 24 ir 25) ir energijos sąnaudų (pav. 26, 27 ir 28) grafikus kiekvienam iš algoritmų matome, kad darbo greitis yra proporcingas suvartotai energijai. Šį santykį pavaizdavome grafiškai (žr. pav. 32). Kaip matosi iš grafiko, santykis kinta nuo 0,7 (DSA-512) iki 0,84 (AES-192, Camellia-192 ir Serpent-192). Laiko ir suvartotos energijos santykių išsibarstymas tarp skirtingų algoritmų ir raktų ilgių rodo, jog šifravimui suvartojama energija priklauso nuo kriptoalgoritmo ir naudojamo rakto ilgio.



27 pav. Simetrinių algoritmų energijos sąnaudų priklausomybė nuo rakto ilgio

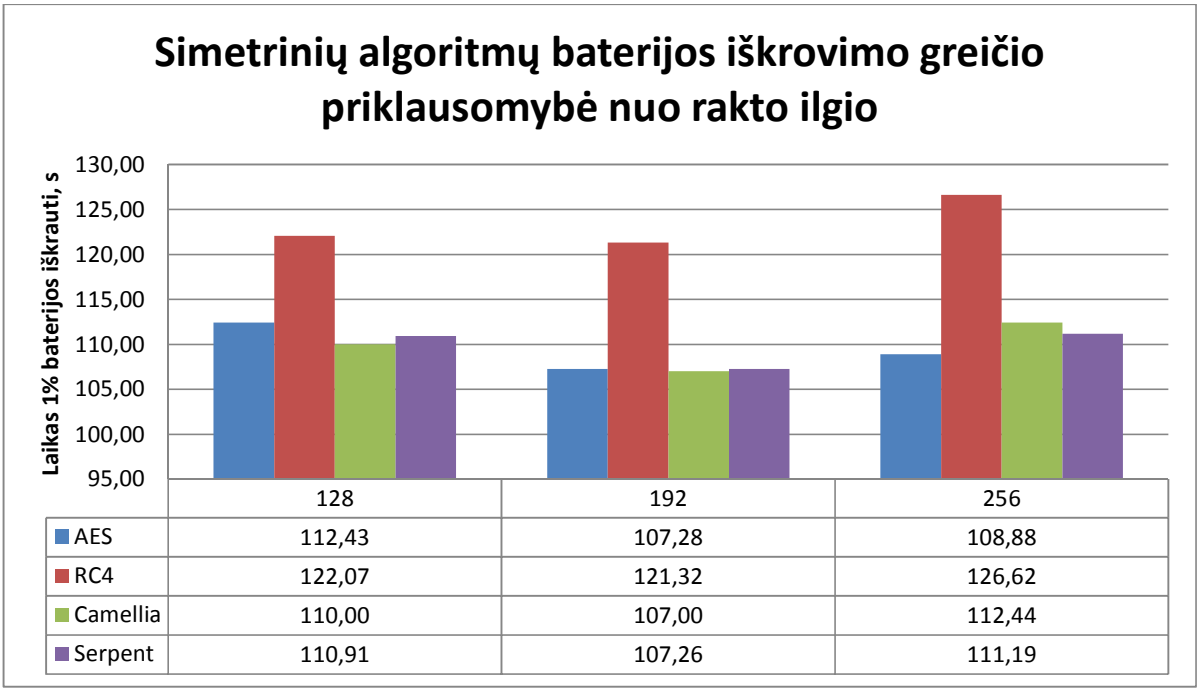


28 pav. Maišos funkcijų energijos sąnaudos



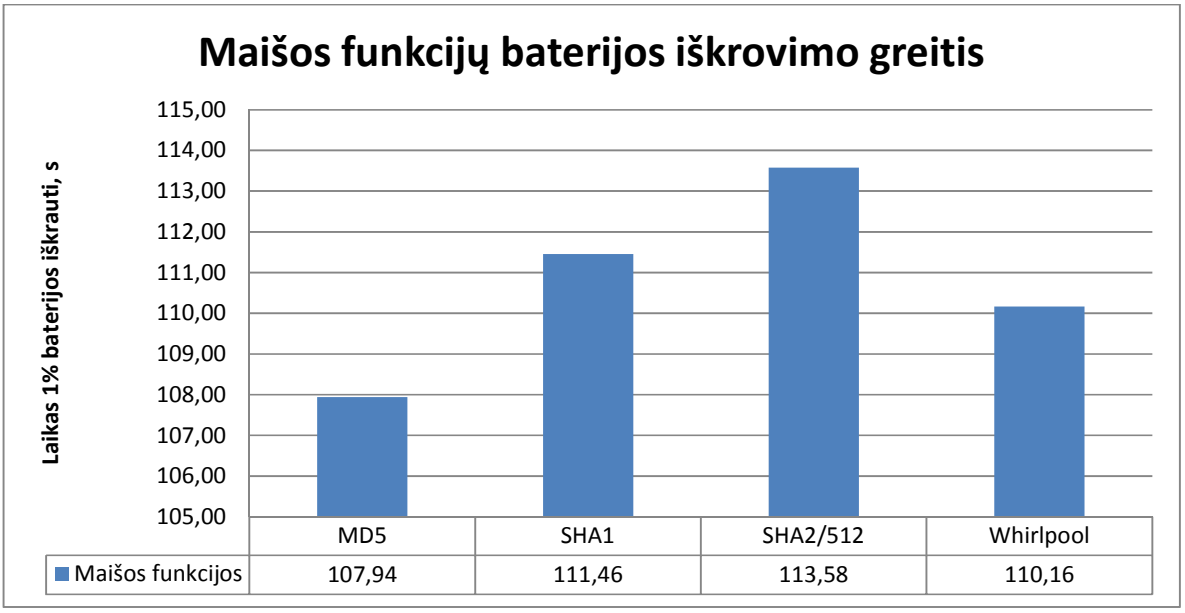
	512	768	1024
RSA	125,68	123,54	123,06
DSA	128,38	121,19	123,94
ECDSA	116,51	117,02	118,36
ElGamal	121,44	119,37	108,95

29 pav. Asimetrinių algoritmų baterijos iškrovimo greičio priklausomybė nuo rakto ilgio

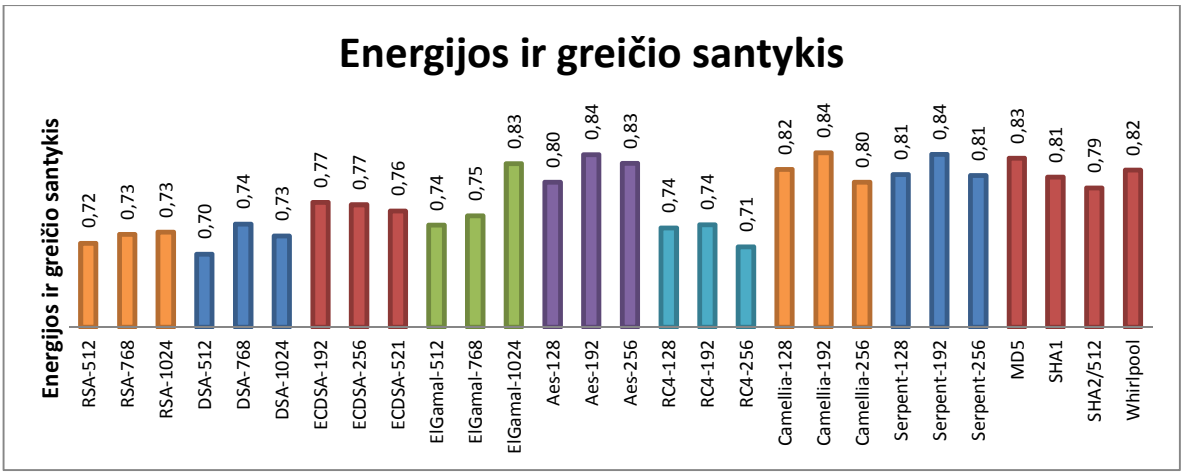


	128	192	256
AES	112,43	107,28	108,88
RC4	122,07	121,32	126,62
Camellia	110,00	107,00	112,44
Serpent	110,91	107,26	111,19

30 pav. Simetrinių algoritmų baterijos iškrovimo greičio priklausomybė nuo rakto ilgio

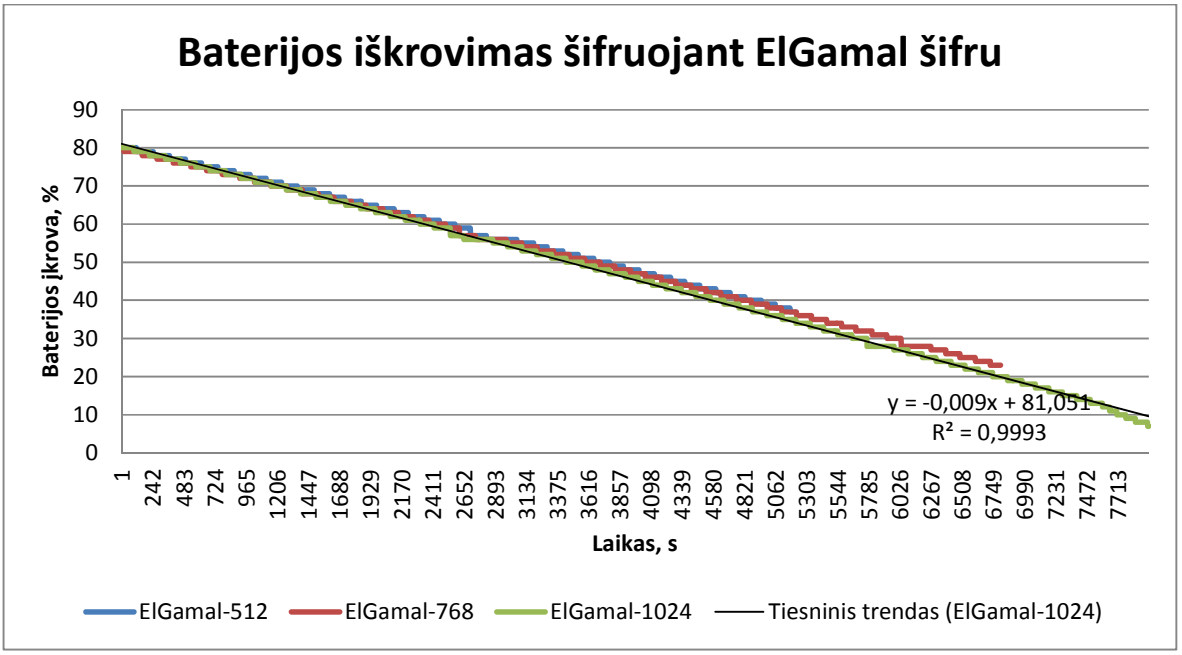


31 pav. Maišos funkcijų baterijos iškrovimo greitis

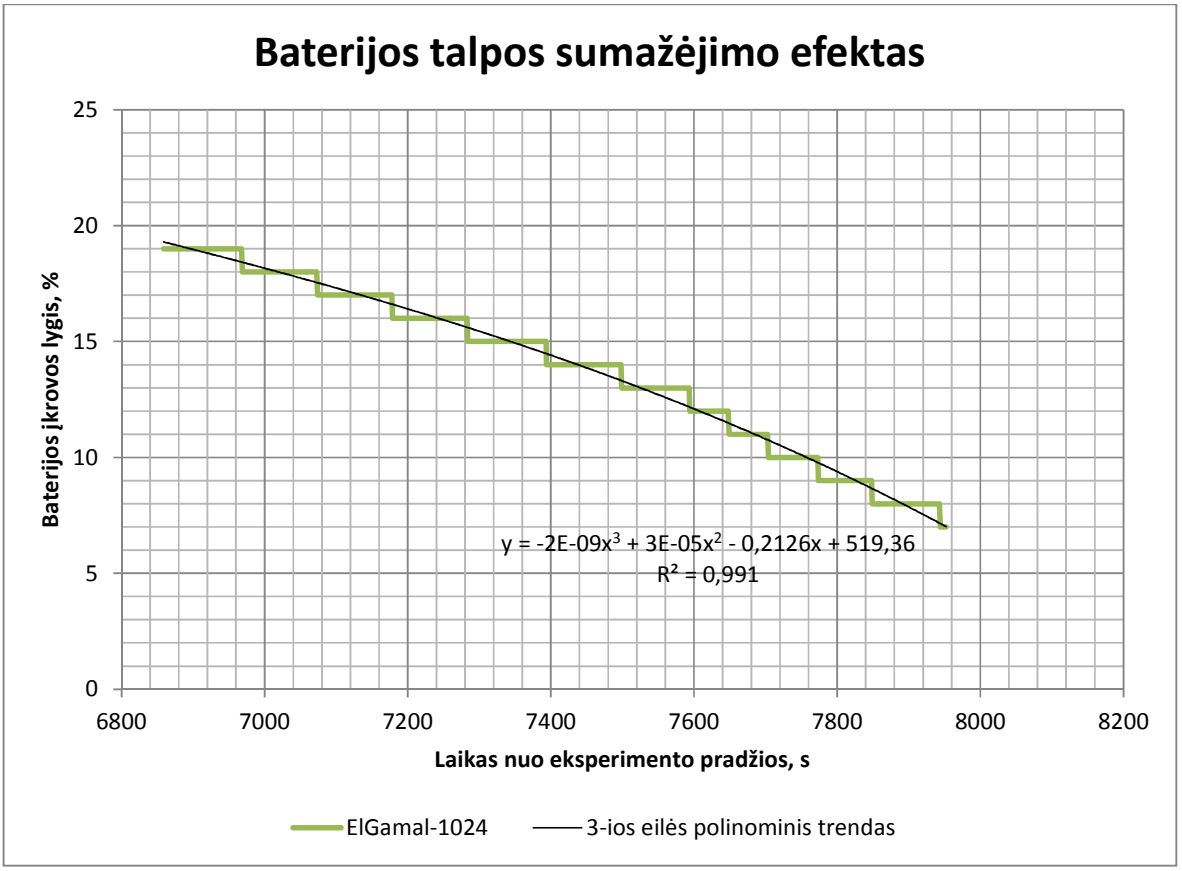


32 pav. Kriptografinių algoritmų energijos ir greičio santykis

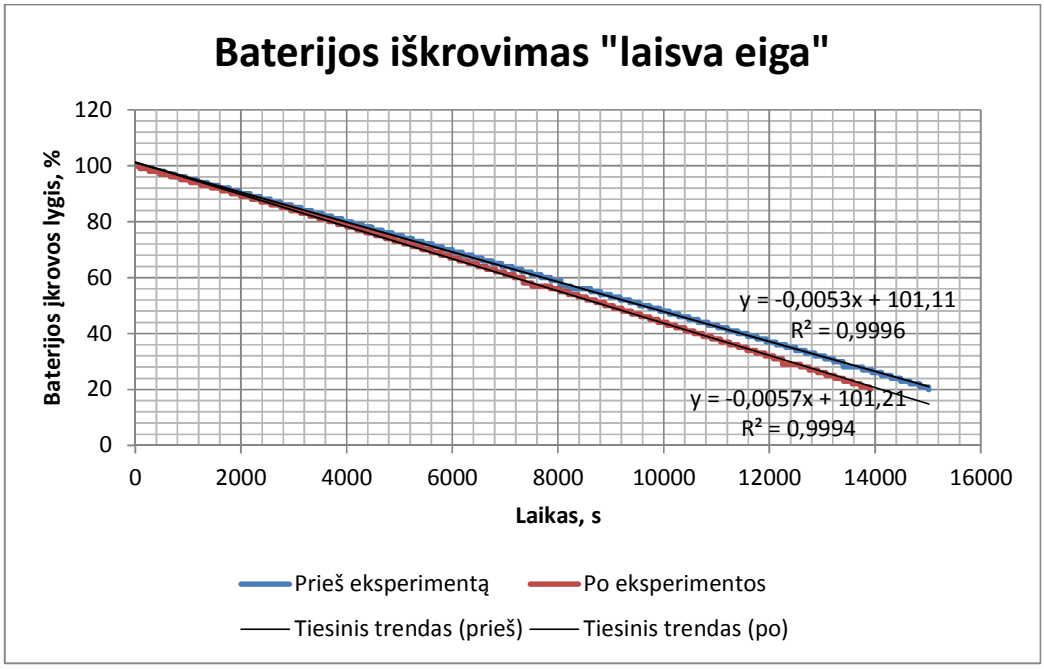
Vykdant bandymus su ElGamal asimetriniu kriptoalgoritmu ir naudojant 1024 bitų ilgio raktą, kompiuterio baterija išsikrovė iki 7% (žr. pav. 33). Grafike aiškiai matosi baterijos talpos sumažėjimo efektas jai baigiant išsikrauti, t.y. baterijos įkrovai artėjant į pabaigą pastebimas greitesnis išsikrovimas. Pav. 34 pavaizduota baterijos iškrovimo pabaiga stambesniu masteliu. Paskaičiuotas ir atvaizduotas trečios eilės polinominis trendas su R^2=0,991 rodo, kad baterijai baigiant išsikrauti jos likutinė talpa kinta netiesiškai.



33 pav. Baterijos iškrovimas šifruojant ElGamal šifru



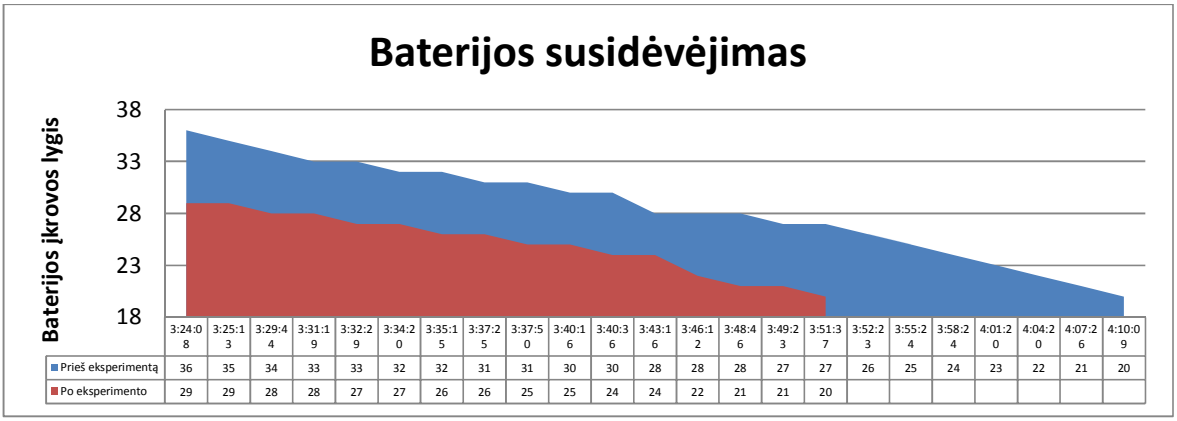
34 pav. Baterijos talpos sumažėjimo efektas



35 pav. Baterijos iškrovimas "laisva eiga"

Eksperimento pradžioje ir pabaigoje buvo atlikti matavimai „tuščia eiga“, t.y. buvo matuojamas baterijos išsikrovimas be jokios procesoriaus ir atminties apkrovos. Bandymo metu veikė tik operacinė sistema ir visa periferija, kuri buvo aktyvi ir visų kitų bandymų metu. Pav. 35 grafiškai pavaizduotas baterijos iškrovimas. Kaip matosi iš tiesinių trendų, baterijos iškrovimas nuo 100% iki 20% (sunaudojant 7200mAh energijos) yra tiesiškas. Buvo pastebėta, kad iškrovimo eksperimento pradžioje ir pabaigoje tiesės nesutampa. Tai rodo baterijos dėvėjimąsi sulig kiekvienu įkrovimo-iškrovimo ciklu.

Baterijos susidėvėjimas grafiškai parodytas pav. 36. Viso eksperimento metu buvo atlikti 93 pilni baterijos įkrovimo-iškrovimo ciklai. Prieš eksperimentą atliekant matavimą be apkrovos baterijos užteko 18min ir 13sek ilgiau, nei po eksperimento atliekant identišką matavimą.



36 pav. Baterijos susidėvėjimas eksperimento metu

3.2. Išvados

- Atliekant eksperimentą su pasiūlytu vėlinančiu baterijos elgsenos modeliu šifravimo algoritmų energijos sąnaudų grafikuose pastebėta jog energijos sąnaudų dinamiškumas atvaizduojamas siaurame 5% procesoriaus apkrovos diapazone (tarp 95% ir 100%);
- Baterijos dinaminės elgsenos įvertinimui platesniame procesoriaus apkrovimo diapazone (35% - 100%) pasiūlyta ir panaudota 3DMark etaloninių testavimų programinė įranga;
- 3DMark etaloninių testavimų programinės įrangos panaudojimas eksperimente leido nustatyti, kad pasiūlyto modelio vėlinimo konstantos yra:
    - Procesoriaus vėlinimas: 340s;
    - Fizinės atminties išskyrimo vėlinimas: 340s;
    - Fizinės atminties atlaisvinimo vėlinimas: 180s;
- Nenustatyta stiprios koreliacijos tarp procesoriaus apkrovos ir energijos sąnaudų. Vėlinimas tarp energiją vartojančių įvykių ir baterijos įkrovos pokyčių apsunkina baterijos įkrovos prognozavimą. Pasiūlytas vėlinimo modelis gali būti naudojamas kaip teorinis pagrindas energijos sąnaudų etaloniniams matavimam atlikti;
- Atlikus eksperimentus su kriptografiniais algoritmais paskaičiavome vidutinę kiekvieno iš algoritmų generuojamą apkrovą centriniam procesoriui:
    - Visų kriptografinių algoritmų generuojama procesoriaus apkrova nenukrenta žemiau 95%;
    - Iš likusių šifrų išsiskiria ECDSA kriptoalgoritmas – vidutinė procesoriaus apkrova $\leq 96\%$;
    - Pilnai apkrauti procesoriaus nesugebėjo ir AES bei Camellia šifrai;
    - Matoma aiški tiesioginė Camellia kripto algoritmo procesoriaus apkrovimo priklausomybė nuo rakto ilgio
- Ilgėjant šifravimo raktui, nuo 512 bitų iki 1024 bitų, asimetriniuose kriptografiniuose algoritmuose šifravimo greitis sulėtėja:
    - ECDSA – 10,2 karto;
    - ElGamal – 7,6 karto;
    - RSA – 5,6 karto;
    - DSA – 2,8 karto;
- Ilgėjant šifravimo raktui, nuo 128 bitų iki 256 bitų, simetriniuose kriptografiniuose algoritmuose šifravimo greičio pokytis yra nuo 0 iki 20 milisekundžių, kas sudaro maksimalų sulėtėjimą 1,25 karto;
- Maišos funkcijų santraukų skaičiavimo greičiai vienam megabaitui informacijos yra:
    - MD5– 60ms;
    - SHA1 – 100ms;
    - SHA2/512 – 350ms;
    - Whirlpool – 910ms;
- Atlikus eksperimento rezultatų analizę pastebėta, kad:
    - ištirtų kriptoalgoritmų darbo greičio ir energijos sąnaudų santykis kinta intervale nuo 0,7 iki 0,84;
    - baterijai baigiant išsikrauti jos likutinė talpa kinta netiesiškai;
    - viso eksperimento metu buvo atlikti 93 pilni baterijos įkrovimo-iškrovimo ciklai, kurie baterijos tarnavimo laiką sumažino 18min ir 13sek.

4. IŠVADOS

- Išanalizuotuose mokslinėse publikacijose, daugelyje atvejų, kriptografinių algoritmų energijos sąnaudos įvertinamos naudojant tikslią laboratorinę matavimo įrangą pateikiant rezultatus mikro džiauliais, kas realiose sąlygose neįvertina sistemų aparatūrinės ir programinės įrangos veiklų kurios įtakoja energijos sąnaudas;
- Informacijos saugos lygiai priklauso nuo naudojamų šifravimo raktų ilgio, todėl sėkmingam energijos sąnaudų tyrimui reikalingas baterijos elgsenos modelio sudarymas ir vartojamos energijos matavimo programinės įrangos sukūrimas veikiančios taikomųjų programų lygmenyje;
- Pasiūlytas baterijos elgsenos modelis kuris sudarytas iš trijų pagrindinių komponentų:
    - *Apkrovos-talpos efektas* stebimas, kai baterija iškraunama pastoviai;
    - *Atsistatymo efektas* stebimas, kaip baterijos likutinės energijos atsistatymas bateriją iškraunant maža srove arba jos visai neiškraunant ilgesnį laiko tarpą;
    - *Programinis energijos sąnaudų planavimas* – procesas apima pavyzdžiui procesoriaus taktinio dažnio ir/arba maitinimo įtampos dinaminį keitimą, periferinių įrenginių atjungimą kurie nėra naudojami, ekrano ryškumo keitimą ir pan.
- Sukurta ir matematine lygtimi aprašyta metrika, kuri paremta pasiūlytu baterijos iškrovimo vėlinimo modeliu ir sukurta programinė įranga baterijos energijos sąnaudų matavimui taikomųjų programų lygmenyje įvertinant sistemos apkrovimą;
- Atliekant eksperimentą su pasiūlytu vėlinančiu baterijos elgsenos modeliu šifravimo algoritmų energijos sąnaudų grafikuose pastebėta jog energijos sąnaudų dinamiškumas atvaizduojamas siaurame procesoriaus apkrovos diapazone nuo 95% iki 100%;
- Baterijos dinaminės elgsenos įvertinimui platesniame procesoriaus apkrovimo diapazone pasiūlyta ir panaudota 3DMark etaloninių testavimų programinė įranga, kurios dėka pavyko pasiekti rezultatą diapazone nuo 35% iki 100%;
- 3DMark etaloninių testavimų programinės įrangos panaudojimas eksperimente leido nustatyti, kad pasiūlyto modelio vėlinimo konstantos yra:
    - procesoriaus vėlinimas: 340s;
    - fizinės atminties išskyrimo vėlinimas: 340s;
    - fizinės atminties atlaisvinimo vėlinimas: 180s;
- Atlikus eksperimento rezultatų analizę pastebėta, kad:
    - ilgėjant šifravimo raktui asimetriniai kriptoalgoritmai sulėtėja nuo 2,8 iki 10,2 karto, simetrinių kriptoalgoritmų maksimalus sulėtėjimas sudaro 1,25 karto;
    - maišos funkcijų santraukų skaičiavimo greičiai vienam megabaitui informacijos kinta nuo 60ms (MD5) iki 910ms (Whirlpool);
    - ištirtų kriptoalgoritmų darbo greičio ir energijos sąnaudų santykis kinta intervale nuo 0,7 iki 0,84;
    - baterijai baigiant išsikrauti jos likutinė talpa kinta netiesiškai;
    - viso eksperimento metu buvo atlikti 93 pilni baterijos įkrovimo-iškrovimo ciklai, kurie baterijos tarnavimo laiką sumažino 18min ir 13sek.

5. LITERATŪRA

[1] R. Damaševičius, J. Toldinas ir G. Grigaravičius, „Modelling Battery Behaviour using Chipset Energy Benchmarking," *Elektronika ir Elektrotechnika,* nr. 1392-1215, 2013.

[2] J. G. Simmons, Contemporary Cryptology: The Science of Information Integrity, Wiley-IEEE Press, 1999.

[3] P. Goutam ir M. Subhamoy, RC4 Stream Cipher and Its Variants, Kolkata, India: CRC Press, 2011.

[4] B. Schneier, Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C, John Wiley & Sons, Inc., 1996.

[5] K. Nyberg, „Perfect nonlinear S-boxes," įtraukta *EUROCRYPT'91*, Brighton, 1991.

[6] A. V. Taddeo ir A. Ferrante, „Run-time Selection of Security Algorithms For Networked Devices," įtraukta *Q2SWinet'09*, 2009.

[7] A. Biryukov ir C. D. Canniere, „Block ciphers and systems of quadratic equations," įtraukta *FSE 2003*, 2003.

[8] K. Aoki, T. Ichikawa, M. Kanda, M. Masui, S. Moriai, J. Nakajima ir T. Tokita, „Specification of Camellia - a 128-bit Block Cipher," Nippon Telegraph and Telephone Corporation, Mitsubishi Electric Corporation, 2001.

[9] National Institute of Standards and Technology, „Report on the Development of the Advanced Encryption Standard (AES)," 2 10 2000. [Tinkle]. Available: http://csrc.nist.gov/archive/aes/round2/r2report.pdf. [Kreiptasi 1 5 2013].

[10] National Institute of Standards and Technology, „NIST Archives - AES Issues," 2000. [Tinkle]. Available: http://csrc.nist.gov/archive/aes/round2/comments/20000523-msmid-2.pdf. [Kreiptasi 1 5 2013].

[11] H. Gilbert ir T. Peyrin, „Super-Sbox Cryptanalysis: Improved Attacks for AES-like permutations," 2009.

[12] A. Bogdanov, D. Khovratovich ir C. Rechberger, „Biclique Cryptanalysis of the Full AES," Microsoft, Redmond, 2011.

[13] Federal Information Processing Standards Publication, „FIPS197 - Specification for the Advanced Encryption Standard (AES)," Federal Information Processing Standards Publication, 2001.

[14] nobody@jpunix.com, „Thank you Bob Anderson," 9 9 1994. [Tinkle]. Available: http://web.archive.org/web/20080404222417/http://cypherpunks.venona.com/date/1994/09/msg00304.html. [Kreiptasi 13 4 2013].

[15] M. C. Henderson, „RC4 Algorithm revealed.," Wimsey Information Services, 14 9 1994. [Tinkle]. Available: https://groups.google.com/group/sci.crypt/msg/e8f2c0f059720aae?dmode=source&output=gplain&noredirect&pli=1. [Kreiptasi 13 4 2013].

[16] European Network of Excellence in Cryptology II, „ECRYPT II Yearly Report on Algorithms and Keysizes (2010-2011)," European Network of Excellence in Cryptology II, 2011.

[17] N. T. Courtois ir J. Pieprzyk, „Cryptanalysis of Block Ciphers with Overdefined Systems of Equations," *Springer-Verlag,* 2002.

[18] R. Anderson, E. Biham ir L. Knudsen, „SERPENT - A Candidate Block Cipher for the Advanced Encryption Standard," 1998. [Tinkle]. Available: http://www.cl.cam.ac.uk/~rja14/serpent.html. [Kreiptasi 8 5 2013].

[19] E. Biham ir A. Shamir, „Differential Cryptanalysis of DES-like Cryptosystems," *Journal of Cryptology,* t. 4, pp. 3-72, 1991.

[20] P. H. Nguyen, H. Wu ir H. Wang, Improving the Algorithm 2 in Multidimensional Linear Cryptanalysis, Springer Link, 2011.

[21] N. Koblitz, „Elliptic Curve Cryptosystems," *Mathematics of Computation,* 1987.

[22] V. S. Miller, „Use of Elliptic Curves in Cryptography," įtraukta *Advances in Cryptology—CRYPTO '85*, Springer-Verlag, 1986.

[23] R. L. Rivest, A. Shamir ir L. Adleman, „A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM,* t. 21, pp. 120-126, 1978.

[24] National Institute of Standards and Technology, „Federal Information Processing Standards Publication 186," NIST, 19 5 1994. [Tinkle]. Available: http://www.itl.nist.gov/fipspubs/fip186.htm. [Kreiptasi 9 5 2013].

[25] A. J. Menezes, P. C. v. Oorschot ir S. A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.

[26] T. ElGamal, „A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory,* %1 t. iš %2IT-31, 1985.

[27] R. L. Rivest, „Network Working Group Request for Comments 1321," 1992. [Tinkle]. Available: http://tools.ietf.org/html/rfc1321. [Kreiptasi 11 5 2013].

[28] R. Rivest, „The MD5 Message-Digest Algorithm," MIT Laboratory for Computer Science, 1992.

[29] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvic ir B. d. Weger, „MD5 considered harmful today: Creating a rogue CA certificate," 30 12 2008. [Tinkle]. Available: http://www.win.tue.nl/hashclash/rogue-ca/. [Kreiptasi 08 04 2013].

[30] National Institute of Standards and Technology, „NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition," National Institute of Standards and Technology, 2 10 2012. [Tinkle]. Available: http://www.nist.gov/itl/csd/sha-100212.cfm. [Kreiptasi 11 5 2013].

[31] S. Manuel, „Classification and Generation of Disturbance Vectors for Collision Attacks against SHA-1," Paris Rocquencourt, 2008.

[32] National Institute of Standards and Technology, „SHS Validation List," National Institute of Standards and Technology, 5 4 2013. [Tinkle]. Available: http://csrc.nist.gov/groups/STM/cavp/documents/shs/shaval.htm. [Kreiptasi 8 4 2013].

[33] „Thank you, Datel.," 24 3 2008. [Tinkle]. Available: http://debugmo.de/2008/03/thank-you-datel/. [Kreiptasi 11 5 2013].

[34] S. Gueron, S. Johnson ir J. Walker, „SHA-512/256," 2010.

[35] V. Rijmen ir P. S. L. M. Barreto, „The WHIRLPOOL Hash Function," 2000. [Tinkle]. Available: http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html. [Kreiptasi 11 5 2013].

[36] P. W. Shor, „Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Journal on Scientific Computing,* nr. 26, 1997.

[37] K. Piotrowski, P. Langendoerfer ir S. Peter, „How Public Key Cryptography Influences Wireless Sensor Node Lifetime," įtraukta *SASN'06*, 2006.

[38] N. R. Potlapally, S. Ravi, A. Raghunathan ir N. K. Jha, „Analyzing the Energy Consumption of Security Protocols," įtraukta *ISLPED'03*, Seoul, 2003.

[39] R. Chandramouli, S. Bapatla, K. P. Subbalakshmi ir R. N. Uma, „Battery Power-Aware Encryption," įtraukta *IEEE International Conference on Communications 7*, 2004.

[40] University of Southern California, „The USC-SIPI Image Database," [Tinkle]. Available: http://sipi.usc.edu/database/database.php?volume=misc&image=12#top. [Kreiptasi 10 1 2013].

[41] A. Jaiantilal, Y. Jiang and S. Mishra, "Modeling CPU Energy Consumption for Evergy Efficient Scheduling," in *GCM'10*, Bangalore, India, 2010.

6. PRIEDAI

Visi magistrinio darbo priedai, kaip ir pats darbas elektroniniame formate, yra įrašyti į kompaktinę plokštelę, kuri prisegta priede 6.3. Priedus sudaro:

- Žurnale „ELEKTRONIKA IR ELEKTROTECHNIKA“ paskelbtas straipsnis magistro darbo tematika;
- Straipsnio priėmimo spausdinimui pažyma;
- Techninės įrangos naudotos eksperimente specifikacijos;
- Kompaktinė plokštelė su visais duomenimis ir dokumentais.

6.1. Straipsnis

http://dx.doi.org/10.5755/j01.eee LEKTRONIKA IR ELEKTROTECHNIKA, ISSN 1392-1215, VOL. XX, NO. X, 20XX

Modelling Battery Behaviour using Chipset Energy Benchmarking

R. Damasevicius[1], J. Toldinas[2], G. Grigaravicius[2]
[1]*Department of Software Engineering, Kaunas University of Technology, Studentu St. 50, LT-51368 Kaunas, Lithuania*
[2]*Department of Computer Science, Kaunas University of Technology, Studentu St. 50, LT-51367 Kaunas, Lithuania*
robertas.damasevicius@ktu.lt

***Abstract*—Despite advances in low power system design, short battery life remains a significant user concern. Effective management of energy resources available on a mobile device requires understanding of the principles of battery behaviour. We propose a time-delay model of a battery, which depends upon three non-linear processes: rate-capacity effect, recovery effect and software scheduling effect. We provide an analysis of the power consumption results using 3DMark'06 chipset benchmarks and demonstrate that a moderate-to-strong correlation between power consumption vs. CPU load, memory allocation and memory release is observed. Finally, we apply our model for chipset energy efficiency profiling and propose a power benchmark metric.**

***Index Terms*—Energy consumption modelling, battery behaviour, time-delay model, power benchmarking.**

I. Introduction

Short battery life was, and still remains, a significant concern for mobile device users. There is a big gap between the energy resources needed by the mobile device and the energy available from the battery; therefore, the average battery life of actively used mobile devices is usually less than two days [1]. Another matter of concern is the ability to predict battery life using available information on the application usage habits, the device's modes of operation and power management schemes so that the user could decide how to use the remaining battery time most effectively. Such prediction is only possible when the behaviour of the battery can be modelled accurately taking all internal (electro-chemical) and external (CPU load, memory usage, display rendering, etc.) factors that influence the *State of Charge* (SoC) of the battery into account.

The most important factors influencing its lifetime are the battery's capacity and the battery's discharge rate. This rate is influenced by three non-linear processes, two of which are determined by the electrochemical properties of the battery. The *rate-capacity effect* is observed when a battery is discharged continuously; a high discharge current causes a battery to provide less energy until the end of its lifetime as compared to a lower discharge current [2]. The *recovery effect* causes the battery capacity to recover to a certain extent during periods of low or no discharge. A third non-linear process that has influence on the state of the battery is *software scheduling* schemes introduced at application [3] or operating system level, which control, e.g., CPU rate and energy consumption level of peripheral devices that are considered non-essential for some applications such as display brightness. The result of these effects is the dependency of the battery lifetime upon battery discharge distribution over time [4], which in turn depends upon user behaviour and usage patterns. Furthermore, the peak power usage can sometimes be a more important factor in determining battery capacity than average power usage [2]. Finally, the effective capacity of a battery depends on the rate at which it is discharged, because the electrochemical actions in the batter take a finite time to complete and they can not follow the battery load instantaneously.

To investigate the influence of the device workload on the battery lifetime, a battery model is needed that includes the above described effects. In electrical engineering, electrical circuit models [5], [6], and electro-chemical models [7] are used. Also high-level analytic and stochastic battery models [4] are available. We treat the battery, CPU and memory data gathered during the execution of the computer benchmarking tests as complex time series. An accurate identification of the dynamics underlying complex time series, is of crucial importance in understanding the corresponding physical process, and in turn affects the subsequent model development [8].

We propose to use time-delay models to study the relationships between CPU load, physical memory usage (allocation, release) and battery charge levels. Time-delay models have been used previously to explain many natural, biological and social processes such as pray-predator systems [9], precipitation patterns [10], business cycles [11].

II. Time-Delay Model of Battery's State of Charge (SoC)

The nonlinear battery system can be modelled by a state equation and an output equation [6]:

Manuscript received April XX, 20XX; accepted April XX, 20XX.

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) + w_k \\ y_k &= g(x_k, u_k) + v_k \end{aligned}, \quad (1)$$

here x is the system state, u is the system's input, w is the unmeasured process noise that affects the system's state, y is the output of the system, v is the measurement noise, and k is the discrete time index of the time series.

The state of the battery can be described by the following time delay differential equation [12]:

$$\dot{x}(t) = f(x(t), x(t-\tau)), \quad (2)$$

here τ is the process lag.

In constructing the battery behaviour model, we use the following assumptions:

1. The state of a battery is determined by two factors: CPU load and physical memory usage.
2. Due to slow electrochemical process, the change of the battery charge level lags behind the changes in CPU usage, memory allocate and release events.

We formulate our time-delay model of the battery's state of charge (SoC) as follows:

$$\dot{c}(t) = f(c(t), c(t-\tau_c), l(t-\tau_l), m_t(t-\tau_a), m_f(t-\tau_r)), \quad (3)$$

here $c(t)$ is the battery's SoC, $l(t)$ is the CPU load process, $m_t(t)$ is the physical memory access process, $m_f(t)$ is the physical memory release process, τ_c is the lag of the battery discharge process, τ_l is the lag of the CPU load process, τ_a is the lag of the memory access process, and τ_r is the lag of the memory release process.

Battery discharge is a continuous process while the events that draw power (CPU calls, memory access and release events) are discrete, and the dependent variable (measured battery charge level) is also discrete. Moreover, the value of CPU load is instantaneous, while the number of memory accesses is only known over a time span. To represent these values as the continuous ones, some smoothing is required.

Let l_t be the CPU load value (in percents) in time t. Let m_t be the amount of used physical memory in time t. Let c_t be the battery charge value (in percents) in time t. Let $h(x)$ be a step function $h(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}$. Physical memory accessed in a time span $(t_0, t_0 + \Delta t)$ is estimated as $m_a(t_0, \Delta t) = \sum_{t=t_0}^{t_0+\Delta t} h(m_t - m_{t-1})$. Physical memory released in a time span $(t_0, t_0 + \Delta t)$ is estimated as $m_r(t_0, \Delta t) = \sum_{t=t_0}^{t_0+\Delta t} h(m_{t-1} - m_t)$.

Let $f_s(X, t_0, w) = \frac{1}{w}\sum_{t=t_0}^{t_0+w-1} x_t$ be the smoothing function (we use moving average, MVA) of the time series $x_t \in X$, and w is the length of a smoothing window. Averaged CPU load over window w_l is $\overline{l(t)} = f_s(l_t, t_0, w_l)$. Averaged physical memory access over window w_a is $\overline{m_t(t)} = f_s(m_t, t_0, w_a)$. Averaged physical memory release over window w_r is $\overline{m_f(t)} = f_s(m_f, t_0, w_r)$. Averaged battery charge value over window w_c is $\overline{c(t)} = f_s(c_t, t_0, w_c)$.

To analyse the model, we formulate the following hypotheses:

Hypothesis H1. There is no time-delay relationship between the battery charge level and CPU load.

Let $\overline{c(t+\tau)} \square g(\overline{l(t)})$ be a hypothetical functional relationship between $\overline{c(t)}$ and $\overline{l(t)}$, where τ is the lag value. Pearson correlation of $\overline{c(t)}$ and $\overline{l(t)}$ is $\rho_{c,l} = \max_{\tau, w_c, w_l} \rho(\overline{c(t+\tau)}, \overline{l(t)})$, where w_c and w_l are smoothing parameters of $\overline{c(t)}$ and $\overline{l(t)}$.

Hypothesis H2. There is no time-delay relationship between the battery charge level and memory access events.

Let $\overline{c(t+\tau)} \square g(\overline{m_t(t)})$ be a hypothetical functional relationship between $\overline{c(t)}$ and $\overline{m_t(t)}$. Pearson correlation of $\overline{c(t)}$ and $\overline{m_t(t)}$ is $\rho_{c,m_t} = \max_{\tau, w_c, w_a} \rho(\overline{c(t+\tau)}, \overline{m_a(t)})$, where w_c and w_a are smoothing parameters of $\overline{c(t)}$ and $\overline{m_t(t)}$.

Hypothesis H3. There is no time-delay relationship between the battery charge level and memory release events.

Let $\overline{c(t+\tau)} \square g(\overline{m_f(t)})$ be a hypothetical functional relationship between $\overline{c(t)}$ and $\overline{m_f(t)}$. Pearson correlation of $\overline{c(t)}$ and $\overline{m_f(t)}$ is $\rho_{c,m_f} = \max_{\tau, w_c, w_r} \rho(\overline{c(t+\tau)}, \overline{m_r(t)})$, where w_c and w_a are smoothing parameters of $\overline{c(t)}$ and $\overline{m_f(t)}$.

III. APPLICATION OF THE TIME-DELAY MODEL FOR POWER BENCHMARKING

Energy efficiency is a critical design factor on a battery-powered mobile device. Typically, computer benchmarks are used to evaluate the performance of a computer by performing a strictly defined set of operations and returning some numerical result (a metric) describing how well the tested computer performed. Running the same benchmark test on multiple computers allows the computers to be compared with respect to their performance.

To evaluate energy efficiency of a device, power benchmarks [13] are used. Known examples of such benchmarks include SPECpower_ssj2008 [14] benchmark for measuring the performance and energy consumption of a system running Java-based workloads, TPC-Energy [15] metric and its extensions such as the dynamic weighted energy-efficiency benchmark (DWEE) [16].

Here we propose a power benchmarking metric based on the time-delay model of the battery's SoC as follows (the smaller value of the metric is the better one):

$$P_b = \frac{s_c}{s_{perf}} \bullet \left(c(t_{start}) - \frac{\sum_{t=t_{start}}^{t_{end}} c(t)}{\Delta t(t_{end} - t_{start})} \right), \qquad (4)$$

here t_{start} is start time of benchmark run, t_{end} is time at the end of benchmark run, Δt is sampling period, s_{perf} is the value of the benchmark performance score, and s_c is a scaling constant introduced for usability purposes and is equal to 1000, if $c(t)$ is evaluated in percents.

IV. Case Study and Experimental Results

We use the Futuremark® 3DMark®06 version 1.2.0 benchmark, which provides tests for testing the DirectX 9 gaming performance of Windows PCs and devices, graphics, CPU and GPU feature tests. We registered battery charge level, CPU load and free physical memory every 1 s starting from the fully charged battery. The experiments were performed on Hewlett-Packard F.23 laptop PC running Windows 7 Ultimate 32 bit OS on Intel® Core Duo T2250 1.73 GHz CPU, 4GB DDR2 RAM 265.4 MHz, Mobile Intel® 945 Express Chipset, i945GM 400 MHz GPU with 8MB internal DDR2 memory. We used the measurement methodology already described in [3]. The results of measurements are presented in Figs. 1-3. Fig.1 shows the battery charge level, Fig. 2 shows CPU load and Fig. 3 shows the free physical memory in a computer registered during 9 consecutive runs of the standard 3DMark®06 test.

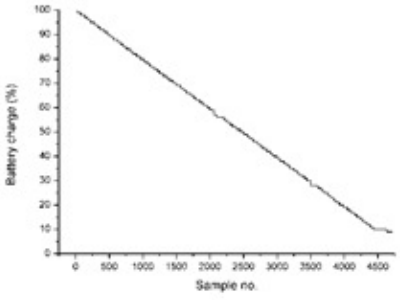


Fig 1. Battery charge level during a standard 3DMark®06 test.

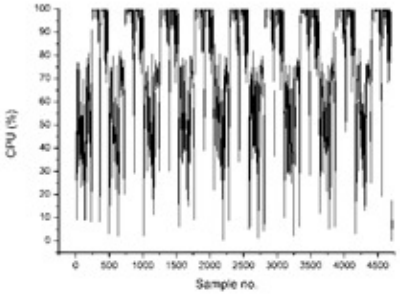


Fig 2. CPU load during a standard 3DMark®06 test.

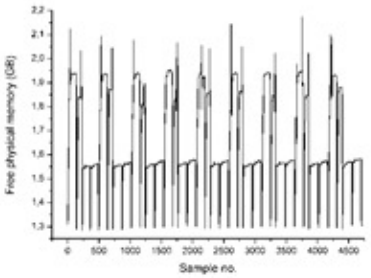


Fig 3. Physical memory use during a standard 3DMark®06 test.

The time-delay model of the battery's SoC is evaluated in Fig. 4-6. Fig. 4 shows the relationship of lagged (lag = 340 s) MVA (window length is 60 s) of the CPU load value vs. MVA (window length is 150 s) of power consumption. The lag value corresponds to the largest value of the Pearson correlation (see Table I). The battery has two distinct states: 1) high-load (CPU load > 70%), mean MVA of consumed power is 1.20%; 2) low-load (CPU load < 70%), mean MVA of consumed power is 1.07%.

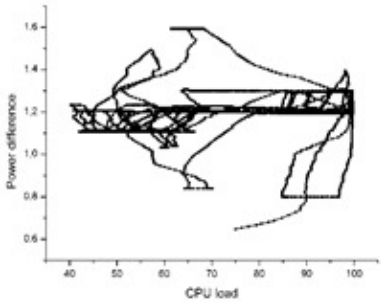


Fig 4. MVA of CPU load vs. MVA of power consumption.

Fig. 5 shows the relationship of the lagged (lag = 520 s) MVA (window length is 180 s) of the memory allocation value vs. MVA (window length is 150 s) of the power consumption. The plot demonstrates the complex multi-state behaviour of power consumption process for high (> 256 MB) memory allocation events.

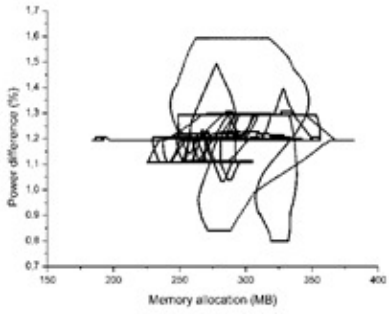


Fig. 5. MVA of physical memory allocation vs. MVA of power consumption.

Fig. 6 shows the relationship of the lagged (lag = 180 s) moving average (window length is 480 s) of the memory allocation value vs. moving average (window length is 150s) of the power consumption. The plot is similar to Fig. 5: for high (> 256 MB) memory release events, power consumption becomes less predictable.

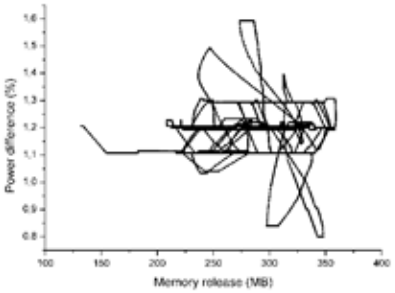


Fig. 6. MVA of physical memory release vs. MVA of power consumption.

We evaluate the hypotheses using the two-tailed test of the Pearson correlation coefficients. The correlations coefficients are significant at $p = 0.001$ ($\rho > 0.104$ for $N = 1000$). The relationship is usually considered strong, if $\rho > 0.4$, and moderate, if $\rho > 0.3$. The results are summarized in Table I. Based on these results, we reject the hypotheses H1, H2 and H3, and confirm the dependency of the battery's charge level upon time-delayed values of CPU load and memory access and release values.

TABLE I. PEARSON CORRELATION OF MVAS OF CPU LOAD, PHYSICAL MEMORY ALLOCATION AND RELEASE VS. MVA OF POWER CONSUMPTION.

Process	MVA window size, s	Lag value, s	Pearson correlation	Hypothesis outcome
CPU load	60	340	0,603	Rejected
Memory allocation	150	520	0,360	Rejected
Memory release	480	180	0,590	Rejected

To validate the power benchmark metric, we have run the standard 3DMark®06 test 8 times on the HP laptop PC and have averaged the results. We have obtained the 3DMark®06 score value of 224 and the power metric value (Eq. 4) $P_s = 24.0$ (std. deviation = 1.5).

To compare, we have run the benchmark on the Acer Aspire 1800 laptop PC running Windows 7 Professional 32 bit OS on Intel® Pentium4 2.93GHz CPU, 1GB DDR RAM 166MHz, Intel® i915P/i915g chipset, and PA3206U (59Ah, 17V) battery and have obtained the 3DMark®06 test score value of 138 and the power metric value $P_s = 303.9$ (std. deviation = 18.7).

Based on these results we conclude that the Acer Aspire has worse energy efficiency due to higher CPU speed that requires more energy, smaller RAM (which means more physical memory accesses are required) and worse chipset (including GPU) characteristics.

V. CONCLUSIONS

1) Battery is a time-delay system with the experimentally-determined lag values of CPU is 340 s, physical memory allocation lag is 340 s and release lag is 180 s.

2) There is no strong correlation between CPU load and power consumption, however there is a strong correlation between the memory usage and power consumption which corresponds well to long-known observation that memory is the largest consumer of power in modern computers thus confirming the validity of the proposed model.

3) Lag between energy consumption events and battery charge value decreases the predictability of the battery's State of Charge (SoC).

4) Battery-powered computers have the properties of multi-state systems with complex relations between states therefore battery charge forecasting is difficult.

5) The proposed time-delay model of the battery's SoC can be used as a theoretical background for power efficiency benchmarking.

6) We propose a formula for computing the power efficiency metric based on the 3DMark®06 test score and the battery charge level measurement results.

REFERENCES

[1] N. Korhonen, "Predicting mobile device battery life", M.S. thesis, Aalto University, Finland, 2011.

[2] T.L. Martin, "Balancing Batteries, Power, and Performance: System Issues in CPU Speed-Setting for Mobile Computing", Ph.D. dissertation, Carnegie Mellon University, USA, 1999.

[3] J. Toldinas, V. Stuikys, R. Damasevicius, and G. Ziberkas, "Application-level energy consumption in communication models for handhelds", *Electronics and Electrical Engineering*, vol. 6(49), pp. 73–76, 2009.

[4] M.R. Jongerden, "Model-based energy analysis of battery powered systems", Ph.D. dissertation, University of Twente, the Netherlands, 2010.

[5] S. Gold, "A PSPICE macromodel for lithium-ion batteries", in *Proc. of the 12th Annual Battery Conference on Applications and Advances*, Long Beach, CA, USA, pp. 215–222, 1997.

[6] F. Xuyun; S. Zechang, "A battery model including hysteresis for State-of-Charge estimation in Ni-MH battery," *Vehicle Power and Propulsion Conference, VPPC '08*, pp.1-5, 3-5 Sept. 2008.

[7] M. Doyle, T. F. Fuller, and J. Newman, "Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell", *Journal of the Electrochemical Society*, vol. 140(6), pp. 1526–1533, 1993.

[8] K. Pukenas, K. Muckus, "Algorithm for Detecting Deterministic Chaos in Pseudoperiodic Time Series", *Electronics and Electrical Engineering*, vol. 8(80), pp. 53–56, 2007.

[9] Y. Qu, J. Wei, "Bifurcation analysis in a time-delay model for prey-predator growth with stage-structure", *Nonlinear Dynamics*, vol. 49, pp. 285–294, 2007.

[10] R.B. Smith, "A linear upslope-time- delay model for orographic precipitation", *Journal of Hydrology*, vol. 282, iss. 1-4, pp. 2–2. Elsevier, 2003.

[11] L. Zhou and Y. Li, "A dynamic IS-LM business cycle model with two time delays in capital accumulation equation", *Journal of Computational and Applied Mathematics*, vol. 228(1), pp. 182–187. Elsevier, 2009.

[12] J.E. Forde, "Delay Differential Equation Models in Mathematical Biology", Ph.D. dissertation, University of Michigan, USA, 2005.

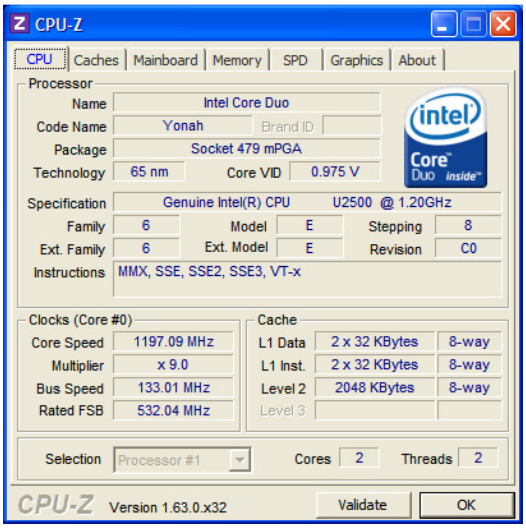
[13] M. Marcu, D. Tudor, S. Fuicu, S. Copil-Crisan, F. Maticu, and M. Micea, "Power efficiency study of multi-threading applications for multi-core mobile systems", *WSEAS Transactions on Computers*, vol. 7(12), pp. 1875–1885, 2008.

[14] Standard Performance Evaluation Corporation (SPEC), SPECpower ssj2008, 2008. [Online]. Available: http://www.spec.org/power_ssj2008

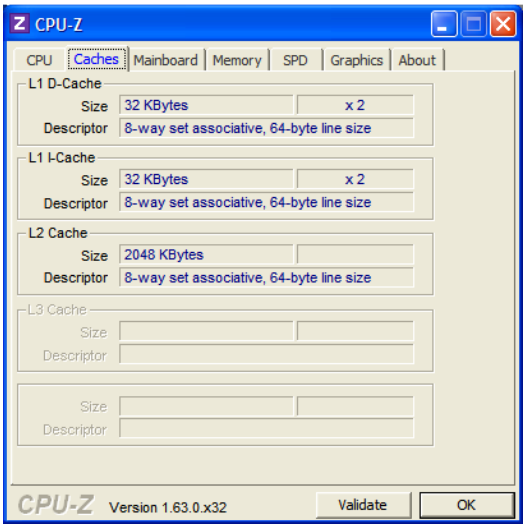
[15] M. Poess, R. Othayoth Nambiar, K. Vaid, J. M. Stephens, Jr., Karl Huppler, and E. Haines, "Energy benchmarks: a detailed analysis", in *Proc. of the 1st Int. Conference on Energy-Efficient Computing and Networking (e-Energy '10)*, ACM, New York, USA, pp. 131–140, 2010.

[16] D. Schall, V. Hoefner, and M. Kern, "Towards an enhanced benchmark advocating energy-efficient systems", in *Proc. of the Third TPC Technology conference on Topics in Performance Evaluation, Measurement and Characterization (TPCTC'11)*, Springer-Verlag, Berlin, Heidelberg, pp. 31–45, 2011.

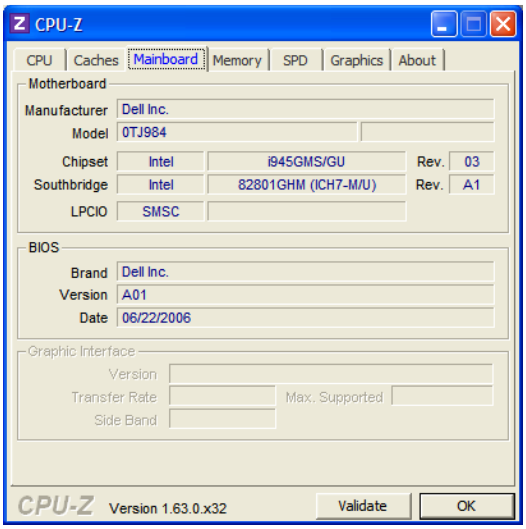
6.2. Techninės įrangos parametrai



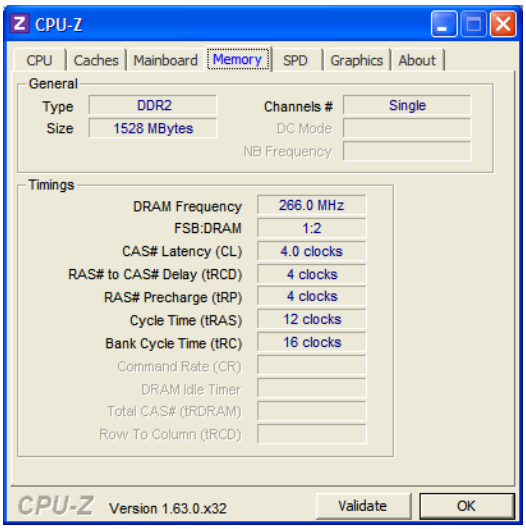
37 pav. CPU-Z rodomos procesoriaus charakteristikos



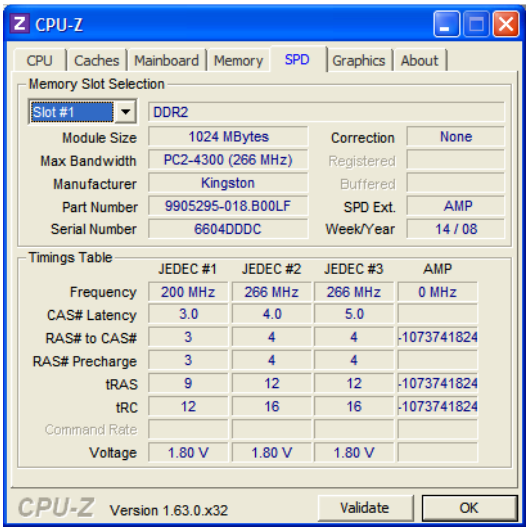
38 pav. CPU-Z rodomos procesoriaus kešo charakteristikos



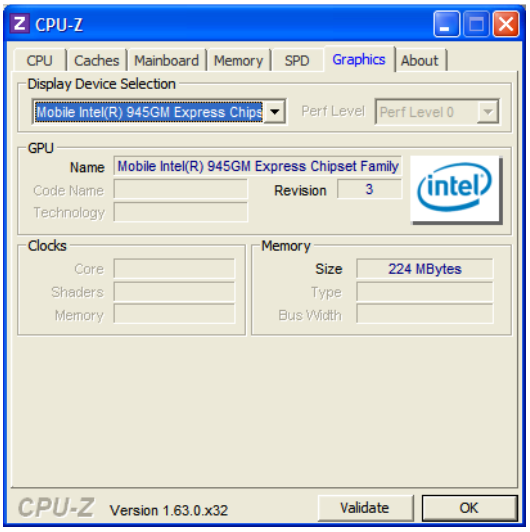
39 pav. CPU-Z rodomos pagrindinės plokštės charakteristikos



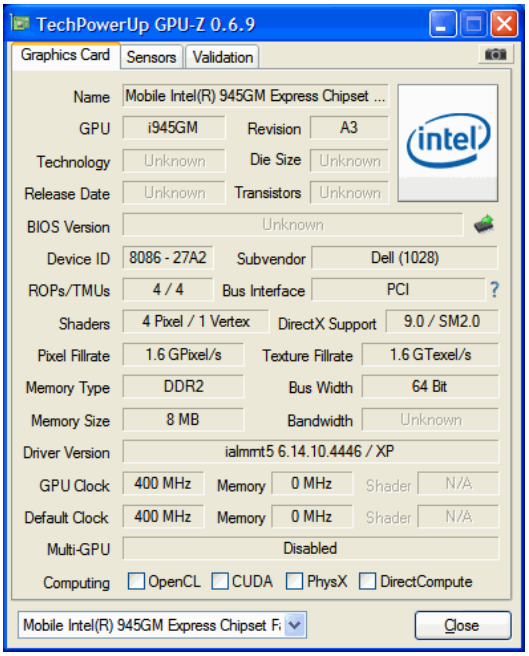
40 pav. CPU-Z rodomos operatyviosios atminties charakteristikos



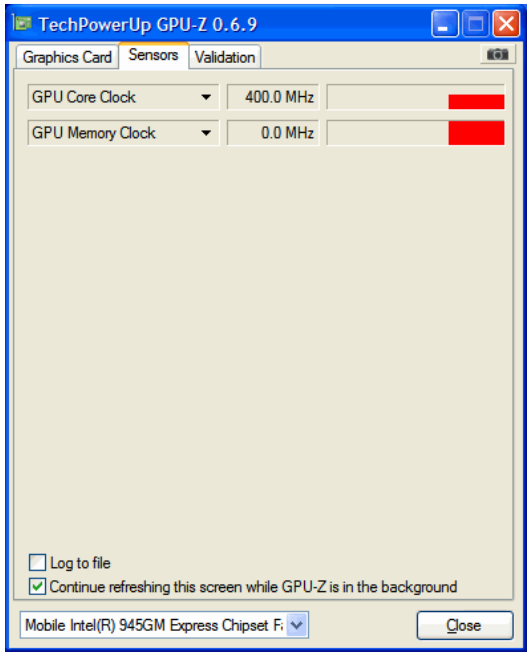
41 pav. CPU-Z rodomos operatyviosios atminties modulio charakteristikos



42 pav. CPU-Z rodomos grafinės plokštės charakteristikos



43 pav. GPU-Z rodomos grafinės plokštės charakteristikos



44 pav. GPU-Z rodomos grafinės plokštės daviklių reikšmės

6.3. Eksperimento duomenys ir dokumentai