KAUNO TECHNOLOGIJOS UNIVERSITETAS
MATEMATIKOS IR GAMTOS MOKSLŲ FAKULTETAS

Lukas Litvinas

MATRICINIŲ LYGČIŲ VIRŠ BAIGTINIO ŽIEDO SPRENDINIŲ ANALIZĖ

Baigiamasis magistro projektas

Vadovas
Lekt. dr. Kęstutis Lukšys

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS
MATEMATIKOS IR GAMTOS MOKSLŲ FAKULTETAS

MATRICINIŲ LYGČIŲ VIRŠ BAIGTINIO ŽIEDO SPRENDINIŲ ANALIZĖ

Baigiamasis magistro projektas
Taikomoji matematika (kodas 621G10003)

Vadovas
Lekt. dr. Kęstutis Lukšys
2017 06 05

Recenzentas
Prof. dr. Eligijus Sakalauskas
2017 06 05

Projektą atliko
Lukas Litvinas
2017 06 05

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Matematikos ir gamtos mokslų fakultetas
(Fakultetas)

Lukas Litvinas
(Studento vardas, pavardė)

Taikomoji matematika (621G10003)
(Studijų programos pavadinimas, kodas)

„Matricinių lygčių virš baigtinio žiedo sprendinių analizė“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 ____ m. _____ _____ d.
Kaunas

Patvirtinu, kad mano, **Luko Litvino**, baigiamasis projektas tema „Matricinių lygčių virš baigtinio žiedo sprendinių analizė“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

_____ _____
(vardą ir pavardę įrašyti ranka) *(parašas)*

Litvinas, L. Analysis of Matrix Equations Over the Finite Ring: *Master's Degree in Applied Mathematics* final thesis / supervisor lect. dr. Kęstutis Lukšys; Department of Applied Mathematics, Faculty of Mathematics and Natural Sciences, Kaunas University of Technology. – Kaunas, 2017. 58 p.

SUMMARY

Cryptanalysis might seem to be a natural opposition of cryptography but very commonly these two subjects are complementary. Good understanding of cryptanalysis helps to create better cryptographic methods. The most straightforward method of secret key cryptosystem attack is brute-force search. But there is also the easiest way to protect against it by choosing the large enough key space. Other ways of breaking the cryptosystems are more complex and by proving that they are not more efficient than brute-force attacks may prove the security of the cryptosystem.

In this paper we analyze matrix equations over the finite ring. These equations were derived from the simple matrix cipher relating unknown secret key with given plaintext and corresponding ciphertext. We had few goals: to check whether there is more than one solution of the key with fixed plaintext and cypher text information by brute-force search and to create a more efficient, algebraic equations solving algorithm.

Simple matrix cipher (SMC) – is a typical symmetric secret key cryptosystem and has three main steps: key generation, encryption, decryption. SMC works in multiplicative semigroup of square matrices over the finite ring. By performing known-plaintext attack with matrix brute-force search algorithm in this system, we found that when increasing the plaintext and ciphertext (number of input and output matrices) the numer of solutions decreses. It is clear, that using 7 input and output matrices pairs with each of checked rings, we reach such limit, that solutions number stabilizes. Moreover, we proved the number of true solutions theoretically. Number of true solutions is always equal to the number of inverse elements in the ring.

Alternatively, it is possible to find SMC keys in algebraic way, solving nonlinear systems of equations. Theoretically this leads us to quite hard problem of solving systems of multivariate quadratic (MQ) equations. The associated MQ-problem is known to be NP-complete. Fortunately, linearization methods to solve such systems effectively exists and we found the true solutions with an algebraic equations solving algorithm.

Algebraic equations solving algorithm had more constrains and chosen-plaintext attack was performed over the finite field, but such algorithm is considerably more effective than matrix brute-force search.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

ĮVADAS

Jau nuo seno kriptografija naudota apsaugoti itin slaptas bendravimo formas – šnipų ir diplomatų siunčiamus pranešimus ir panašiai. Modernioji kriptografija – tai mokslo šaka, sprendžianti elektroninės informacijos saugos problemas. Ji plačiai naudojama ypač slaptai informacijai saugoti bei ją siųsti atvirais tinklais, kaip internetas.

Dauguma kriptografinių algoritmų buvo sukurti remiantis tam tikrais skaičių teorijos arba algebros dėsniais. Nuosekliai kaupėme reikalingas teorines žinias. Suformulavome pagrindinį baigiamojo projekto tikslą – atlikti paprasto matricų šifro sprendinių analizę skirtingais metodais ir įvertinti jų efektyvumą. Tikslui pasiekti išskyrėme pagrindinius tarpinius uždavinius:

- Patikrinti, ar paprasto matricų šifro suformuotos matricinės lygtys virš baigtinio žiedo turi daugiau negu vieną sprendinį;
- Atlikti žinomos tekstogramos ataką matricų visiško perrinkimo algoritmu ir nustatyti, kaip kinta sprendinių skaičius didinant tekstogramas ir šifrogramas;
- Patikrinti, ar sprendinių skaičius matricinėse lygtyse virš baigtinio žiedo priklauso nuo pačių įvesties ir išvesties matricų reikšmių;
- Atlikti sprendinių skaičiaus teorinį pagrindimą;
- Išsiaiškinti, ar vykdant pasirinktos tekstogramos ataką, paprasto matricų šifro raktus atitinkančius sprendinius įmanoma surasti algebriniu būdu;
- Palyginti algebrinių lygčių sprendimo ir matricų visiško perrinkimo algoritmų efektyvumą atliekant matricinių lygčių sprendinių analizę su antros ir trečios eilės kvadratinėmis matricomis.

Norint pasiekti pagrindinį tikslą ir atlikti išvardintus uždavinius reikalingos bendros matematinės ir kriptografinės žinios bei geras algebrinių struktūrų išmanymas.

Darbe pirmiausiai apžvelgiama reikalinga darbui atlikti teorija, susipažįstama su svarbiausiomis matematinėmis savokomis ir apibrėžimais. Tiriamojoje dalyje pateikiami pagrindiniai metodai, reikalingi atlikti matricinių lygčių virš baigtinio žiedo sprendinių analizę. Sprendinių analizės skyriuje išsamiai aprašomi atlikti tyrimai pasirinktais metodais ir palyginamas jų efektyvumas. Taip pat apžvelgiama programinė realizacija. Visiško perrinkimo ir algebrinių lygčių sprendimo algoritmai realizuojami laisvai pasirinktomis techninėmis ir programinėmis priemonėmis. Pagrindinis įrankis – *Visual Studio*. Galiausiai atliekama diskusija, formuluojamos išvados ir rekomendacijos.

Gebėjimas kurti darbe aprašomus kriptoanalizės metodus yra aktualus saugesnių kriptografinių algoritmų kūrimui. Paprasto matricų šifro užšifravimo veiksmai yra panašūs į matricinio laipsnio funkciją, naudojamą matricinio laipsnio šifre, todėl sukurtus kriptoanalizės metodus galima pritaikyti ir matricinio laipsnio šifro sprendinių nustatymui.

1. TEORINĖ DALIS

1.1. PAGRINDINIAI TEORINIAI ASPEKTAI

1.1.1 BENDRIEJI ŠIFRAVIMO BŪDAI

Mokslas apie informacijos slėpimą vadinamas kriptologija. Jis susideda iš dviejų labai plačių mokslo sričių: kriptografijos ir kriptoanalizės. Daugelio įvairių kriptosistemų ir protokolų kūrimą ir pritaikymą nagrinėja kriptografija, o tuo tarpu kriptoanalizė yra skirta šių algoritmų paslėptos informacijos suradimui arba klastojimui, nežinant pačios pradinės paslapties. Visų iki šiol sukurtų ir naujų kriptografinių algoritmų kūrimas apima abi šias mokslo sritis. Taigi, jeigu kuriame naują kriptografinį algoritmą, visuomet privalome jį ištirti ir vėliau įrodyti, kad jis saugus prieš visas žinomas kriptoanalizės atakas (Sakalauskas, Listopadskis ir Dosinas, 2008).

Pradinė informacija, kurią norima kriptografiškai apsaugoti yra vadinama pradiniu tekstu (angl. plaintext). Užšifravimas – tai procesas, kurio metu pradinis tekstas yra paslepiamas arba paverčiamas į tam tikrą neskaitomą formą, vadinamą šifruotu tekstu (angl. ciphertext). Jis kartais vadinamas kriptograma (angl. cryptogram). Iššifravimas – tai atvirkštinis procesas, kai iš šifruoto teksto atkuriamas pradinis. Šifras tai algoritmas, skirtas šifravimui ir iššifravimui. Dažniausiai bet koks šifras valdomas raktu – slapta informacijos detale, kuri turi įtakos šifruoto teksto sukūrimui. Kriptografinis protokolas nuosekliai apibrėžia kaip šifrai ir kiti kriptografiniai primityvai turi būti naudojami siekiant tam tikrų rezultatų. Tiriamų protokolų, šifrų, raktų valdymo bei vartotojo vykdymų veiksmų visuma sudaro kriptosistemą (Menezes, 2005; Sakalauskas ir kt., 2008).

Skirtingas kriptosistemas, skirtas šifravimui, galima suskirstyti į dvi pagrindines rūšis: simetrines ir asimetrines. Trumpai pristatysime šių klasių skirtumus. Įvairiose simetrinėse kriptografinėse sistemose, duomenų užšifravimui ir iššifravimui yra naudojamas vienas ir tas pats slaptasis raktas. Priešingai, kalbant apie asimetrines sistemas, čia visuomet yra slaptųjų raktų pora – privatusis ir viešasis, kurie pagal tam tikrus reikalavimus yra matematiškai susiję tarpusavyje. Viešasis raktas gali būti žinomas visiškai bet kokiam subjektui, o privatusis raktas – slaptas. Daug kas priklauso ir nuo pačios kriptosistemos paskirties. Įvairi informacija gali būti užšifruojama ir iššifruojama su skirtingais raktais (Sakalauskas ir kt., 2008; Lukšys, 2013)

Magistro baigiamajame projekte plačiau nagrinėsime simetrines šifravimo sistemas. Įvairiais būdais nagrinėjant sukurtus algoritmus, visada laikysime, kad galioja principas – kriptografinė sistema privalo būti saugi, net jei apie ją yra žinoma viskas, išskyrus jos slaptąjį raktą. Tai vadinama Kerkhofo principu. Šis principas leidžia plačiai ištirti naujų algoritmų saugumą, dėl to jis bus reikalaujamas ir darbo eigoje (Sakalauskas ir kt., 2008; Lukšys, 2013).

1.1.2 KRIPTOANALIZĖS APŽVALGA

Kriptoanalizė – tai tokia kriptologijos sritis, kurioje tiriami įvairiais būdais užšifruotos informacijos atkūrimo metodai. Nežinoma slapta pradinė informacija, kuri yra užšifruota tam tikru šifru. Nereikėtų pamiršti, jog kriptoanalizė taikoma ir kituose su kriptografija susijusiuose protokoluose, t. y. ne tik šifravimo sistemose. Unikalių atakų tikslai gali būti skirtingi priklausomai nuo srities (Lukšys, 2013). Šiame darbe labiau koncentruosimės į kriptoanalizę, susijusią su simetrinėmis sistemomis.

Visas žinomas kriptografines atakas potencialaus kenkėjo atžvilgiu galima išskirstyti į keletą pagrindinių tipų (Oppliger, 2005), kuriuos pateikiame:

- Pagal kenkėjo galimybes, t. y. pagal skaičiavimo pajėgumus ir pagal tai, kokią informaciją kenkėjas jau turi.
- Pagal kenkėjo tikslus, t. y. pagal tai, ko jis siekia. Paprastai kenkėjo tikslas – paslapties atskleidimas, tačiau kartais tikslas gali būti konkretus, pavyzdžiui, kenkėjas gali siekti atkurti raktą ar iššifruoti šifrogramą.

Kalbant apie kenkėjo skaičiavimo pajėgumus, dažniausiai išskiriamos tokios atakos (van Tilborg, 2005):

- Šifrogramos ataka (angl. ciphertext-only attack): kenkėjas bando „nulaužti“ kriptosistemą, perimdamas šifruotus pranešimus. Jis nieko nežino nei apie tekstogramas, nei apie raktą.
- Žinomos tekstogramos ataka (angl. known-plaintext attack): kenkėjas gauna šifrogramas atitinkančias tekstogramas (su nežinomu raktu) ir kriptosistemą bando „nulaužti“ naudodamasis šia papildoma informacija.
- Pasirinktos tekstogramos ataka (angl. chosen-plaintext attack): kenkėjas turi prieigą prie šifravimo sistemos ir gali užšifruoti norimas tekstogramas (su nežinomu raktu) bei gauti atitinkamas šifrogramas. Kriptosistemą mėginama „nulaužti“ bandant įvairius derinius.
- Pasirinktos šifrogramos ataka (angl. chosen-ciphertext attack): kenkėjas turi prieigą prie šifravimo sistemos ir gali iššifruoti bet kokias pasirinktas šifrogramas (su nežinomu raktu) bei gauti atitinkamas tekstogramas. Kriptosistemą bandoma „nulaužti“ naudojant įvairias gautas tekstogramų ir šifrogramų poras. Siekiama arba iššifruoti kitas šifrogramas, nesinaudojant šifravimo sistema, arba nustatyti šifravimo raktą (Opplinger, 2005; Lukšys, 2013).

Dažniausiai šifravimo sistemose taikomas jau minėtas Kerkhofo dėsnis ir šifravimo algoritmo saugumas priklauso tik nuo rakto slaptumo (Lukšys, 2013).

Tokiu atveju, kai kenkėjas žino, kokia šifravimo sistema yra naudojama, jis gali įvykdyti šifrogramos ataką, bandydamas atspėti šifravimo raktą. Ši ataka gali būti vykdoma lygiagrečiai, t. y. kenkėjas gali lygiagrečiai perrinkti galimus variantus. Ji vadinama visišku raktų perrinkimu.

Akivaizdu, kad šiai atakai įvykdyti nepakanka vien perrinkti visus galimus variantus. Norėdamas sėkmingai ją baigti, kenkėjas turi gebėti nustatyti, ar konkretus perrenkamas raktas yra teisingas, t. y. ar gauta tekstograma yra prasmingas simbolių rinkinys (Sakalauskas, Listopadskis ir Dosinas, 2008).

Visą kriptoanalizę galima skirtstyti į dvi pagrindines rūšis: diferencinę ir algebrinę. Diferencinę kriptoanalizę pasiūlė Bihamas ir Šamiras (Biham ir Shamir, 1991). Šiuo metu tai yra vienas iš efektyviausių kriptoanalizės įrankių prieš standartines blokines simetrines kriptosistemas. Vienas svarbiausių atsparumo šiai atakai matų yra tekstogramų ir šifrogramų porų skaičius, reikalingas kriptoanalizei atlikti. Diferencinė kriptoanalizė tiria, kaip pradiniai įvesties duomenų skirtumai kinta įvairiose šifro operacijose ir šifravimo veiksmuose. Dažniausiai naudojant diferencialinę kriptoanalizę, suformuojama kiekvieno bloko arba kitos netiesinės operacijos lentelė su skirtumų skirstiniu (angl. difference distribution table). Tokios lentelės eilutėse yra visi galimi įvesties skirtumai, o jos stulpeliuose – visi galimi išvesties skirtumai. Kiek įvesties ir išvesties porų atitinka konkrečius skirtumus parodo šios lentelės įrašai. Norint užpildyti skirtumų skirstinio lentelę, pertikrinamos visos galimos įvesties duomenų poros, turinčios konkretų skirtumą (Lukšys, 2013).

Žinoma ir kita kriproanalizės rūšis – algebrinė kriptoanalizė. Vienos pirmųjų algebrinės kriptoanalizės idėjų pasirodė 1983 metais (Schaumüller-Bichl, 1983). Moderniosios kriptologijos pradininką Šanoną galima vadinti pačių pagrindinių idėjų autoriumi. Jis tvirtino, kad labai gero šifro „nulaužimui“ reikia tiek daug pastangų, kiek ir jų reikėtų išspręsti didelės apimties sudėtingų lygčių sistemą (Shannon, 1949). Pati algebrinė kriptoanalizė paremta tuo, kad bet kokį šifrą arba atskiras to šifro sudedamąsias dalis galima apibrėžti algebrinių lygčių sistema, apjungiančia tekstogramų, šifrogramų ir naudojamų raktų duomenis (Meier ir kt., 2004). Sudarius šią lygčių sistemą ir turint vieną ar daugiau įvesties bei šifruotų duomenų porų, galima ieškoti algebrinių lygčių sistemos sprendinio ir mėginti surasti šifravimo raktą. Priešingai nei bet kokios kitos kriptografinės atakos atveju, algebrinei atakai nebūtina turėti labai daug tekstogramų ir šifrogramų porų. Pastarosios atakos atveju pakanka tik vienos ar keleto porų, o tai stipriai padidina algebrinės kriptoanalizės efektyvumą.

Nepaisant to, kad daugelis šifrų pasižymi palyginus gerais kriterijais prieš įprastas atakas, pasirodo, jog juos nesunkiai gali pažeisti algebrinė kriptoanalizė. Tie šifrai realizuojami laipsninėmis šifravimo funkcijomis, tokiomis kaip Gold, Kasami, atvirkštine ir kitomis panašiomis (Cheon ir Lee, 2004; Lukšys, 2013). Jos yra galimai jautrios algebrinei kriptoanalizei, kadangi šios funkcijos gali būti aprašomos palyginus nesudėtingomis algebrinėmis lygtimis.

Visa kriptoanalizė gali pasirodyti kaip visiškai natūrali kriptografijos priešingybė, tačiau dažniausiai šios mokslo šakos yra viena kitą papildančios – geras kriptoanalizės supratimas leidžia kurti saugesnius kriptografijos metodus ar algoritmus. Jeigu kriptosistemoje naudojamas raktas ar slaptažodis, tai dažniausiai yra silpniausia sistemos vieta. Tokiais atvejais atakose gali būti naudojamas visiško perrinkimo arba koks nors kitas panašus algoritmas. Labai sudėtingų raktų analizė yra sunkiai

galima dėl praktinių energijos sąnaudų. Pavyzdžiui, yra žinoma, kad Saulės per metus išspinduliuojamos energijos užtektų tik 187 bitų rakto perrinkimui. Kita vertus, manoma, kad pastarąjį apribojimą leistų apeiti kvantiniai kompiuteriai (Menezes, 2005; Sakalauskas ir kt., 2008).

1.1.3 MATEMATINĖS SĄVOKOS

Šiame skyriuje pateikiami pagrindinių darbe naudojamų matematinių sąvokų apibrėžimai bei detaliau aptariami kiti svarbūs teoriniai aspektai: modulinė atvirkštinė matrica, visiško perrinkimo algoritmas, Hilo šifras ir kt.

Pirmiausiai apžvelgsime svarbiausius algebrinių struktūrų apibrėžimus.

1.1 Apibrėžimas. ***Pusgrupe*** yra vadinama netuščia elementų aibė A su apibrėžta dvinare operacija *, kad $A \times A \to A$, t. y. bet kuriai elementų $a, b \in A$ porai (a, b) priskiriamas tos pačios aibės elementas: $(a, b) \to a * b \in A$ ir tenkinamos aksiomos:

a) aibė A yra uždara operacijos * atžvilgiu;
b) operacija * yra asociatyvioji su visais $a, b, c \in G$, t. y. $(a * b) * c = a * (b * c)$.

Pusgrupė žymima simboliu $\langle A; * \rangle$. Šios pusgrupės elementų skaičius žymimas $|A|$ ir vadinamas aibės A galia. Elementų skaičius gali būti baigtinis arba begalinis (Cauer ir kt., 2000; Dosinas ir Navickas, 2010).

1.2 Apibrėžimas. Operacijos * vietoje, pusgrupėse dažnai naudojamas daugybos ženklas. Tokiu atveju vietoj $a * b$ rašoma $a \cdot b$ ir pusgrupė vadinama ***multiplikacine***, o žymima $\langle A; \cdot \rangle$. Jeigu * yra kita operacija, pavyzdžiui sudėtis, tada naudojamas sudėties ženklas, $\langle A; + \rangle$ ir pusgrupė šiuo atveju vadinama ***adicine*** (Cauer ir kt., 2000; Dosinas ir Navickas, 2010).

1.3 Apibrėžimas. ***Grupe*** yra vadinama netuščia elementų aibė G su tokia apibrėžta dvinare operacija *, kad $G \times G \to G$, t. y. bet kuriai elementų $a, b \in G$ porai (a, b) priskiriamas tos pačios aibės elementas: $(a, b) \to a * b \in G$ ir tenkinamos tokios aksiomos:

a) aibė G yra uždara operacijos * atžvilgiu;
b) operacija * yra asociatyvioji su visais $a, b, c \in G$, t. y. $(a * b) * c = a * (b * c)$;
c) egzistuoja toks vienintelis neutralusis operacijos * elementas e, kad su visais $a \in G$ teisinga tokia lygybė: $a * e = e * a = a$;
d) kiekvienam $a \in G$ egzistuoja toks atvirkštinis (simetriškasis) elementas $a^{-1} \in G$, kad teisinga tokia lygybė: $a * a^{-1} = a^{-1} * a = e$ (Cauer ir kt., 2000; Dosinas ir Navickas, 2010).

Grupę žymėsime simboliu $\langle G; * \rangle$. Šios grupės elementų skaičių žymėsime $|G|$. Jis gali būti baigtinis arba begalinis. Jeigu grupėje $\langle G; * \rangle$ reikalaujame, kad būtų tenkinama komutatyvumo aksioma, t. y. su visais $a, b \in G$ galioja $b * a = a * b$, tuomet sakome, kad grupė yra komutatyvioji arba ***Abelio grupė*** (Cauer ir kt., 2000; Dosinas ir Navickas, 2010).

1.4 Apibrėžimas. Dviveiksmė algebrinė struktūra $\langle A; +; \cdot\rangle$ vadinama ***žiedu***, jeigu struktūra $\langle A; +\rangle$ yra Abelio grupė, o struktūra $\langle A; \cdot\rangle$ – multiplikacinė pusgrupė bei galioja distributyvumo aksioma: su visais $a, b, c \in A$, tenkinamos lygybės $a \cdot (b + c) = a \cdot b + a \cdot c$ ir $(a + b) \cdot c = a \cdot c + b \cdot c$.

Pats paprasčiausias žiedo pavyzdys – sveikųjų skaičių žiedas $\langle \mathbb{Z}; +; \cdot\rangle$ (Lukšys, 2014).

1.5 Apibrėžimas. Žiedą $\langle A; +; \cdot\rangle$ vadiname ***lauku***, jeigu struktūra $\langle A; \cdot\rangle$ yra multiplikacinė Abelio grupė. Akivaizdu, kad bet kuris laukas turi ne mažiau kaip du elementus (Sakalauskas, Listopadskis ir Dosinas, 2008).

1.6 Apibrėžimas. Sveikojo skaičiaus a ***atvirkštinis skaičius*** moduliu p yra toks sveikasis skaičius x, kad galioja $a \cdot x \equiv 1\ (mod\ p)$ (Rosen, 1993).

1.7 Apibrėžimas. Dviejų arba daugiau skaičių ***didžiausias bendras daliklis*** GCD (angl. greatest common divisor) – tai didžiausias skaičius, kuris dalina visus tuos skaičius be liekanos. Jei didžiausias bendras daliklis yra 1, tada šie skaičiai vadinami ***tarpusavyje pirminiais*** (Long, 1972).

1.1 Teorema. Liekanų žiedas $\langle \mathbb{Z}_p; +; \cdot\rangle$ yra laukas, kai p yra pirminis skaičius.

Jeigu p – pirminis, tai kiekvienam nenuliniam elementui $a \in \mathbb{Z}_p$ galime rasti atvirkštinį. Tai išplaukia iš lyginio $a \cdot x \equiv 1\ (mod\ p)$ išsprendžiamumo, nes $a \in \{1, 2, \dots, p-1\}$ ir $(a, p) = 1$. Todėl turime lauką $\langle \mathbb{Z}_p; +; \cdot\rangle$ (Sakalauskas, Listopadskis ir Dosinas, 2008).

1.8 Apibrėžimas. ***Oilerio funkcija*** $\varphi(p)$ yra natūraliųjų skaičių, mažesnių už p ir tarpusavyje pirminių su p, skaičius. Pagal susitarimą: $\varphi(1) = 1$ (Sajavičius, 2003).

Pavyzdžiui, $\varphi(4) = 2$, $\varphi(5) = 4$, $\varphi(6) = 2$ ir t. t.

Toliau pateikiami pagrindiniai darbe naudojami matricų teorijos apibrėžimai.

1.9 Apibrėžimas. n-tosios eilės ***determinantas*** – tiesinės algebros funkcija, kiekvienai kvadratinei $n \times n$ matricai A, priskirianti skaliarinę reikšmę $\det(A)$. Toliau pateikiama determinanto $n \times n$ formulė (Horn ir Johnson, 2013):

$$\det(A) = |A| = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \sum_{i=1}^{n!} (-1)^{p(i)} \cdot a_{1k_{i1}} a_{2k_{i2}} \dots a_{nk_{in}}$$

1.10 Apibrėžimas. Išbraukę determinanto i-tąją eilutę ir j-tąjį stulpelį, kurių susikirtime yra elementas a_{ij}, gausime determinantą, kuris vadinamas elemento a_{ij} ***minoru*** ir žymimas M_{ij} (Campbell, 1971).

1.11 Apibrėžimas. Determinanto elemento a_{ij} ***adjunktu*** vadinamas jo minoras M_{ij}, padaugintas iš daugiklio $(-1)^{i+j}$ ir žymimas $A_{ij} = (-1)^{i+j} \cdot M_{ij}$ (Strang, 2003).

1.12 Apibrėžimas. ***Atvirkštine*** kvadratinės matricos A matrica vadinama matrica A^{-1}, tenkinanti lygybes $A \cdot A^{-1} = A^{-1} \cdot A = I$, kur I – tapačioji kvadratinė matrica. Atvirkštinė matrica išreiškiama:

$$A^{-1} = \frac{1}{|A|} \cdot \begin{pmatrix} A_{11} & A_{21} & \dots & A_{n1} \\ A_{12} & A_{22} & \dots & A_{n2} \\ \dots & \dots & \dots & \dots \\ A_{1n} & A_{2n} & \dots & A_{nn} \end{pmatrix},$$

čia A_{ij}, $i, j = \overline{1, n}$, yra matricos A elementų a_{ij} adjunktai (Strang, 2003).

1.13 Apibrėžimas. Kvadratinė n-tos eilės matrica yra vadinama ***reguliariąja***, jei jos determinantas $|A| \neq 0$ ir ***singuliariąja*** priešingu atveju. Matrica turi atvirkštinę tada ir tik tada, jei ji reguliari (Campbell, 1971).

Papildomai pateikiami bendrieji matematiniai apibrėžimai, ir kai kurie teoriniai aspektai, plačiai naudojami įvairiose matematikos šakose.

Šiek tiek detaliau aprašysime modulinės atvirkštinės matricos skaičiavimą, kuris bus reikalingas matricinėse lygtyse, duomenų iššifravimo veiksmui atlikti. Modulinės atvirkštinės matricos skaičiavimą pristatysime išskirdami 3 pagrindinius žingsnius.

1 žingsnis. Ieškome pasirinktos matricos atvirkštinės. Kiekvienai reguliariai $n \times n$ matmenų matricai

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \tag{1.1}$$

galime rasti atvirkštinę matricą A^{-1}. Pavyzdžiui, matricos

$$A = \begin{pmatrix} 5 & 8 \\ 17 & 3 \end{pmatrix} \tag{1.2}$$

atvirkštinė matrica yra:

$$A^{-1} = \frac{1}{-121} \cdot \begin{pmatrix} 3 & -8 \\ -17 & 5 \end{pmatrix} = 121^{-1} \cdot \begin{pmatrix} -3 & 8 \\ 17 & -5 \end{pmatrix}. \tag{1.3}$$

2 žingsnis. Modulinės atvirkštinės matricos elementai privalo būti iš baigtinio žiedo $\mathbb{Z}_p$ arba lauko, todėl atvirkštinį skaičių rasime laikydamiesi pasirinkto žiedo (arba lauko) aritmetikos. Veiksmai atliekami moduliu p. Randame pirmojo nario atvirkštinį, pagal apibrėžimą $a \cdot x \equiv 1 \, (mod\, p)$. Pavyzdžio atveju, skaičiavimus atliksime žiede $\mathbb{Z}_{26}$. Įsistatome reikšmes ir randame 121^{-1} (sveikojo skaičiaus 121 atvirkštinį x moduliu $p = 26$):

$$121 \cdot x \equiv 1 \, (mod\, 26). \tag{1.4}$$

Iš šios tapatybės gauname, kad $x = 23$.

3 žingsnis. Sujungiame gautus rezultatus. Dauginant abu narius moduliu p gaunama modulinė atvirkštinė matrica. Pavyzdyje, taip pat nesunkiai galime apskaičiuoti modulinę A^{-1}:

$$A^{-1} = 23 \cdot \begin{pmatrix} -3 & 8 \\ 17 & -5 \end{pmatrix} (mod\, 26) = \begin{pmatrix} 9 & 2 \\ 1 & 15 \end{pmatrix}. \tag{1.5}$$

Taigi, gavome modulinę atvirkštinę matricą (Schneier, 1996).

Trumpai susipažindime su teorija, kuri tiria uždavinių sudėtingumą. Žinoma, tai lemia ne uždavinio formuluotės, o jam spręsti skirto algoritmo sudėtingumas, todėl dažnai tokia teorija vadinama algoritmų sudėtingumo teorija. Algoritmo sudėtingumas suprantamas kaip laikas (veiksmų skaičius) ir (arba) atmintinės kiekis, reikalingas kokiam nors uždaviniui išspręsti (Garey ir Johnson, 1979; Sakalauskas ir kt., 2008). Susipažinkime su uždavinių sudėtingumo klasėmis (P, NP, NP-pilnieji).

1.14 Apibrėžimas. Uždavinys L yra polinominio sudėtingumo arba priklauso P (polinominių) uždavinių klasei, jei egzistuoja toks algoritmas A, kad:

a) A išsprendžia uždavinį L;
b) A visada pateikia aiškų atsakymą – taip arba ne;
c) egzistuoja polinominė funkcija p, kad algoritmas, spręsdamas uždavinius su n dydžio pradiniais duomenimis, visada baigia darbą po $p(n)$ arba mažiau žingsnių.

Taigi, P – klasė uždavinių, kuriuos įmanoma išspręsti priimtinu laiku (Sakalauskas ir kt., 2008).

Pateiksime negriežtą apibrėžimą NP sudėtingumo klasės uždaviniams. Taigi, uždavinys L yra priskiriamas NP uždavinių klasei, jei jo sprendinį įmanoma patikrinti priimtinu laiku (Sakalauskas ir kt., 2008).

1.15 Apibrėžimas. Uždavinys L yra vadinamas NP-pilnuoju, jei:

a) $L \in NP$;
b) jei $A \in NP$, tai A per polinominį laiką redukuojamas į L (Sakalauskas ir kt., 2008).

Papildomai apžvelgsime ir visiško perrinkimo algoritmą (angl. brute-force search). Matematikoje ir kompiuterijoje jis žinomas kaip generavimų ir tikrinimų algoritmas. Tai yra apibendrinta problemos sprendimo technika, kuri susideda iš sistemiško visų galimų variantų perrinkimo atsižvelgiant į tai, ar kiekvienas iš tikrinimo atvejų tenkina tam tikras sąlygas (Reynard, 1997).

Visiško perrinkimo algoritmas visuomet suras sprendinį, jeigu jis tiktai egzistuoja, bet šio privalumo kaina yra proporcinga galimų sprendinių skaičiui – kuris daugelyje praktinių atvejų linkęs augti labai greitai, didėjant uždavinio dydžiui. Dėl šios priežasties algoritmas naudojamas tik tokiais atvejais, kai uždavinio dydis yra ribotas arba uždaviniui galima pritaikyti tam tikrus euristinius algoritmus, kurie sumažintų jo sudėtingumą bei galimų sprendinių skaičių. Visiško perrinkimo algoritmas taip pat naudojamas tais atvejais, kai visų įmanomų variantų perrinkimas yra svarbesnis už patį algoritmo vykdymo greitį. Jo taikymai yra plačiai paplitę įvairiose srityse: tikrinti kitų algoritmų veikimą, kuriuose klaidos sukeltų labai rimtas pasekmes; matematikoje – siekiant įrodyti teoremas naudojant kompiuterį; inžinerijoje – lyginant algoritmų pranašumus ar trūkumus; šachmatuose – „aštuonių karalienių“ dėlionei ir kt. (Reynard, 1997).

Kriptografijoje visiško perrinkimo algoritmas siejamas su visų įmanomų raktų perrinkimu (van Tilborg, 2005). Ši strategija atakose gali būti naudojama įvairiems užšifruotiems duomenims atskleisti,

tokiu atveju, kai potencialus įsibrovėlis negali pasinaudoti jokia kriptosistemos silpnybe, kuri jo tikslą padarytų lengvesnį. Raktų ilgis naudojamas užšifravimui apibrėžia visiško perrinkimo algoritmo praktines galimybes. Ilgesnius raktus perrinkti yra žymiai sunkiau nei trumpesnius. Vienas iš šifravimo sistemos stiprumo matų yra kaip ilgai įsibrovėliui teoriškai užtruktų sėkmingai įvykdyti visiško perrinkimo ataką nukreiptą prieš šią sistemą. Toliau pateikiamas 1.1 algoritmas – apibendrintas visiško perrinkimo algoritmas (Reynard, 1997).

1.1 algoritmas. Apibendrintas visiško perrinkimo algoritmas (Reynard, 1997).

1. Uždaviniui P sugeneruojamas pirmas kandidatas į sprendinius c.
2. Patikrinama, ar pirmas kandidatas c yra uždavinio P sprendinys.
3. Pagal leksikografinę tvarką sugeneruojamas kitas kandidatas į sprendinius c.
4. Patikrinama, ar c yra uždavinio P sprendinys.
5. Jeigu jau perrinkti visi kandidatai į sprendinius, algoritmas stabdomas. Priešingu atveju, grįžtama į 3 žingsnį.
6. Išvedami visi uždaviniui tinkami sprendiniai.

1.1.4 HILO ŠIFRAS

Tiriamojoje dalyje pateiksime paprasto matricų šifro aprašymą, kuris turi panašumų su Hilo šifru, todėl trumpai apžvelgsime ir pastarąjį. Klasikinėje kriptografijoje Hilo šifras yra pagrįstas tiesinės algebros idėjomis. 1929 metais jį sukūrė žinomas matematikas Lester S. Hill (Hill, 1929). Kiekviena šio šifro raidė atitinka skaičių moduliu 26. Tam paaiškinti dažnai pateikiama paprasta schema: $A = 0, B = 1, \dots, Z = 25$. Norint užšifruoti žinutę, kiekvienas blokas iš n raidžių (vektorius iš n komponentų) yra padauginamas iš $n \times n$ matmenų reguliarios matricos. Skaičiavimai atliekami moduliu 26. Norint iššifruoti žinutę, kiekvienas blokas (vektorius) yra padauginamas iš modulinės atvirkštinės matricos tai, kuri buvo panaudota užšifravime. Atliekant užšifravimo veiksmus naudojama matrica yra šifro raktas. Reguliari, raktą atitinkanti matrica gali būti pasirenkama atsitiktinai, tačiau visi jos elementai privalo būti iš baigtinio žiedo $\mathbb{Z}_{26}$. Žinoma, tokio tipo šifras gali būti pritaikytas bet kokiai abėcėlei, su bet kokiu kiekiu raidžių. Jeigu abėcėlė turės p raidžių, tuomet skaičiavimai bus atliekami laikantis $\mathbb{Z}_p$ žiedo aritmetikos (Hill, 1929; Sastry ir Samson, 2012).

Pateiksime pavyzdį. Laikykime, kad reguliari raktinė matrica K yra:

$$K = \begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix}. \tag{1.6}$$

Tarkime, kad norime užšifruoti žinutę $HELP$. Tada šis pradinis tekstas gali būti išreikštas tokiu būdu:

$$HELP \rightarrow \begin{pmatrix} H \\ E \end{pmatrix}, \begin{pmatrix} L \\ P \end{pmatrix} \rightarrow \begin{pmatrix} 7 \\ 4 \end{pmatrix}, \begin{pmatrix} 11 \\ 15 \end{pmatrix}. \tag{1.7}$$

Atliekame užšifravimo veiksmus:

$$\begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix} \cdot \begin{pmatrix} 7 \\ 4 \end{pmatrix} (mod\ 26) = \begin{pmatrix} 7 \\ 8 \end{pmatrix}, \\ \begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix} \cdot \begin{pmatrix} 11 \\ 15 \end{pmatrix} (mod\ 26) = \begin{pmatrix} 0 \\ 19 \end{pmatrix}. \tag{1.8}$$

Gaunamas užšifruotas tekstas (paslėpta žinutė):

$$\begin{pmatrix} 7 \\ 8 \end{pmatrix}, \begin{pmatrix} 0 \\ 19 \end{pmatrix} \rightarrow \begin{pmatrix} H \\ I \end{pmatrix}, \begin{pmatrix} A \\ T \end{pmatrix}. \tag{1.9}$$

Norint iššifruoti pradinį paslėptą tekstą, apskaičiuojame modulinę atvirkštinę matricą K^{-1}, laikydamiesi $\mathbb{Z}_{26}$ aritmetikos ir gauname:

$$K^{-1} = \begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix}. \tag{1.10}$$

Atlikame iššifravimo veiksmus:

$$\begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix} \cdot \begin{pmatrix} 7 \\ 8 \end{pmatrix} (mod\ 26) = \begin{pmatrix} 7 \\ 4 \end{pmatrix}, \\ \begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 19 \end{pmatrix} (mod\ 26) = \begin{pmatrix} 11 \\ 15 \end{pmatrix}. \tag{1.11}$$

Taigi, sėkmingai atkūrėme pradinį tekstą:

$$\begin{pmatrix} 7 \\ 4 \end{pmatrix}, \begin{pmatrix} 11 \\ 15 \end{pmatrix} \rightarrow \begin{pmatrix} H \\ E \end{pmatrix}, \begin{pmatrix} L \\ P \end{pmatrix} \rightarrow HELP. \tag{1.12}$$

Deja, Hilo šifras yra pažeidžiamas žinomos tekstogramos atakos dėl savo tiesiškumo. Dėl šios priežasties vertėtų turėti keletą raktinių matricų, kurios galėtų sukurti netiesnį raktinių elementų ryšį bei apsunkintų žinomos tekstogramos ataką (Hill, 1929; Mahmoud ir Chefranov, 2010).

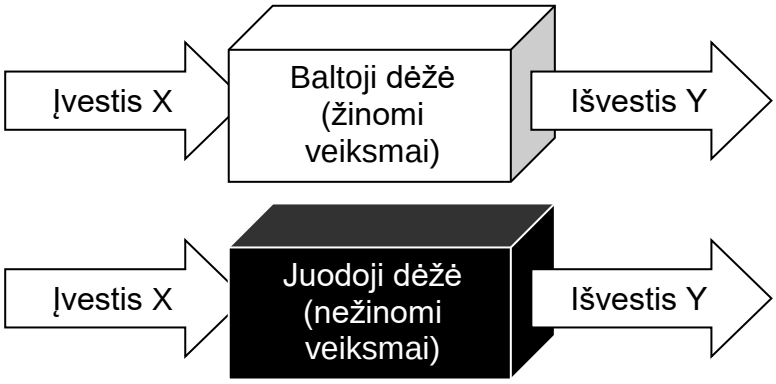
1.1.5 BALTOJI IR JUODOJI DĖŽĖ

Toliau aptarsime baltosios ir juodosios dėžės terminus. Šie terminai į viešumą įžengė 1945 metais. Tuo metu tokiomis dėžėmis buvo laikomos elektroninės schemos arba grandinės, apibūdinamos savo atsako į įvairius signalus. Su šia tema susijusių idėjų labiausiai išvystytą formą 1941 metais pristatė Vilhelmas Kaueris. Nors pats matematikas tokių terminų nevartojo, kiti jo veikla besidomintys mokslininkai tai vadino baltosios arba juodosios dėžės analize. Vėliau ši teorija buvo tobulinama daugelio kitų mokslininkų įvairiose mokslo šakose apytiksliai iki 1960-ųjų metų. Baltosios ir juodosios dėžės terminai laikomi priešingais vienas kitam, dėl to jie yra dažnai lyginami, o jų esmė paaiškinama apibendrintai (Cauer ir kt., 2000; Glanville, 2009).

Moksle, kompiuterijoje ir inžinerijoje baltoji dėžė – tai prietaisas, sistema arba objektas į kurį galima žiūrėti kaip į įvesčių ir išvesčių arba perdavimo charakteristikų visumą, turint bendrą supratimą

apie jos vidinį darbą. Kitaip tariant, jos viduje atliekamų skaičiavimų logika yra žinoma arba „balta“. Kita vertus, nors ir žinoma atliekamų veiksmų logika, tačiau baltosiose dėžėse gali būti nežinoma viena ar kita skaičiavimų metu naudojama detalė (žinomas tik tos detalės panaudojimas). Taigi, baltąja dėže su nežinomais komponentais laikysime darbo metu nagrinėjamas matricines lygtis.

Juodoji dėžė – tai prietaisas, sistema arba objektas į kurį galima žiūrėti kaip į įvesčių ir išvesčių visumą, be jokio supratimo apie jos vidinį darbą. Kitaip tariant, jos vidinis darbas yra neaiškus arba „juodas“ (Glanville, 2009).



1.1 pav. Baltosios ir juodosios dėžės iliustracijos (Glanville, 2009)

Kriptografijoje tokios dėžės naudojamos kuriant ir realizuojant kriptografinius algoritmus; kompiuterių programavime ir programinės įrangos kūrime – patikrinti ar programos išvestis yra tokia, kokios buvo tikimasi; matematiniame modeliavime – kai pagrindinis tikslas duomenų atkartojimas ar kopijavimas; fizikoje – tai sistema, kurios vidinės dalys nėra gerai žinomos ir turi būti išsiaiškintos; aviacijoje juodosios dėžės vardu vadinami skrydžio duomenų registratoriai. Beveik kiekvieną visatos objektą galėtume vadinti baltąja ar juodąja dėže: prietaisą, algoritmą, ar net žmogaus smegenis (Glanville, 2009).

1.2. PROGRAMINĖS ĮRANGOS PASIRINKIMO PAGRINDIMAS

Studijuojant matematinių sričių modulius, universitete naudojama šiuolaikinė matematikos programinė įranga ir daugiausiai paplitę standartiniai programiniai įrankiai (1.1 lentelė). Ataskaitos rengimui naudotas *Microsoft Word*. Kaip pagalbinė priemonė naudotas *Microsoft Excel* – universalus įrankis, su kuriuo galima atlikti labai įvairius skaičiavimus. Darbo metu jis naudotas nustatyti atvirkštinių elementų skaičiui skirtinguose sveikųjų skaičių žieduose, brėžti grafikus bei papildomiems tikrinimams atlikti. Šis įrankis plačiai naudojamas nesudėtingiems matematikos, informatikos, finansų ir kitų panašių sričių uždaviniams spręsti. *Mathcad* naudotas su matricomis atliekamų veiksmų patikrinimui, determinantų apskaičiavimui, tiesinių ir netiesinių lygčių sistemoms spręsti bei kitiems

smulkiems skaičiavimams. Pagrindinę programos realizaciją atlikome *Microsoft Visual Studio* programinėje aplinkoje, kurią pasirinkome dėl jos išmanymo ir tinkamumo. Ši programavimo aplinka dažniausiai naudojama kuriant programas *Windows* sistemai.

1.1 lentelė. Naudota programinė įranga

Logotipas	Pavadinimas	Aprašymas
	*Microsoft Word*	Teksto redagavimo programa, *Microsoft Office* paketo dalis. Naudojama dokumentų, ataskaitų ruošimui ir peržiūrai.
	*Microsoft Excel*	Universali elektroninė skaičiuoklė. *Microsoft Office* paketo dalis. Skirta atlikti labai įvairaus sudėtingumo skaičiavimus, dirbti su duomenų įrašais, grafiškai vaizduoti duomenis.
	*Mathcad*	Matematinių skaičiavimų programa. Ji pasižymi funkcijų gausa, todėl yra universali. Galima atlikti sudėtingus skaičiavimus. Pagrindinis bruožas – lengvai suprantama vartotojo sąsaja.
	*Microsoft Visual Studio*	Integruota programavimo ir kūrimo aplinka. Dažniausiai naudojama kuriant programas, skirtas *Windows* sistemai. Plačiai naudojamas įrankis programinės įrangos kūrimui.

2. TIRIAMOJI DALIS

2.1. PAGRINDINIAI METODAI

2.1.1 PAPRASTAS MATRICŲ ŠIFRAS

Šiame skyriuje pirmiausiai apžvelgsime šifravimo metodo idėją, o vėliau suformuluosime pagrindinius šifro žingsnius. Atliekamų veiksmų logika, įvestis ir išvestis yra žinoma bet kokiam potencialiam kenkėjui, tačiau nežinomos dvi labai svarbios detalės – slaptos, raktus atitinkančios matricos. Dėl šios priežasties potencialus kenkėjas norėdamas surasti raktus atitinkančias matricas susiduria su matricinių lygčių sprendimo problema. Čia galima teigti, kad turima baltoji dėžė, su keletu nežinomų komponentų. Skaičiavimai atliekami matricų pusgrupėse $\langle \mathbb{M}_m(\mathbb{Z}_p); \cdot \rangle$, kur $\mathbb{Z}_p$ – baigtinis sveikųjų skaičių žiedas arba laukas, o $\mathbb{M}_m(\mathbb{Z}_p)$ – matricų aibė, kurios elementai yra apibrėžtos algebrinės struktūros elementai. Pirmiausiai reikia parinkti sveikąjį skaičių p, matricos eilę m, įvesties matricų skaičių n. Parametras n apibrėžia, kiek įvesties matricų naudojama (tekstogramos dydis). Matricos eilė m nusako su kokio dydžio (eilučių ir stulpelių skaičius) kvadratinėmis matricomis

atliekami veiksmai. Sveikasis skaičius p nurodo žiedo arba lauko dydį. Atliekant veiksmus su matricomis, šios sveikųjų skaičių aibės elementai ir bus matricos pozicijas užpildantys elementai. Pusgrupėje apibrėžta viena dvinarė operacija – daugyba, todėl pusgrupė yra multiplikacinė ir sistemos veikimo principas yra paremtas matricų daugyba. Toliau išskiriami ir detaliau paaiškinami pagrindiniai šio šifravimo metodo žingsniai.

Analizuojamą paprastą matricų šifrą (angl. simple matrix cipher), darbe dar vadinsime ir SMC vardu. SMC – tai tipinė simetrinė kriptosistema, kuri turi tris pagrindinius žingsnius: raktų generavimas, užšifravimas ir iššifravimas. Jau išsiaiškinome, kad šifras šifruoja kvadratines matricas, sveikųjų skaičių multiplikacinėse pusgrupėse, virš baigtinio žiedo arba lauko. Šias algebrines struktūras (žiedą arba lauką) apibrėžia parametrų m ir p rinkinys.

Raktų generavimo žingsnis. SMC slaptieji raktai yra dvi reguliariosios matricos A ir B. Jos generuojamos atsitiktinai ir tikrinama, kad jų determinantai būtų nelygūs nuliui: $|A| \neq 0, |B| \neq 0$.

Užšifravimo žingsnis. Pradinis tekstas (angl. plaintext) yra taip pat interpretuojamas kaip matricos: $X \in \mathbb{M}_m(\mathbb{Z}_p)$. Šifrogramą (angl. ciphertext) atitinkančios matricos yra randamos apskaičiuojant sandaugą:

$$A \cdot X \cdot B \ (mod\ p) = Y. \tag{2.1}$$

Iššifravimo žingsnis. Apskaičiuojamos modulinės atvirkštinės matricos A^{-1} ir B^{-1} bei atliekamas iššifravimas:

$$A^{-1} \cdot Y \cdot B^{-1} \ (mod\ p) = X. \tag{2.2}$$

Turint n tekstogramos matricų, galime gauti matricinių lygčių sistemą:

$$A \cdot X_i \cdot B \ (mod\ p) = Y_i, i = 1, \dots n. \tag{2.3}$$

SMC turi panašumų su apibendrintu Hilo šifru (Mahmoud ir Chefranov, 2010; Sastry ir Samson, 2012), bet čia antros raktų matricos panaudojimas sukuria netiesnį raktinių elementų ryšį ir žinomos tekstogramos (angl. known-plaintext) ataka tampa dar labiau komplikuota.

Geresniam atliekamų veiksmų supratimui pateiksime keletą praktinių pavyzdžių, kuriuose atsispindi aprašyti žingsniai ir juose atliekami veiksmai. Pavyzdžiai pateikiami 2.1, 2.2 lentelėse.

2.1 lentelė. Matricų šifravimo pavyzdys su viena X, Y matricų pora

Žingsnis	Atliekami veiksmai
Generavimas	Parenkami sisteminiai parametrai: $m = 2, p = 10 \rightarrow \langle \mathbb{M}_2(\mathbb{Z}_{10}); \cdot \rangle$, nurodomas įvesties matricų skaičius $n = 1$. Pagal sisteminius parametrus sugeneruojamos kenkėjams nežinomos raktinės matricos A ir B: $A = \begin{pmatrix} 2 & 5 \\ 7 & 9 \end{pmatrix}$ ir $B = \begin{pmatrix} 1 & 9 \\ 2 & 9 \end{pmatrix}$, kur $\lvert A \rvert = 3 \neq 0$ ir $\lvert B \rvert = 1 \neq 0$.

Užšifravimas	Sugeneruojama viena įvesties matrica $X_1 = \begin{pmatrix} 0 & 4 \\ 3 & 6 \end{pmatrix}$. Atliekamas užšifravimo veiksmas: $A \cdot X_1 \cdot B \ (mod\ p) = \begin{pmatrix} 2 & 5 \\ 7 & 9 \end{pmatrix} \cdot \begin{pmatrix} 0 & 4 \\ 3 & 6 \end{pmatrix} \cdot \begin{pmatrix} 1 & 9 \\ 2 & 9 \end{pmatrix} (mod\ 10) = \begin{pmatrix} 1 & 7 \\ 1 & 1 \end{pmatrix} = Y_1$. Taigi, gavome 1-mąją X, Y porą.
Iššifravimas	Randamos modulinės atvirkštinės matricos: $A^{-1} = \begin{pmatrix} 3 & 5 \\ 1 & 4 \end{pmatrix}$ ir $B^{-1} = \begin{pmatrix} 9 & 1 \\ 8 & 1 \end{pmatrix}$, atliekamas iššifravimo veiksmas: $A^{-1} \cdot Y_1 \cdot B^{-1} \ (mod\ p) = \begin{pmatrix} 3 & 5 \\ 1 & 4 \end{pmatrix} \cdot \begin{pmatrix} 1 & 7 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 9 & 1 \\ 8 & 1 \end{pmatrix} \ (mod\ 10) = \begin{pmatrix} 0 & 4 \\ 3 & 6 \end{pmatrix} = X_1$. Taigi, duomenis iššifravome.

2.2 lentelė. Matricų šifravimo pavyzdys su keliomis X, Y matricų poromis

Žingsnis	Atliekami veiksmai
Generavimas	Parenkami sisteminiai parametrai: $m = 2, p = 15 \rightarrow \langle \mathbb{M}_2(\mathbb{Z}_{15}); \cdot \rangle$, nurodomas įvesties matricų skaičius $n = 2$. Pagal sisteminius parametrus sugeneruojamos kenkėjams nežinomos raktinės matricos A ir B: $A = \begin{pmatrix} 12 & 14 \\ 8 & 2 \end{pmatrix}$ ir $B = \begin{pmatrix} 6 & 13 \\ 8 & 13 \end{pmatrix}$, kur $\lvert A \rvert = 11 \neq 0$ ir $\lvert B \rvert = 1 \neq 0$.
Užšifravimas	Sugeneruojamos dvi įvesties matricos $X_1 = \begin{pmatrix} 8 & 11 \\ 8 & 14 \end{pmatrix}$ ir $X_2 = \begin{pmatrix} 8 & 1 \\ 4 & 9 \end{pmatrix}$. Atliekami užšifravimo veiksmai: $A \cdot X_1 \cdot B \ (mod\ p) = \begin{pmatrix} 12 & 14 \\ 8 & 2 \end{pmatrix} \cdot \begin{pmatrix} 8 & 11 \\ 8 & 14 \end{pmatrix} \cdot \begin{pmatrix} 6 & 13 \\ 8 & 13 \end{pmatrix} (mod\ 15) = \begin{pmatrix} 2 & 8 \\ 13 & 13 \end{pmatrix} = Y_1$ ir $A \cdot X_2 \cdot B \ (mod\ p) = \begin{pmatrix} 12 & 14 \\ 8 & 2 \end{pmatrix} \cdot \begin{pmatrix} 8 & 1 \\ 4 & 9 \end{pmatrix} \cdot \begin{pmatrix} 6 & 13 \\ 8 & 13 \end{pmatrix} (mod\ 15) = \begin{pmatrix} 6 & 5 \\ 10 & 14 \end{pmatrix} = Y_2$ Taigi, gavome 1-mąją ir 2-ąją X, Y poras.
Iššifravimas	Randamos modulinės atvirkštinės matricos: $A^{-1} = \begin{pmatrix} 1 & 8 \\ 11 & 6 \end{pmatrix}$ ir $B^{-1} = \begin{pmatrix} 7 & 8 \\ 13 & 9 \end{pmatrix}$, atliekami iššifravimo veiksmai: $A^{-1} \cdot Y_1 \cdot B^{-1} \ (mod\ p) = \begin{pmatrix} 1 & 8 \\ 11 & 6 \end{pmatrix} \cdot \begin{pmatrix} 2 & 8 \\ 13 & 13 \end{pmatrix} \cdot \begin{pmatrix} 7 & 8 \\ 13 & 9 \end{pmatrix} \ (mod\ 15) = \begin{pmatrix} 8 & 11 \\ 8 & 14 \end{pmatrix} = X_1$ ir $A^{-1} \cdot Y_2 \cdot B^{-1} \ (mod\ p) = \begin{pmatrix} 1 & 8 \\ 11 & 6 \end{pmatrix} \cdot \begin{pmatrix} 6 & 5 \\ 10 & 14 \end{pmatrix} \cdot \begin{pmatrix} 7 & 8 \\ 13 & 9 \end{pmatrix} \ (mod\ 15) = \begin{pmatrix} 8 & 1 \\ 4 & 9 \end{pmatrix} = X_2$. Taigi, duomenys iššifruoti.

2.1.2 MATRICŲ VISIŠKO PERRINKIMO ALGORITMAS

Pirmiausiai susipažinkime su iteracijų skaičiaus sąvoka. Iteracijų skaičiaus nurodo, kiek kartų prie fiksuotų raktinių matricų A, B bus pilnai įvykdytas visiško perrinkimo algoritmas. Laikysime, kad naujos iteracijos metu kinta tik įvesties ir išvesties matricų X, Y reikšmės.

Trumpam prisiminkime ankstesniame skyriuje nagrinėtus pavyzdžius, pateiktus 2.1 ir 2.2 lentelėse. Jų pirmuosiuose žingsniuose sugeneruojamos slaptos raktinės matricos A ir B, kurios naudojamos matricinėse lygtyse, tačiau potencialiems kenkėjams nėra žinomos. Vėliau atliekami matricų daugybos veiksmai ir priklausomai nuo įvesties matricų X skaičiaus gaunamas tam tikras skaičius X, Y porų. Tuomet iškyla klausimas: ar yra daugiau matricų A', B' porų, su kuriomis būtų galima gauti tas pačias įvesties ir išvesties matricų poras X, Y, kaip ir su raktinėmis A, B? Teigiamai atsakius į šį klausimą, natūraliai kyla dar vienas: o koks tokių A', B' porų skaičius egzistuoja prie tam tikro skaičiaus n matricų X, Y porų bei konkrečių parametrų m ir p?

Tikrinimui naudojamas visiško perrinkimo algoritmas matricų pusgrupėse. Tokią sprendinių paiešką interpretavome, kaip žinomos tekstogramos ataką. Jau susipažinome su apibendrintu visiško perrinkimo algoritmu, taigi paanalizuosime jo pritaikymo galimybes matricinėse lygtyse, matricų multiplikacinėse pusgrupėse $\langle \mathbb{M}_m(\mathbb{Z}_p); \cdot \rangle$ virš baigtinio žiedo. Skaičiavimus atliekant tokio tipo algebrinėse struktūrose, pagal tam tikrą strategiją perrenkamos visos galimos alternatyvių raktinių matricų A' ir B' kombinacijos. Nurodyto sveikųjų skaičių žiedo $\mathbb{Z}_p$ elementai pagal leksikografinę tvarką surašomi į matricų eilutes ir stulpelius. Perrinkimas pradedamas nuo nulinių matricų ir vykdomas tol, kol bus patikrintos visos sąlygas tenkinančios matricos. Galiausiai pagal įvestus parametrus ir gautus rezultatus galimas alternatyvių raktinių matricų skaičiaus nustatymas. Taigi, toliau pateikiamas sukurtas 2.1 algoritmas – visiško perrinkimo algoritmas matricų pusgrupėse.

2.1 algoritmas. Visiško perrinkimo algoritmas matricų pusgrupėse.

1. Įvedami algoritmo sisteminiai parametrai m, p ir n bei sugeneruojamos slaptos raktinės matricos A ir B, kurioms bus ieškoma alternatyvių. Čia galioja: $|A| \neq 0$, $|B| \neq 0$.

2. Inicijuojamos nulinės matricos A', B' ir tinkamų lygčiai išspręsti A', B' porų skaičius prilyginamas nuliui.

3. Atsitiktinai sugeneruojama bent viena įvesties matrica X_i, kur $i = \overline{1,n}$. Naudojant slaptas raktines matricas A ir B, suskaičiuojamos išvesties matricos: $A \cdot X_i \cdot B \ (mod\ p) = Y_i$. Matricos X_i, Y_i pridedamos į joms skirtus X, Y matricų sąrašus.

4. Jei jau perrinktos visos A', algoritmas stabdomas. Kol neperrinktos visos matricos A', pagal leksikografinę tvarką parenkama viena sekanti A', su visomis X_i atliekama matricų daugyba: $A' \cdot X_i \ (mod\ p) = A'X_i$ ir einama į 5 žingsnį.

5. Jei jau perrinktos visos B' grįžtama į 4 žingsnį. Kol neperrinktos visos matricos B', pagal leksikografinę tvarką parenkama viena sekanti B', su visomis $A'X_i$ atliekama matricų daugyba $A'X_i \cdot B' \ (mod\ p) = Y_i'$ ir einama į 6 žingsnį.

6. Jeigu bent viena $Y_i' \neq Y_i$, grįžtama į 5 žingsnį. Priešingu atveju, t. y. jei visos $Y_i' = Y_i$, surastas A', B' matricas pridedame į tinkamų raktinių A', B' matricų sąrašą. Tinkamų lygčiai išspręsti A', B' porų skaičius padidinamas vienetu ir grįžtama į 5 žingsnį.

2.1.3 ALGEBRINIŲ LYGČIŲ SPRENDIMO ALGORITMAS

Šiame skyriuje aprašysime algebrinių lygčių sprendimo algoritmą. Šis algoritmas kuriamas, tuo tikslu, kad galėtume gauti sprendinius, analogiškus visiško perrinkimo algoritmo sprendiniams efektyvesniu būdu. Tiesa, čia matricinės lygtys sprendžiamos tik virš baigtinio lauko $\mathbb{Z}_p$ (p – pirminis skaičius). Algebrinių lygčių sprendimo algoritmu atliekama pasirinktos tekstogramos (angl. chosen-plaintext) ataka. Pirmiausiai pristatysime algoritme atliekamų veiksmų įdėją, o vėliau pateiksime ir pavyzdį, kuris parodys, kad tokiu būdu surasti sprendinį tikrai įmanoma.

Suformuluokime keletą teorinių prielaidų. Į sandaugą $A \cdot X_i \cdot B \ (mod\ p) = Y_i$, kur $i = 1, \dots n$, pažiūrėkime kaip į netiesinių lygčių sistemą, su nežinomais A ir B elementais.

Pažymėkime:

$$A' = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, B' = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \tag{2.4}$$

Šių matricų elementai yra nežinomi. Naudojant slaptąsias A ir B, gaunamos įvesties ir išvesties matricos. Toliau, bendru atveju įvesties (arba tekstogramos) ir išvesties (arba šifrogramos) matricas pažymėsime:

$$X_i = \begin{pmatrix} x_{11}^i & x_{12}^i \\ x_{21}^i & x_{22}^i \end{pmatrix}, Y_i = \begin{pmatrix} y_{11}^i & y_{12}^i \\ y_{21}^i & y_{22}^i \end{pmatrix}. \tag{2.5}$$

Čia tokių X_i, Y_i matricų porų galime sugeneruoti kiek norime. Pagal apibrėžtą matricų šifravimo metodą (2.2), sudarome netiesinių lygčių sistemą:

$$A' \cdot X_i \cdot B' \ (mod\ p) = Y_i, i = 1, \dots n. \tag{2.6}$$

Ji atitinka netiesinių lygčių sistemą, pagal mūsų pažymėjimus:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x_{11}^i & x_{12}^i \\ x_{21}^i & x_{22}^i \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} (mod\ p) = \begin{pmatrix} y_{11}^i & y_{12}^i \\ y_{21}^i & y_{22}^i \end{pmatrix}. \tag{2.7}$$

Atliekame daugybos veiksmus su pirmąja raktine matrica:

$$\begin{pmatrix} a_{11}x_{11}^i + a_{12}x_{21}^i & a_{11}x_{12}^i + a_{12}x_{22}^i \\ a_{21}x_{11}^i + a_{22}x_{21}^i & a_{21}x_{12}^i + a_{22}x_{22}^i \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} (mod\ p) = \begin{pmatrix} y_{11}^i & y_{12}^i \\ y_{21}^i & y_{22}^i \end{pmatrix}. \tag{2.8}$$

Atliekame daugybos veiksmus su antrąja raktine matrica:

$$\begin{pmatrix} (a_{11}x_{11}^i + a_{12}x_{21}^i)b_{11} + (a_{11}x_{12}^i + a_{12}x_{22}^i)b_{21} & (a_{11}x_{11}^i + a_{12}x_{21}^i)b_{12} + (a_{11}x_{12}^i + a_{12}x_{22}^i)b_{22} \\ (a_{21}x_{11}^i + a_{22}x_{21}^i)b_{11} + (a_{21}x_{12}^i + a_{22}x_{22}^i)b_{21} & (a_{21}x_{11}^i + a_{22}x_{21}^i)b_{12} + (a_{21}x_{12}^i + a_{22}x_{22}^i)b_{22} \end{pmatrix} (mod\ p) = \begin{pmatrix} y_{11}^i & y_{12}^i \\ y_{21}^i & y_{22}^i \end{pmatrix} \tag{2.9}$$

Taigi, galime sudaryti netiesinių lygčių sistemas:

$$\begin{cases} (a_{11}x_{11}^i + a_{12}x_{21}^i)b_{11} + (a_{11}x_{12}^i + a_{12}x_{22}^i)b_{21} = y_{11}^i \\ (a_{11}x_{11}^i + a_{12}x_{21}^i)b_{12} + (a_{11}x_{12}^i + a_{12}x_{22}^i)b_{22} = y_{12}^i \\ (a_{21}x_{11}^i + a_{22}x_{21}^i)b_{11} + (a_{21}x_{12}^i + a_{22}x_{22}^i)b_{21} = y_{21}^i \\ (a_{21}x_{11}^i + a_{22}x_{21}^i)b_{12} + (a_{21}x_{12}^i + a_{22}x_{22}^i)b_{22} = y_{22}^i \end{cases} \rightarrow$$

$$\rightarrow \begin{cases} x_{11}^i a_{11}b_{11} + x_{21}^i a_{12}b_{11} + x_{12}^i a_{11}b_{21} + x_{22}^i a_{12}b_{21} - y_{11}^i = 0 \\ x_{11}^i a_{11}b_{12} + x_{21}^i a_{12}b_{12} + x_{12}^i a_{11}b_{22} + x_{22}^i a_{12}b_{22} - y_{12}^i = 0 \\ x_{11}^i a_{21}b_{11} + x_{21}^i a_{22}b_{11} + x_{12}^i a_{21}b_{21} + x_{22}^i a_{22}b_{21} - y_{21}^i = 0 \\ x_{11}^i a_{21}b_{12} + x_{21}^i a_{22}b_{12} + x_{12}^i a_{21}b_{22} + x_{22}^i a_{22}b_{22} - y_{22}^i = 0 \end{cases} \quad (2.10)$$

Iš viso tokių netiesinių lygčių sistemų turėsime tiek, kiek bus naudojama įvesties ir išvesties porų X_i, Y_i. Atlikus aprašytus veiksmus, netiesinių lygčių sistemos turės 8 nežinomuosius (a_{ij}, b_{ij}) (su didesnėmis matricų eilėmis m, nežinomųjų skaičius bus lygus $2m^2$). Kiekviena tekstogramos ir šifrogramos matricų pora duos 4 netiesines lygtis (su didesnėmis m, duos m^2 netiesinių lygčių).

Teoriškai iš to išplaukia palyginus sudėtingas uždavinys, susijęs su kelių kintamųjų (MQ) lygčių sistemų sprendimu (angl. multivariate quadratic). Ši MQ-problema yra žinoma kaip NP-pilnųjų sudėtingumo klasės uždavinys (Garey ir Johnson, 1979). Čia egzistuoja netiesinių lygčių tiesinimo metodai, tam kad tokias sistemas galėtume išspręsti efektyviai (Thomae ir Wolf, 2010). Darbe šiuos metodus pritaikėme ir savo SMC sistemoms (2.10).

Taigi, sudarytas netiesines lygtis galima išspręsti jas pertvarkant. Tada tikslas turėti tik vieną nepriklausomą kintamąjį a_{11}, o likusius išreikšti tik per jį. Norint pasiekti šį tikslą iš (2.10) reikia sukonstruoti 7 tiesiškai nepriklausomas lygtis (su didesnėmis m reikės sukonstruoti $2m^2 - 1$ tiesiškai nepriklausomų lygčių).

Toliau paaiškinsime sprendinių radimą, kai naudojamos antros eilės matricos bei keturios įvesties ir išvesties matricos ($m = 2, n = 4$). Sugeneruojamos slaptos raktinės matricos A, B. Kadangi vykdome pasirinktos tekstogramos ataką, pasirenkame įvesties matricas X_i:

$$X_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, X_2 = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, X_3 = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, X_4 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}. \quad (2.11)$$

Tokias įvesties matricas pasirinkome tam, kad galėtume nesunkiai patikrinti ieškomų sprendinių elementus. Apskaičiuojama išvestis:

$$A \cdot X_1 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^1 & y_{12}^1 \\ y_{21}^1 & y_{22}^1 \end{pmatrix} = Y_1, \quad A \cdot X_2 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^2 & y_{12}^2 \\ y_{21}^2 & y_{22}^2 \end{pmatrix} = Y_2,$$
$$A \cdot X_3 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^3 & y_{12}^3 \\ y_{21}^3 & y_{22}^3 \end{pmatrix} = Y_3, \quad A \cdot X_4 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^4 & y_{12}^4 \\ y_{21}^4 & y_{22}^4 \end{pmatrix} = Y_4. \quad (2.12)$$

Pagal (2.4) pažymime A', B' ir gauname:

$$A' \cdot X_1 \cdot B' \ (mod\ p) = Y_1, \rightarrow \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} \\ a_{21}b_{11} & a_{21}b_{12} \end{pmatrix} = \begin{pmatrix} y_{11}^1 & y_{12}^1 \\ y_{21}^1 & y_{22}^1 \end{pmatrix}, \quad (2.13)$$

$$A' \cdot X_2 \cdot B' \ (mod\ p) = Y_2, \rightarrow \begin{pmatrix} a_{11}b_{21} & a_{11}b_{22} \\ a_{21}b_{21} & a_{21}b_{22} \end{pmatrix} = \begin{pmatrix} y_{11}^2 & y_{12}^2 \\ y_{21}^2 & y_{22}^2 \end{pmatrix},$$

$$A' \cdot X_3 \cdot B' \ (mod\ p) = Y_3, \rightarrow \begin{pmatrix} a_{12}b_{11} & a_{12}b_{12} \\ a_{22}b_{11} & a_{22}b_{12} \end{pmatrix} = \begin{pmatrix} y_{11}^3 & y_{12}^3 \\ y_{21}^3 & y_{22}^3 \end{pmatrix},$$

$$A' \cdot X_4 \cdot B' \ (mod\ p) = Y_4, \rightarrow \begin{pmatrix} a_{12}b_{21} & a_{12}b_{22} \\ a_{22}b_{21} & a_{22}b_{22} \end{pmatrix} = \begin{pmatrix} y_{11}^4 & y_{12}^4 \\ y_{21}^4 & y_{22}^4 \end{pmatrix}.$$

Įvairiais būdais pertvarkome netiesines lygtis, kad jose liktų tik vienas nepriklausomas kintamasis a_{11}, o visus kitus galėtume išreikšti per jį. Čia kiekvienam kintamajam išsireikšti yra daugiau nei vienas būdas (tačiau mums reikalingas tik vienas). Skaičiavimai atliekami laikantis modulinės aritmetikos $(mod\ p)$. Pirmiausiai pertvarkome lygtis, tam kad gautume visus B' matricos elementus:

$$\begin{aligned} &a_{11}b_{11} = y_{11}^1 \rightarrow b_{11} = y_{11}^1 a_{11}^{-1}, \ \ a_{11}b_{12} = y_{12}^1 \rightarrow b_{12} = y_{12}^1 a_{11}^{-1}, \\ &a_{11}b_{21} = y_{11}^2 \rightarrow b_{21} = y_{11}^2 a_{11}^{-1}, \ \ a_{11}b_{22} = y_{12}^2 \rightarrow b_{22} = y_{12}^2 a_{11}^{-1}. \end{aligned} \tag{2.14}$$

Pasinaudoję jau išvestomis lygtimis (2.14), sudarome likusias lygtis visiems reikiamiems A' elementams gauti:

$$\begin{aligned} &\begin{cases} a_{12}b_{11} = y_{11}^3 \\ a_{11}b_{11} = y_{11}^1 \end{cases} \rightarrow \begin{cases} b_{11} = y_{11}^3 a_{12}^{-1} \\ b_{11} = y_{11}^1 a_{11}^{-1} \end{cases} \rightarrow y_{11}^3 a_{12}^{-1} = y_{11}^1 a_{11}^{-1} \rightarrow a_{12} = y_{11}^3 (y_{11}^1)^{-1} a_{11}, \\ &\begin{cases} a_{21}b_{11} = y_{21}^1 \\ a_{11}b_{11} = y_{11}^1 \end{cases} \rightarrow \begin{cases} b_{11} = y_{21}^1 a_{21}^{-1} \\ b_{11} = y_{11}^1 a_{11}^{-1} \end{cases} \rightarrow y_{21}^1 a_{21}^{-1} = y_{11}^1 a_{11}^{-1} \rightarrow a_{21} = y_{21}^1 (y_{11}^1)^{-1} a_{11}, \\ &\begin{cases} a_{22}b_{11} = y_{21}^3 \\ a_{11}b_{11} = y_{11}^1 \end{cases} \rightarrow \begin{cases} b_{11} = y_{21}^3 a_{22}^{-1} \\ b_{11} = y_{11}^1 a_{11}^{-1} \end{cases} \rightarrow y_{21}^3 a_{22}^{-1} = y_{11}^1 a_{11}^{-1} \rightarrow a_{22} = y_{21}^3 (y_{11}^1)^{-1} a_{11}. \end{aligned} \tag{2.15}$$

Iš (2.14) ir (2.15) gavome $2m^2 - 1 = 7$ lygtis. Dabar jose turime vienintelį nepriklausomą kintamąjį a_{11}, o likę išsireiškia per jį. Gautas sprendinys:

$$A' = \begin{pmatrix} a_{11} & y_{11}^3 (y_{11}^1)^{-1} a_{11} \\ y_{21}^1 (y_{11}^1)^{-1} a_{11} & y_{21}^3 (y_{11}^1)^{-1} a_{11} \end{pmatrix}, B' = \begin{pmatrix} y_{11}^1 a_{11}^{-1} & y_{12}^1 a_{11}^{-1} \\ y_{11}^2 a_{11}^{-1} & y_{12}^2 a_{11}^{-1} \end{pmatrix}. \tag{2.16}$$

Panašias į (2.10) netiesinių lygčių sistemas galime susidaryti ir su trečios eilės kvadratinėmis matricomis. Laikysime, kad pati netiesinių lygčių sudarymo esmė jau yra aiški, todėl $m = 3$ atveju jos nebedetalizuosime. Tikslas išlieka toks pats – turėti vieną nepriklausomą kintamąjį a_{11}, o likusius išreikšti tik per jį. Norint tai pasiekti, čia reikia sukonstruoti $2 \cdot m^2 - 1 = 17$ tiesiškai nepriklausomų lygčių.

Sprendinių radimas, kai naudojamos trečios eilės matricos bei devynios įvesties ir išvesties matricos $(m = 3, n = 9)$. Sugeneruojamos slaptos raktinės matricos A, B. Kadangi vėl vykdome pasirinktos tekstogramos ataką, pasirenkame tokias įvesties matricas X_i, kad vėliau perrenkant tik vieną elementą visada būtų gaunami teisingi rezultatai:

$$X_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, X_2 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, X_3 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$
$$X_4 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, X_5 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, X_6 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \quad (2.17)$$
$$X_7 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, X_8 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, X_9 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Apskaičiuojame išvesties matricas:

$$A \cdot X_1 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^1 & y_{12}^1 & y_{13}^1 \\ y_{21}^1 & y_{22}^1 & y_{23}^1 \\ y_{31}^1 & y_{32}^1 & y_{33}^1 \end{pmatrix} = Y_1,$$
$$A \cdot X_2 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^2 & y_{12}^2 & y_{13}^2 \\ y_{21}^2 & y_{22}^2 & y_{23}^2 \\ y_{31}^2 & y_{32}^2 & y_{33}^2 \end{pmatrix} = Y_2,$$
$$A \cdot X_3 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^3 & y_{12}^3 & y_{13}^3 \\ y_{21}^3 & y_{22}^3 & y_{23}^3 \\ y_{31}^3 & y_{32}^3 & y_{33}^3 \end{pmatrix} = Y_3,$$
$$A \cdot X_4 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^4 & y_{12}^4 & y_{13}^4 \\ y_{21}^4 & y_{22}^4 & y_{23}^4 \\ y_{31}^4 & y_{32}^4 & y_{33}^4 \end{pmatrix} = Y_4,$$
$$A \cdot X_5 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^5 & y_{12}^5 & y_{13}^5 \\ y_{21}^5 & y_{22}^5 & y_{23}^5 \\ y_{31}^5 & y_{32}^5 & y_{33}^5 \end{pmatrix} = Y_5, \quad (2.18)$$
$$A \cdot X_6 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^6 & y_{12}^6 & y_{13}^6 \\ y_{21}^6 & y_{22}^6 & y_{23}^6 \\ y_{31}^6 & y_{32}^6 & y_{33}^6 \end{pmatrix} = Y_6,$$
$$A \cdot X_7 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^7 & y_{12}^7 & y_{13}^7 \\ y_{21}^7 & y_{22}^7 & y_{23}^7 \\ y_{31}^7 & y_{32}^7 & y_{33}^7 \end{pmatrix} = Y_7,$$
$$A \cdot X_8 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^8 & y_{12}^8 & y_{13}^8 \\ y_{21}^8 & y_{22}^8 & y_{23}^8 \\ y_{31}^8 & y_{32}^8 & y_{33}^8 \end{pmatrix} = Y_8,$$
$$A \cdot X_9 \cdot B \ (mod\ p) = \begin{pmatrix} y_{11}^9 & y_{12}^9 & y_{13}^9 \\ y_{21}^9 & y_{22}^9 & y_{23}^9 \\ y_{31}^9 & y_{32}^9 & y_{33}^9 \end{pmatrix} = Y_9.$$

Atliekame pažymėjimus:

$$A' = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, B' = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}. \quad (2.19)$$

Sudauginant raktines ir įvesties matricas nesudėtinga apskaičiuoti šifrogramos matricas:

$$A' \cdot X_1 \cdot B' \ (mod\ p) = Y_1, \rightarrow \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} \\ a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} \\ a_{31}b_{11} & a_{31}b_{12} & a_{31}b_{13} \end{pmatrix} = \begin{pmatrix} y_{11}^1 & y_{12}^1 & y_{13}^1 \\ y_{21}^1 & y_{22}^1 & y_{23}^1 \\ y_{31}^1 & y_{32}^1 & y_{33}^1 \end{pmatrix},$$

$$A' \cdot X_2 \cdot B' \ (mod\ p) = Y_2, \rightarrow \begin{pmatrix} a_{11}b_{21} & a_{11}b_{22} & a_{11}b_{23} \\ a_{21}b_{21} & a_{21}b_{22} & a_{21}b_{23} \\ a_{31}b_{21} & a_{31}b_{22} & a_{31}b_{23} \end{pmatrix} = \begin{pmatrix} y_{11}^2 & y_{12}^2 & y_{13}^2 \\ y_{21}^2 & y_{22}^2 & y_{23}^2 \\ y_{31}^2 & y_{32}^2 & y_{33}^2 \end{pmatrix},$$

$$A' \cdot X_3 \cdot B' \ (mod\ p) = Y_3, \rightarrow \begin{pmatrix} a_{11}b_{31} & a_{11}b_{32} & a_{11}b_{33} \\ a_{21}b_{31} & a_{21}b_{32} & a_{21}b_{33} \\ a_{31}b_{31} & a_{31}b_{32} & a_{31}b_{33} \end{pmatrix} = \begin{pmatrix} y_{11}^3 & y_{12}^3 & y_{13}^3 \\ y_{21}^3 & y_{22}^3 & y_{23}^3 \\ y_{31}^3 & y_{32}^3 & y_{33}^3 \end{pmatrix},$$

$$A' \cdot X_4 \cdot B' \ (mod\ p) = Y_4, \rightarrow \begin{pmatrix} a_{12}b_{11} & a_{12}b_{12} & a_{12}b_{13} \\ a_{22}b_{11} & a_{22}b_{12} & a_{22}b_{13} \\ a_{32}b_{11} & a_{32}b_{12} & a_{32}b_{13} \end{pmatrix} = \begin{pmatrix} y_{11}^4 & y_{12}^4 & y_{13}^4 \\ y_{21}^4 & y_{22}^4 & y_{23}^4 \\ y_{31}^4 & y_{32}^4 & y_{33}^4 \end{pmatrix},$$

$$A' \cdot X_5 \cdot B' \ (mod\ p) = Y_5, \rightarrow \begin{pmatrix} a_{12}b_{21} & a_{12}b_{22} & a_{12}b_{23} \\ a_{22}b_{21} & a_{22}b_{22} & a_{22}b_{23} \\ a_{32}b_{21} & a_{32}b_{22} & a_{32}b_{23} \end{pmatrix} = \begin{pmatrix} y_{11}^5 & y_{12}^5 & y_{13}^5 \\ y_{21}^5 & y_{22}^5 & y_{23}^5 \\ y_{31}^5 & y_{32}^5 & y_{33}^5 \end{pmatrix}, \quad (2.20)$$

$$A' \cdot X_6 \cdot B' \ (mod\ p) = Y_6, \rightarrow \begin{pmatrix} a_{12}b_{31} & a_{12}b_{32} & a_{12}b_{33} \\ a_{22}b_{31} & a_{22}b_{32} & a_{22}b_{33} \\ a_{32}b_{31} & a_{32}b_{32} & a_{32}b_{33} \end{pmatrix} = \begin{pmatrix} y_{11}^6 & y_{12}^6 & y_{13}^6 \\ y_{21}^6 & y_{22}^6 & y_{23}^6 \\ y_{31}^6 & y_{32}^6 & y_{33}^6 \end{pmatrix},$$

$$A' \cdot X_7 \cdot B' \ (mod\ p) = Y_7, \rightarrow \begin{pmatrix} a_{13}b_{11} & a_{13}b_{12} & a_{13}b_{13} \\ a_{23}b_{11} & a_{23}b_{12} & a_{23}b_{13} \\ a_{33}b_{11} & a_{33}b_{12} & a_{33}b_{13} \end{pmatrix} = \begin{pmatrix} y_{11}^7 & y_{12}^7 & y_{13}^7 \\ y_{21}^7 & y_{22}^7 & y_{23}^7 \\ y_{31}^7 & y_{32}^7 & y_{33}^7 \end{pmatrix},$$

$$A' \cdot X_8 \cdot B' \ (mod\ p) = Y_8, \rightarrow \begin{pmatrix} a_{13}b_{21} & a_{13}b_{22} & a_{13}b_{23} \\ a_{23}b_{21} & a_{23}b_{22} & a_{23}b_{23} \\ a_{33}b_{21} & a_{33}b_{22} & a_{33}b_{23} \end{pmatrix} = \begin{pmatrix} y_{11}^8 & y_{12}^8 & y_{13}^8 \\ y_{21}^8 & y_{22}^8 & y_{23}^8 \\ y_{31}^8 & y_{32}^8 & y_{33}^8 \end{pmatrix},$$

$$A' \cdot X_9 \cdot B' \ (mod\ p) = Y_9, \rightarrow \begin{pmatrix} a_{13}b_{31} & a_{13}b_{32} & a_{13}b_{33} \\ a_{23}b_{31} & a_{23}b_{32} & a_{23}b_{33} \\ a_{33}b_{31} & a_{33}b_{32} & a_{33}b_{33} \end{pmatrix} = \begin{pmatrix} y_{11}^9 & y_{12}^9 & y_{13}^9 \\ y_{21}^9 & y_{22}^9 & y_{23}^9 \\ y_{31}^9 & y_{32}^9 & y_{33}^9 \end{pmatrix}.$$

Tuomet įvairiais būdais pertvarkome netiesines lygtis, kad jose liktų tik vienas nepriklausomas kintamasis a_{11}, o visus kitus galėtume išreikšti per jį. Visi skaičiavimai, panašiai kaip ir antros eilės matricų atveju, atliekami laikantis modulinės aritmetikos $(mod\ p)$. Pirmiausiai pertvarkome lygtis tam, kad gautume visus B' matricos elementus. Šių elementų galime ieškoti daugiau nei vieninteliu metodu, tačiau sprendiniui sudaryti pakaks tik vieno, kurį ir pateikiame.

$$\begin{aligned} &a_{11}b_{11} = y_{11}^1 \rightarrow b_{11} = y_{11}^1 a_{11}^{-1},\ a_{11}b_{12} = y_{12}^1 \rightarrow b_{12} = y_{12}^1 a_{11}^{-1},\ a_{11}b_{13} = y_{13}^1 \rightarrow b_{13} = y_{13}^1 a_{11}^{-1} \\ &a_{11}b_{21} = y_{11}^2 \rightarrow b_{21} = y_{11}^2 a_{11}^{-1},\ a_{11}b_{22} = y_{12}^2 \rightarrow b_{22} = y_{12}^2 a_{11}^{-1},\ a_{11}b_{23} = y_{13}^2 \rightarrow b_{23} = y_{13}^2 a_{11}^{-1} \\ &a_{11}b_{31} = y_{11}^3 \rightarrow b_{31} = y_{11}^3 a_{11}^{-1},\ a_{11}b_{32} = y_{12}^3 \rightarrow b_{32} = y_{12}^3 a_{11}^{-1},\ a_{11}b_{33} = y_{13}^3 \rightarrow b_{33} = y_{13}^3 a_{11}^{-1} \end{aligned} \quad (2.21)$$

Taip pat sudarome likusias lygtis visiems reikiamiems A' elementams gauti. Suradę ir šios matricos reikšmes, jau galėsime sudaryti pilną sprendinį.

$$
\begin{aligned}
&\begin{cases} a_{12}b_{11} = y_{11}^{4} \\ a_{11}b_{11} = y_{11}^{1} \end{cases} \rightarrow \begin{cases} b_{11} = y_{11}^{4}a_{12}^{-1} \\ b_{11} = y_{11}^{1}a_{11}^{-1} \end{cases} \rightarrow y_{11}^{4}a_{12}^{-1} = y_{11}^{1}a_{11}^{-1} \rightarrow a_{12} = y_{11}^{4}(y_{11}^{1})^{-1}a_{11}, \\
&\begin{cases} a_{13}b_{11} = y_{11}^{7} \\ a_{11}b_{11} = y_{11}^{1} \end{cases} \rightarrow \begin{cases} b_{11} = y_{11}^{7}a_{13}^{-1} \\ b_{11} = y_{11}^{1}a_{11}^{-1} \end{cases} \rightarrow y_{11}^{7}a_{13}^{-1} = y_{11}^{1}a_{11}^{-1} \rightarrow a_{13} = y_{11}^{7}(y_{11}^{1})^{-1}a_{11}, \\
&\begin{cases} a_{21}b_{11} = y_{21}^{1} \\ a_{11}b_{11} = y_{11}^{1} \end{cases} \rightarrow \begin{cases} b_{11} = y_{21}^{1}a_{21}^{-1} \\ b_{11} = y_{11}^{1}a_{11}^{-1} \end{cases} \rightarrow y_{21}^{1}a_{21}^{-1} = y_{11}^{1}a_{11}^{-1} \rightarrow a_{21} = y_{21}^{1}(y_{11}^{1})^{-1}a_{11}, \\
&\begin{cases} a_{22}b_{11} = y_{21}^{4} \\ a_{11}b_{11} = y_{11}^{1} \end{cases} \rightarrow \begin{cases} b_{11} = y_{21}^{4}a_{22}^{-1} \\ b_{11} = y_{11}^{1}a_{11}^{-1} \end{cases} \rightarrow y_{21}^{4}a_{22}^{-1} = y_{11}^{1}a_{11}^{-1} \rightarrow a_{22} = y_{21}^{4}(y_{11}^{1})^{-1}a_{11}, \\
&\begin{cases} a_{23}b_{11} = y_{21}^{7} \\ a_{11}b_{11} = y_{11}^{1} \end{cases} \rightarrow \begin{cases} b_{11} = y_{21}^{7}a_{23}^{-1} \\ b_{11} = y_{11}^{1}a_{11}^{-1} \end{cases} \rightarrow y_{21}^{7}a_{23}^{-1} = y_{11}^{1}a_{11}^{-1} \rightarrow a_{23} = y_{21}^{7}(y_{11}^{1})^{-1}a_{11}, \\
&\begin{cases} a_{31}b_{11} = y_{31}^{1} \\ a_{11}b_{11} = y_{11}^{1} \end{cases} \rightarrow \begin{cases} b_{11} = y_{31}^{1}a_{31}^{-1} \\ b_{11} = y_{11}^{1}a_{11}^{-1} \end{cases} \rightarrow y_{31}^{1}a_{31}^{-1} = y_{11}^{1}a_{11}^{-1} \rightarrow a_{31} = y_{31}^{1}(y_{11}^{1})^{-1}a_{11}, \\
&\begin{cases} a_{32}b_{11} = y_{31}^{4} \\ a_{11}b_{11} = y_{11}^{1} \end{cases} \rightarrow \begin{cases} b_{11} = y_{31}^{4}a_{32}^{-1} \\ b_{11} = y_{11}^{1}a_{11}^{-1} \end{cases} \rightarrow y_{31}^{4}a_{32}^{-1} = y_{11}^{1}a_{11}^{-1} \rightarrow a_{32} = y_{31}^{4}(y_{11}^{1})^{-1}a_{11}, \\
&\begin{cases} a_{33}b_{11} = y_{31}^{7} \\ a_{11}b_{11} = y_{11}^{1} \end{cases} \rightarrow \begin{cases} b_{11} = y_{31}^{7}a_{33}^{-1} \\ b_{11} = y_{11}^{1}a_{11}^{-1} \end{cases} \rightarrow y_{31}^{7}a_{33}^{-1} = y_{11}^{1}a_{11}^{-1} \rightarrow a_{33} = y_{31}^{7}(y_{11}^{1})^{-1}a_{11}.
\end{aligned}
\tag{2.22}
$$

Gautas matricų reikšmes sustatome į reikiamas vietas ir galiausiai pateikiame surastą sprendinį trečios eilės matricoms:

$$
\begin{aligned}
A' &= \begin{pmatrix} a_{11} & y_{11}^{4}(y_{11}^{1})^{-1}a_{11} & y_{11}^{7}(y_{11}^{1})^{-1}a_{11} \\ y_{21}^{1}(y_{11}^{1})^{-1}a_{11} & y_{21}^{4}(y_{11}^{1})^{-1}a_{11} & y_{21}^{7}(y_{11}^{1})^{-1}a_{11} \\ y_{31}^{1}(y_{11}^{1})^{-1}a_{11} & y_{31}^{4}(y_{11}^{1})^{-1}a_{11} & y_{31}^{7}(y_{11}^{1})^{-1}a_{11} \end{pmatrix}, \\
B' &= \begin{pmatrix} y_{11}^{1}a_{11}^{-1} & y_{12}^{1}a_{11}^{-1} & y_{13}^{1}a_{11}^{-1} \\ y_{11}^{2}a_{11}^{-1} & y_{12}^{2}a_{11}^{-1} & y_{13}^{2}a_{11}^{-1} \\ y_{11}^{3}a_{11}^{-1} & y_{12}^{3}a_{11}^{-1} & y_{13}^{3}a_{11}^{-1} \end{pmatrix}.
\end{aligned}
\tag{2.23}
$$

Algebrinių lygčių sprendimo algoritmo atveju, sprendinių analizės dalyje, nagrinėsime matricines lygtis virš baigtinio lauko tik su antros ($m = 2, n = 4$) ir trečios ($m = 3, n = 9$) eilės kvadratinėmis matricomis.

2.2. SPRENDINIŲ ANALIZĖ

2.2.1 TYRIMAS MATRICŲ VISIŠKO PERRINKIMO ALGORITMU

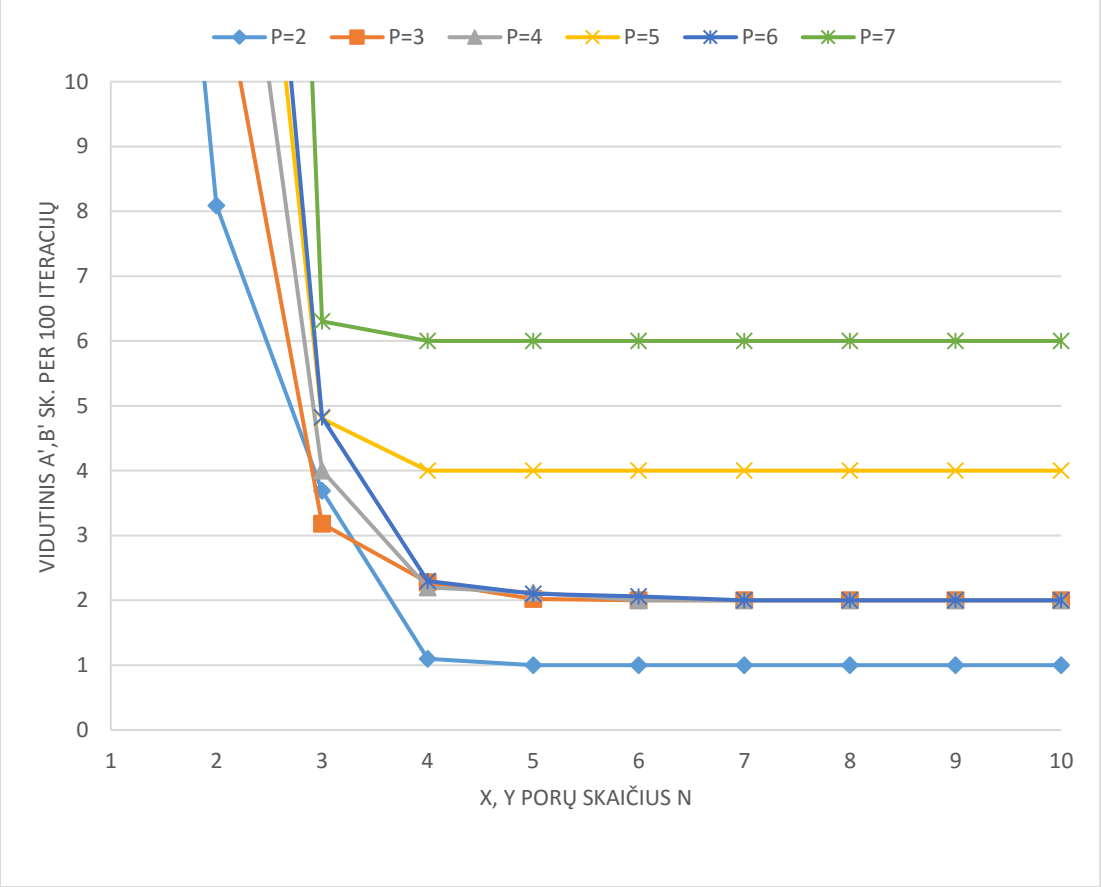
Šiame skyriuje aprašomas sprendinių tyrimas matricų visiško perrinkimo algoritmu. Pateikiami šiuo algoritmu gauti rezultatai, išsamiai aprašoma jų gavimo eiga.

Atliekant eksperimentinius skaičiavimus skirtingose multiplikacinėse matricų pusgrupėse visiško perrinkimo algoritmo pagalba, nesunkiai pastebėjome, kad vykdant daugiau negu vieną iteraciją (kai fiksuojame raktines matricas A, B, tačiau keičiame X, Y matricas) gauname skirtingą tinkamų sprendinių skaičių. Kitaip tariant, tinkamų A', B' skaičius priklauso nuo pačių X, Y matricų elementų.

Tuo galėsime įsitikinti kiek vėliau, programinės realizacijos skyriuje detaliau paanalizavę 3.5 pav. matomą „A', B' pairs number“ langą, kur registruojamas iteracijos numeris ir tos iteracijos metu surastas tinkančių lygčiai išspręsti A', B' raktinių matricų skaičius.

Nesunku suprasti, kad vykdant visišką perrinkimą su didesniu įvesties ir išvesties matricų X, Y porų skaičiumi n, tikrinamos matricos turi tenkinti daugiau matricinių lygčių. Dėl šios priežasties esant didesniam n, tinkamų sprendinių A', B' skaičius bus mažesnis. Tokiu būdu siekiama rasti A', B' sprendinius, kurie tiktų bet kokiai matricų X, Y porai.

Pastaba: atkreipkime dėmesį, jog kalbame apie sprendinius, visiškai tinkančius bet kokiai matricų X, Y porai. Jei turime nedidelį, fiksuotą X, Y porų skaičių (pavyzdžiui tik vieną), tai sprendinių gali būti ir daugiau.



2.1 pav. Ryšys tarp vidutinio A', B' porų skaičiaus ir X, Y porų skaičiaus n, kai $m = 2$

Pateiksime rezultatą, kuris iliustruoja, kaip matricų visiško perrinkimo algoritmu galime rasti A', B' sprendinius, kurie tiktų bet kokiai matricų X, Y porai sprendžiant (2.3) matricines lygtis. Ši sprendinių radimo technika vadinama žinomos tekstogramos ataka. Rezultatas grafiškai atvaizduotas 2.1 pav. Jį trumpai aptarsime. Kiekvienu p atveju naudojamas fiksuotas iteracijų skaičius – 100 iteracijų. Taip pat fiksuota matricų eilė $m = 2$. Čia pateikiami atvejai, kai sveikasis skaičius p kinta

nuo 2 iki 7, o X, Y porų skaičius n kinta nuo 1 iki 10. Su didesnėmis p reikšmėmis algoritmo vykdymo laikas tampa nepriimtinai ilgas, dėl to tokių atvejų toliau nenagrinėjame. Paveiksle matome, kad vykdant palyginus daug iteracijų bei didinant įvesties ir išvesties matricų skaičių n, vidutinis A', B' sprendinių skaičius mažėja. Priklausomai nuo sveikojo skaičiaus p, galiausiai pasiekiama tam tikra riba ir sprendinių skaičius stabilizuojasi.

Taigi, iš grafiko 2.1 pav. fiksuojame, kad visais čia pateiktų skirtingų p atvejais, tinkamų A', B' sprendinių skaičius nusistovi naudojant 7 įvesties ir išvesties matricų X, Y poras.

Tolimesnėje tyrimo eigoje, naudojome mažesnį iteracijų skaičių, nes didinant p sparčiai didėja vykdymo trukmė. Tada didinome X, Y porų skaičių bei panašiu būdu pamėginome patikrinti sprendinių skaičių su didesniais sveikųjų skaičių žiedais – kai p kinta nuo 2 iki 15. Šiems rezultatams pateikti sudaryta 2.3 lentelė.

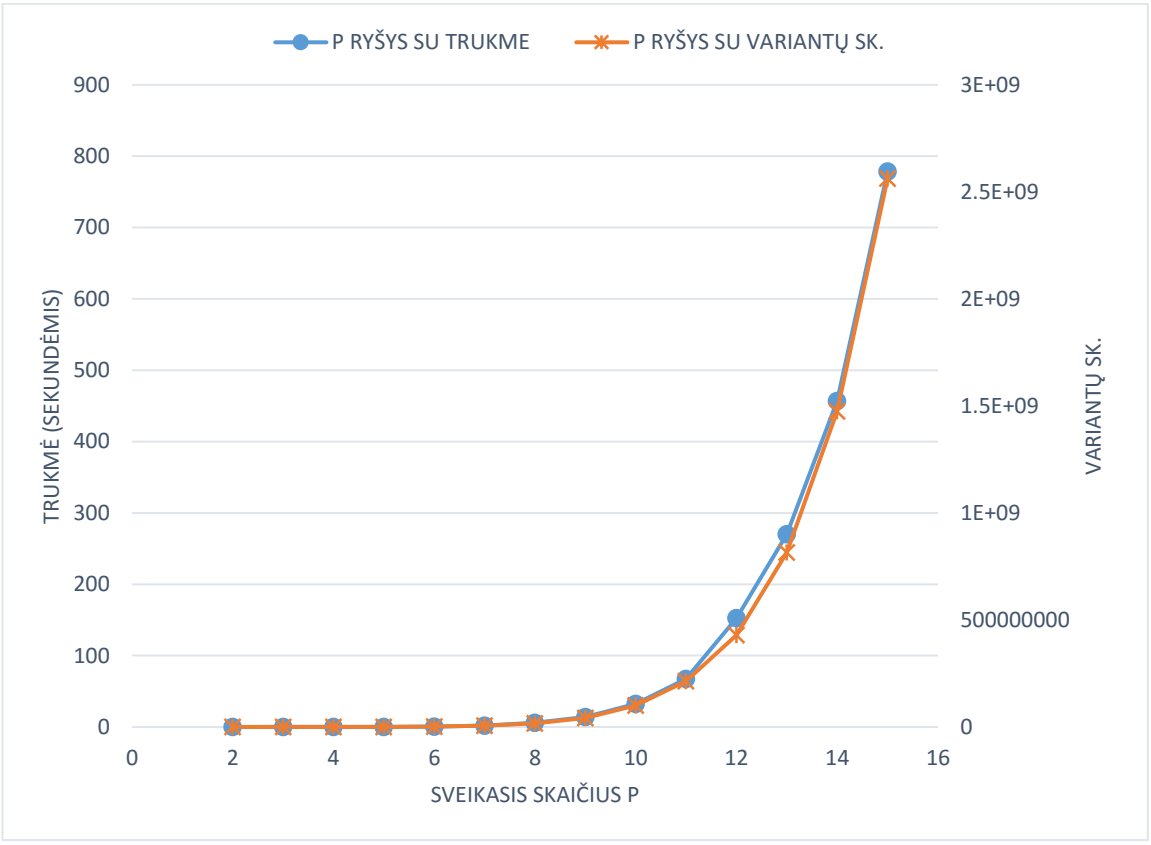
2.3 lentelė. A', B' sprendinių skaičius, prie skirtingų p

p	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sprendinių A', B' sk.	1	2	2	4	2	6	4	6	4	10	4	12	6	8

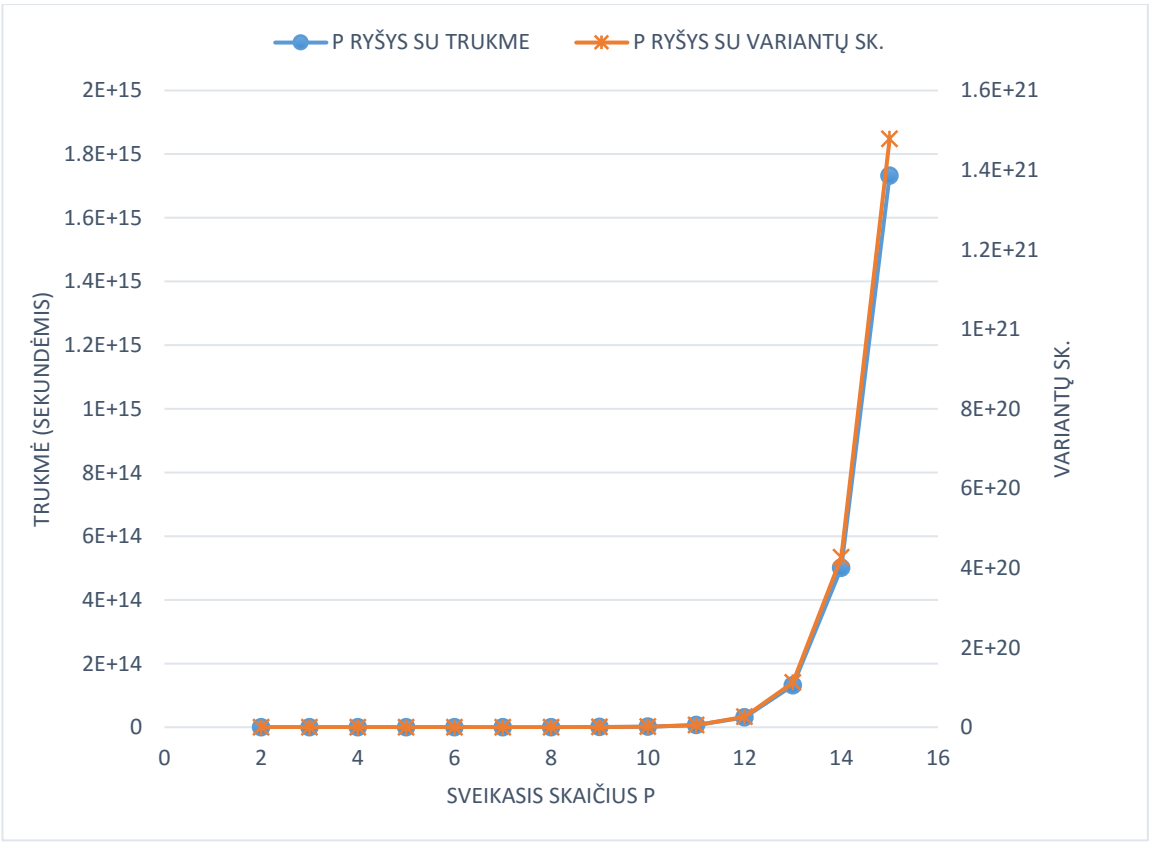
Su dar didesnėmis p reikšmėmis visiško perrinkimo algoritmo sprendinių paieškos laikas užtrunka labai ilgai, net ir atliekant tik vieną iteraciją. Dėl šios priežasties rezultatai pateikiami, tik su p iš intervalo nuo 2 iki 15. Tam kad nustatytume, kaip keičiasi sprendinių paieškos laikas, kai didinamas p, atlikome visiško perrinkimo algoritmo efektyvumo tyrimą, kurį ir pristatysime sekančiame tyrimo etape. Sprendžiant matricines lygtis virš baigtinio žiedo, algoritmo efektyvumą tyrėme su antros ($m = 2, n = 4$) ir trečios ($m = 3, n = 9$) eilės kvadratinėmis matricomis.

Matricų visiško perrinkimo algoritmo efektyvumui nustatyti su antros eilės matricomis prie skirtingų p reikšmių tyrėme jo vykdymo trukmę. Nagrinėdami šio algoritmo efektyvumą prie skirtingų parametrų p, fiksavome parametrus ($m = 2, n = 4$). Svarbu, kad atliekant tikrinimus n būtų fiksuotas. Priešingu atveju, būtų sprendžiamas ne vienodas matricinių lygčių skaičius. Jau aptarėme, kad sprendžiant daugiau matricinių lygčių algoritmas vykdomas ilgiau, todėl tokiu atveju rezultatų palyginimas būtų ne korektiškas. Rezultatai, susiję su šio algoritmo vykdymo laiko įverčiu (kai $m = 2$), pateikiami 2.2 pav.

Nagrinėjome ir ryšį tarp p ir reikalingų patikrinti variantų skaičiaus. Čia variantų skaičių bendru atveju galime apskaičiuoti pagal formulę: p^{2m^2}. Kadangi 2.2 pav. atsispindi rezultatai su antros eilės matricomis ($m = 2$), tai šiuo konkrečiu atveju variantų skaičius bus $p^{2 \cdot 2^2} = p^8$. Reikalingų patikrinti variantų skaičių taip pat atvaizdavome grafiškai 2.2 pav., panaudodami papildomą vertikalią ašį, ant kurios ir atidėtos variantų skaičiaus reikšmės. Matome, kad grafikai praktiškai sutampa. Tiek sveikojo skaičiaus p su trukme, tiek p su variantų skaičiumi sieja laipsninė priklausomybė.



2.2 pav. p ryšys su trukme ir variantų skaičiumi vykdant matricų visišką perrinkimą ($m = 2$)



2.3 pav. p ryšys su trukme ir variantų skaičiumi vykdant matricų visišką perrinkimą ($m = 3$)

Vėliau pamėginome patyrinėti visiško matricų perrinkimo efektyvumą sprendžiant matricines lygtis virš baigtinio žiedo su didesnėmis kvadratinėmis matricomis ($m = 3, n = 9$). Kadangi su trečios eilės matricomis algoritmo vykdymo laikas didėja labai sparčiai, čia p iš intervalo nuo 2 iki 15 praktiškai patikrinti negalime. Tada pasinaudojome jau surastu panašumu tarp vykdymo trukmės ir variantų skaičiaus, kuomet naudojome antros eilės matricas – 2.2 pav. Tuomet žinant visiško perrinkimo algoritmo vykdymo trukmę su trečios eilės matricomis prie mažų p reikšmių (pavyzdžiui, kai $p = 2$) ir apskaičiavus reikiamų patikrinti variantų skaičių, galime prognozuoti vykdymo trukmę ir su didesniais p. Tai atvaizduota grafiškai 2.3 pav.

Pastebėjome, kad tiek su didesniais žiedais, tiek su didesnėmis matricų eilėmis matricų visiško perrinkimo algoritmo vykdymo laikas didėja labai greitai. Manoma, kad algebrinių lygčių sprendimo algoritmu tinkamas A', B' matricas galime surasti per žymiai trumpesnį laiką, negu atliekant visišką matricų perrinkimą.

2.2.2 TEORINIS SPRENDINIŲ SKAIČIAUS PAGRINDIMAS

Ankstesniame skyriuje visiško perrinkimo algoritmu nustatėme, koks yra visiškai tinkančių bet kokiai X, Y porai A', B' sprendinių skaičius. Šiame skyriuje, surastus sprendinių skaičius prie skirtingų p, pagrįsime teoriškai. Kiek vėliau geresniam supratimui pateiksime ir pavyzdį, kuris įrodo teorinio pagrindimo teisingumą.

Pirmiausiai įvairiais atvejais detaliau panagrinėjome visiško perrinkimo algoritmo surastus sprendinius. Čia pradinis tikslas pastebėti dėsningumus tarp visų tinkamų A', B' matricų. Visas A' matricas pamėginome padauginti iš tikrosios atvirkštinės A^{-1}, ir analogiškai B' matricas iš B^{-1}.

Atlikus tokius veiksmus, visų tinkamų A' sprendinių atvejais, gavome vienetines matricas I, padaugintas iš tam tikros konstantos u: $A' \cdot A^{-1} = uI \rightarrow A' = uA$. Analogiškai, B' atvejais, I padaugintas iš konstantos v: $B' \cdot B^{-1} = vI \rightarrow B' = vB$. Toliau, šias u ir v konstantas sudauginome laikydamiesi pasirinkto žiedo aritmetikos ir visais atvejais gavome sandaugas lygias 1. Pagal žiedų savybes jau žinome, kad skaičiai, kurių sandauga yra lygi 1 vadinami atvirkštiniais.

Viską apibendrius galime užrašyti svarbų teorinį pagrindimą: jei $A \cdot X \cdot B \ (mod\ p) = Y$, $A' = uA$ ir $B' = vB$, tai $Y = A' \cdot X \cdot B' \ (mod\ p) = \ uA \cdot X \cdot vB \ (mod\ p) = uv \cdot A \cdot X \cdot B \ \ (mod\ p)$. Iš čia gauname, kad $uv = 1$, t. y. jie turi būti atvirkštiniai vienas kitam. Taigi, mūsų nagrinėjamose lygtyse, tinkamų A', B' sprendinių skaičius ir atitinka žiedo elementų skaičių, kurie turi atvirkštinius.

Iš teorijos žinome, kad atvirkštinių elementų skaičius žiede yra lygus Oilerio funkcijai $\varphi(p)$ (Sakalauskas ir kt., 2008). Taigi, lygčių sistemai (2.3) visuomet bus $\varphi(p)$ teisingų A', B' matricų porų.

Patikrinus nagrinėtų žiedų atvirkštinių elementų skaičius įsitikinome, kad jie sutampa su A', B' sprendinių skaičiais, surastais visiško perrinkimo algoritmo pagalba (2.3 lentelė). Toliau aiškumo dėlei pateiksime vieną, tai iliustruojantį pavyzdį.

Pavyzdys (su nepirminiu p). Parenkame sisteminius parametrus $p = 6,\ m = 2,\ n = 3$ ir atliekame skaičiavimus pagal anksčiau aprašytą tvarką. Pirmiausiai parenkame raktines A, B matricas ir apskaičiuojame jų modulines atvirkštines matricas:

$$A = \begin{pmatrix} 3 & 2 \\ 2 & 3 \end{pmatrix} \to A^{-1} = \begin{pmatrix} 3 & 2 \\ 2 & 3 \end{pmatrix}, B = \begin{pmatrix} 1 & 2 \\ 5 & 3 \end{pmatrix} \to B^{-1} = \begin{pmatrix} 3 & 2 \\ 5 & 5 \end{pmatrix}. \tag{2.24}$$

Pateikiame atsitiktinai sugeneruotas įvesties matricas X ir jau įprastu būdu apskaičiuotas išvesties matricas Y:

$$\begin{gathered} X_1 = \begin{pmatrix} 2 & 2 \\ 5 & 1 \end{pmatrix}, X_2 = \begin{pmatrix} 0 & 4 \\ 2 & 1 \end{pmatrix}, X_3 = \begin{pmatrix} 5 & 2 \\ 0 & 1 \end{pmatrix}, \\ Y_1 = \begin{pmatrix} 2 & 2 \\ 0 & 5 \end{pmatrix}, Y_2 = \begin{pmatrix} 2 & 2 \\ 1 & 3 \end{pmatrix}, Y_3 = \begin{pmatrix} 1 & 0 \\ 3 & 5 \end{pmatrix}. \end{gathered} \tag{2.25}$$

Visiško matricų perrinkimo algoritmu surandami visi (šiuo atveju tik 2) tinkami sprendiniai:

$$A_1' = \begin{pmatrix} 3 & 2 \\ 2 & 3 \end{pmatrix},\ B_1' = \begin{pmatrix} 1 & 2 \\ 5 & 3 \end{pmatrix} \text{ ir } A_2' = \begin{pmatrix} 3 & 4 \\ 4 & 3 \end{pmatrix},\ B_2' = \begin{pmatrix} 5 & 4 \\ 1 & 3 \end{pmatrix}. \tag{2.26}$$

Gautus tinkamus sprendinius dauginame iš modulinių atvirkštinių matricų A^{-1}, B^{-1}. Tokiu būdu randame konstantas u_1, v_1 ir u_2, v_2:

$$\begin{gathered} A_1' \cdot A^{-1} = \begin{pmatrix} 3 & 2 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 3 & 2 \\ 2 & 3 \end{pmatrix} (mod\ 6) = 1 \cdot I \to u_1 = 1 \to A_1' = u_1 A = 1 \cdot A, \\ B_1' \cdot B^{-1} = \begin{pmatrix} 1 & 2 \\ 5 & 3 \end{pmatrix} \cdot \begin{pmatrix} 3 & 2 \\ 5 & 5 \end{pmatrix} (mod\ 6) = 1 \cdot I \to v_1 = 1 \to B_1' = v_1 B = 1 \cdot B, \\ A_2' \cdot A^{-1} = \begin{pmatrix} 3 & 4 \\ 4 & 3 \end{pmatrix} \cdot \begin{pmatrix} 3 & 2 \\ 2 & 3 \end{pmatrix} (mod\ 6) = 5 \cdot I \to u_2 = 5 \to A_2' = u_2 A = 5 \cdot A, \\ B_2' \cdot B^{-1} = \begin{pmatrix} 5 & 4 \\ 1 & 3 \end{pmatrix} \cdot \begin{pmatrix} 3 & 2 \\ 5 & 5 \end{pmatrix} (mod\ 6) = 5 \cdot I \to v_2 = 5 \to B_2' = v_2 B = 5 \cdot B. \end{gathered} \tag{2.27}$$

Pagrindimas:

$$\begin{gathered} Y = A_1' \cdot X \cdot B_1'\ (mod\ 6) = (u_1 A) \cdot X \cdot (v_1 B)\ (mod\ 6) = (1 \cdot A) \cdot X \cdot (1 \cdot B)(mod\ 6) = \\ (1 \cdot 1) \cdot (A \cdot X \cdot B)\ (mod\ 6) = 1 \cdot (A \cdot X \cdot B) \to u_1 v_1 = 1. \\ Y = A_2' \cdot X \cdot B_2'\ (mod\ 6) = (u_2 A) \cdot X \cdot (v_2 B)\ (mod\ 6) = (5 \cdot A) \cdot X \cdot (5 \cdot B)(mod\ 6) = \\ (5 \cdot 5) \cdot (A \cdot X \cdot B)\ (mod\ 6) = 1 \cdot (A \cdot X \cdot B) \to u_2 v_2 = 1. \end{gathered} \tag{2.28}$$

Gavome, kad konstantų u ir v sandaugos yra lygios 1, o tai reiškia, kad u ir v – vienas kitam atvirkštiniai skaičiai. Skaičiuojame Oilerio funkciją $\varphi(p) = \varphi(6) = 2$. Nesunku įsitikinti, kad 1 ir 5 yra vieninteliai du šio žiedo elementai, kurie turi atvirkštinius. Taigi, tinkamų sprendinių A', B' skaičius ir atvirkštinių elementų skaičius nagrinėtame baigtiniame sveikųjų skaičių žiede sutampa ir yra lygus 2. Tuo galime dar kartą įsitikinti pasitikrindami 2.3 lentelėje. Panašiu būdu tinkamų A', B' skaičių galime pagrįsti teoriškai ir kitais atvejais.

2.2.3 TYRIMAS ALGEBRINIŲ LYGČIŲ SPRENDIMO ALGORITMU

Šiame skyriuje aprašysime algebrinių lygčių sprendimo algoritmu atlikus tyrimus. Jau nustatėme sprendinių skaičių atliekant visišką matricų perrinkimą. Vėliau gautus sprendinių skaičius pagrindėme teoriškai. Sprendinių skaičiui nustatyti perrenkant visus įmanomus variantus matricų visiško perrinkimo algoritmas yra puikiai tinkamas. Deja, susiduriama su kita problema – didinant parametrus pastarojo algoritmo vykdymo trukmė auga labai sparčiai ir norint didesnio efektyvumo sprendinių paieškoje, šis algoritmas nėra pats geriausias pasirinkimas. Todėl alternatyvus algebrinių lygčių sprendimo algoritmas sukurtas tam tikslui, kad analogiškus sprendinius (gautiems visiško perrinkimo algoritmu) surastume efektyvesniu būdu.

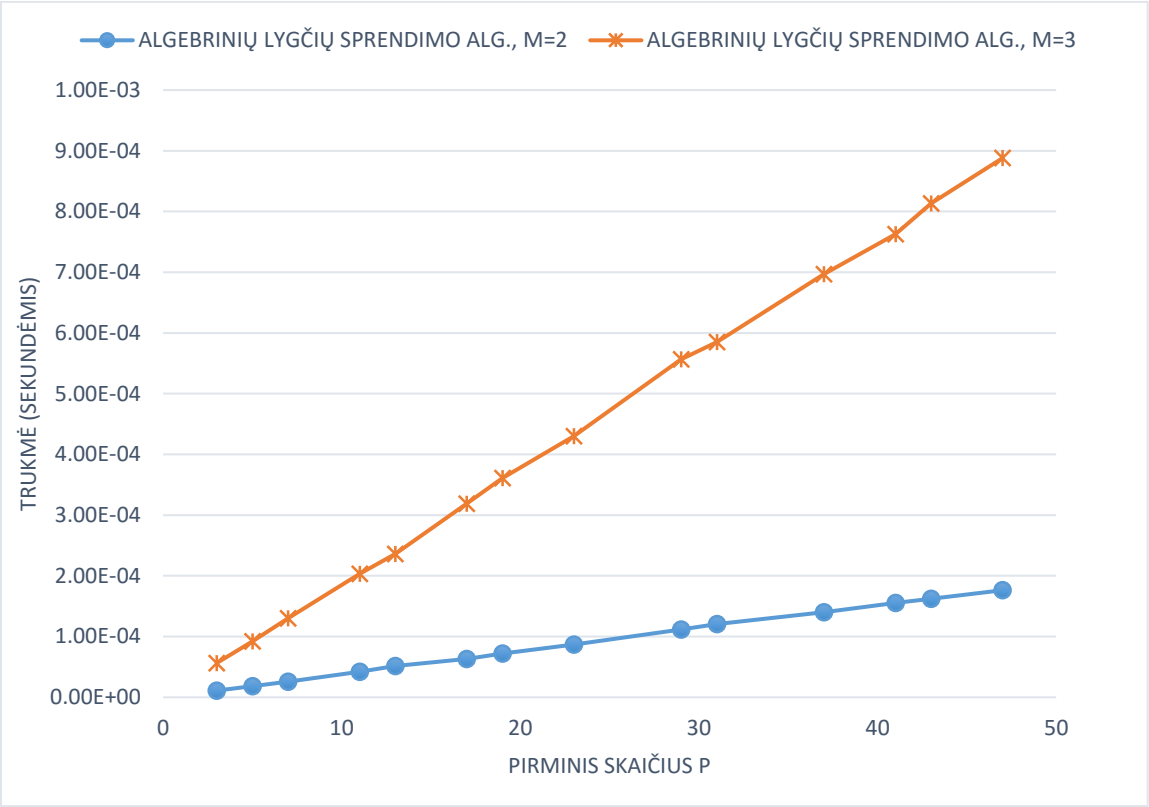
Pagal algebrinių lygčių sprendimo algoritmo aprašymą ankstesniuose skyriuose, atlikome jo programinę realizaciją. Čia reikėtų nepamiršti, jog sprendžiamos matricinės lygtys virš baigtinio lauko, bet ne žiedo (p visuomet tik pirminis). Algebrinių lygčių sprendimo algoritmas spręsti matricines lygtys virš baigtinio žiedo būtų ne visada teisingas, dėl to analizuojamas tik baigtinio lauko atvejis.

Trumpai paaiškinsime šį apribojimą. Pasirodo, kad kai į tvarkingo sprendinio (2.16) arba (2.23) sudarytas lygtis statome reikšmes ir skaičiuojame, kartais gali tekti dauginti iš tokių skaičių atvirkštinių, kurie iš tikrųjų neturi atvirkštinių tame žiede. Dėl to iš karto įvedame apribojimą: naudojami žiedai be nulio $\mathbb{Z}_p\backslash\{0\}$, kadangi nulinis elementas niekada neturi atvirkštinio. Tačiau žieduose yra ir daugiau elementų, kurie neturi atvirkštinių.

Pavyzdžiui, paimkime žiedą su $p = 15$ (tai nėra pirminis skaičius). Tarkime, kad norime apskaičiuoti kurį nors vieną A' arba B' matricos elementą ir atliekant veiksmus tenka dauginti iš skaičiaus 9 atvirkštinio (tačiau žiede $\mathbb{Z}_{15}$ šis skaičius atvirkštinio neturi). Dėl šios priežasties mūsų sudarytų lygčių išspręsti negalėsime. Todėl algebrinių lygčių algoritmu nutarėme tirti tik lauko atvejį – kai p pirminis, liekanų žiedas yra laukas ir visi elementai būtinai turės atvirkštinius. Tai išplaukia iš anksčiau matematinių savokų skyriuje pateiktos teoremos apie lauką (Sakalauskas ir kt., 2008).

Vykdydami pasirinktos tekstogramos ataką alternatyviu algoritmu, tyrimą atlikome su pirminiais p intervale nuo 3 iki 47. Čia $\mathbb{Z}_2$ laukas nenagrinėjamas, nes jame leidžiamų elementų aibė yra tik vienas elementas: $\mathbb{Z}_2 = \{1\}$, o tokiu atveju pradinės raktinės matricos A, B niekada nebūtų reguliarios. Tai reikštų, jog neegzistuoja jų modulinės atvirkštinės matricos ir iš šifrogramas atitinkančių matricų Y negalėtume atkurti pradinės paslėptos informacijos X.

Vėliau ištyrėme algebrinių lygčių sprendimo algoritmo sprendinių suradimo efektyvumą antros ir trečios eilės kvadratinių matricų atvejais ($m = 2, m = 3$). Trukmės palyginimui prie skirtingų matricų eilių pateikiame 2.4 pav. Matome, kad didinant lauko dydį algebrinių lygčių sprendimo algoritmo vykdymo trukmė didėja pagal tiesinę priklausomybę. Čia vykdymo trukmė su didesne matricų eile padidėja ne taip stipriai kaip matricų visiško perrinkimo atveju.



2.4 pav. Algebrinių lygčių sprendimo algoritmas ($m = 2, m = 3$)

Taigi išsiaiškinome, kad baigtinio lauko atveju alternatyviu algoritmu tikrai galime gauti tokius pačius sprendinius kaip ir perrenkant visas įmanomas kombinacijas. Dėl to, paskutiniame tiriamosios dalies etape palyginsime matricų visiško perrinkimo ir algebrinių lygčių sprendimo algoritmų vykdymo trukmes tarpusavyje. Apsvarstysime abiejų kriptoanalizei skirtų algoritmų pranašumus ir trūkumus.

2.2.4 ALGORITMŲ EFEKTYVUMO PALYGINIMAS

Paskutinis tiriamosios dalies skyrius skirtas sukurtų algoritmų efektyvumui palyginti. Kadangi sprendžiant matricines lygtis matricų visiško perrinkimo algoritmu galime gauti teisingą sprendinių skaičių virš baigtinio žiedo, o algebrinių lygčių algoritmu virš baigtinio lauko, fiksavus anksčiau apibrėžtus reikalavimus galime palyginti sprendinių suradimo trukmes. Naudojame antros ir trečios eilės matricas ($m = 2, m = 3$). Tokiam palyginimui fiksuojame 2.3 lentelėje juoda spalva pažymėtas p reikšmes ir prie jų nustatome sukurtų algoritmų vykdymo laiką. Rezultatai susiję su šių algoritmų palyginimu pateikti 2.4 lentelėje.

Pagal algoritmų vykdymo trukmių palyginimo lentelę matome, kad algebrinių lygčių sprendimo algoritmas yra vienareikšmiškai efektyvesnis nei matricų visiškas perrinkimas. Taigi, vykdymo laiko atžvilgiu pranašesnis pastarasis. Kita vertus, prisiminkime, kad algebrinių lygčių sprendimo algoritmą

galime vykdyti tik virš baigtinio lauko. Norėdami užtikrintai rasti sprendinius matricinėms lygtims virš baigtinio žiedo (kai p nebūtinai pirminis), vis dėlto turėtume vykdyti matricų visišką perrinkimą. Visiško perrinkimo pranašumas – sprendinius galime patikrinti matricines lygtis spręsdami su bet kokiais p. Dar vienas matricų visiško perrinkimo pranašumas, kad jį galime vykdyti su bet kokiomis tekstogramą atitinkančiomis matricomis X iš apibrėžtos algebrinės struktūros (žinomos tekstogramos ataka), o algebrinių lygčių sprendimo algoritmo atveju tekstogramą atitinkančios matricos X yra parenkamos (pasirinktos tekstogramos ataka).

2.4 lentelė. Algoritmų efektyvumo palyginimas

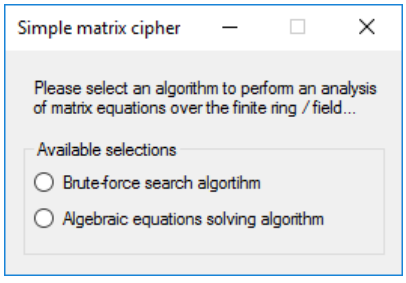
p	$m = 2$		$m = 3$	
	Matricų visiško perrinkimo alg. trukmė (sek.)	Algebrinių lygčių sprendimo alg. trukmė (sek.)	Matricų visiško perrinkimo alg. trukmė (sek.)	Algebrinių lygčių sprendimo alg. trukmė (sek.)
3	$2{,}84 \cdot 10^{-3}$	$1{,}11 \cdot 10^{-5}$	454,193417	$5{,}63 \cdot 10^{-5}$
5	0,144759	$1{,}86 \cdot 10^{-5}$	$4{,}472170 \cdot 10^{6}$	$9{,}20 \cdot 10^{-5}$
7	1,976933	$2{,}55 \cdot 10^{-5}$	$1{,}909075 \cdot 10^{9}$	$1{,}30 \cdot 10^{-4}$
11	67,534757	$4{,}22 \cdot 10^{-5}$	$6{,}518180 \cdot 10^{12}$	$2{,}04 \cdot 10^{-4}$
13	270,375116	$5{,}17 \cdot 10^{-5}$	$1{,}318370 \cdot 10^{14}$	$2{,}36 \cdot 10^{-4}$

Norint patikrinti algoritmų vykdymo trukmes prie kitų žiedo arba lauko dydį nusakančių p reikšmių rekomenduojame atkreipti dėmesį į 1 priedą. Jame pateikiami visų darbo grafikų duomenys.

3. PROGRAMINĖ REALIZACIJA

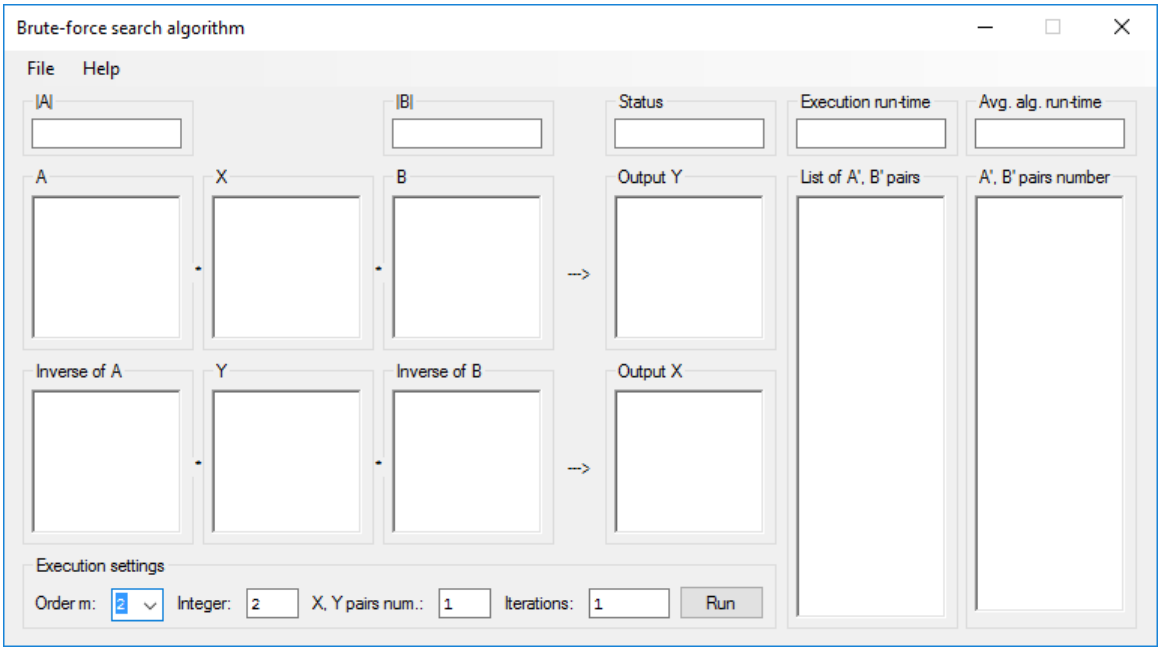
3.1. PROGRAMOS VARTOTOJO VADOVAS

Microsoft Visual Studio programavimo aplinka pasirinkta dėl savo tinkamumo tokio tipo uždaviniams spręsti ir jos panaudojimo išmanymo. Sukurta programinė įranga valdoma „*Windows Forms*“ pagalba. Vartotojo sąsajoje yra keletas išvesties langų, nustatymų ir juos paaiškinančių simbolių, su kuriais susipažinsime šiame skyrelyje. Tokio susipažinimo tikslas – suteikti galimybę sėkmingai dirbti kiekvienam vartotojui.

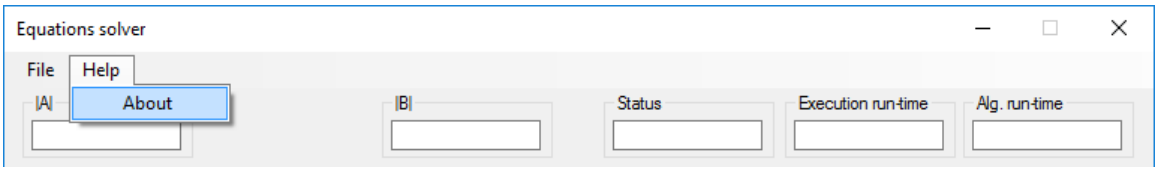


3.1 pav. Pradinis programos langas

Pradėjus darbą su programa atveriamas pradinis jos langas, pateiktas 3.1 pav. Viršutinėje kairėje jo dalyje matomas programos pavadinimas „Simple matrix cipher“. Čia pasiūloma pasirinkti vieną iš algoritmų, skirtų matricinių lygčių sprendiniams analizuoti. Galimi du pasirinkimai: Visiško perrinkimo algoritmas (angl. brute-force search algorithm) ir algebrinių lygčių sprendino algoritmas (angl. algebraic equations solving algorithm). Pirmiausiai pristatysime darbą su visiško perrinkimo algoritmu. Taigi, spragtelėjus pirmąjį pasirinkimą, atveriamas naujas langas. Jis pateiktas 3.2 pav. Viršutiniame dešiniajame kampe – standartiniai „*Windows Forms*“ mygtukai, skirti langui minimizuoti arba jį išjungti. Lango dydžio keitimo mygtukas yra neaktyvus, o lango dydžio redagavimas rankiniu būdu taip pat yra neleidžiamas. Viršutinėje meniu juostoje matomi mygtukai „File“ ir „Help“. „File“ → „Exit“ mygtuko paspaudimu, baigiamas darbas su programa. „Help“ → „About“ mygtuko paspaudimu išvedama informacija apie programos kūrėją. Tai iliustruota 3.3 pav.



3.2 pav. Vartotojo sąsaja skirta darbui su visiško perrinkimo algoritmu



3.3 pav. Darbas su meniu juosta

Apatinėje programos lango dalyje matoma „Execution settings“ nustatymų grupė. Tai vieta, kurioje vartotojas parenka ir koreguoja programos vykdymo parametrus. Pristatysime, kokius pasirinkimus vartotojas čia gali atlikti. „Order m” – tai matricų, su kuriomis dirbama, eilė. Kadangi darbo metu nagrinėjame tik antros ir trečios eilės matricas, šis nustatymas darbo eigoje visada bus

lygus dviem arba trim. Tačiau visiško perrinkimo algoritmą paliekama galimybė vykdyti ir su dar didesnės eilės matricomis. „Integer“ – tai sveikasis skaičius p, kuris apibrėžia sveikųjų skaičių žiedo $\mathbb{Z}_p$ (arba lauko) dydį. Vėliau, atsižvelgiant į pasirinktą $\mathbb{Z}_p$, elementai iš šios aibės bus surašomi į matricų eilutes ir stulpelius. „X, Y pairs num.“ – tai X, Y matricų porų skaičius n. Taip pat galime pasirinkti iteracijų skaičių – „Iterations“. Įvestas iteracijų skaičiaus nurodo, kiek kartų prie fiksuotų raktinių matricų A, B bus pilnai įvykdytas visiško perrinkimo algoritmas. Naujos iteracijos metu kinta įvesties ir išvesties matricų X, Y reikšmės. Paskutinis veiksmas, kurį turi atlikti vartotojas – tai programos paleidimas spaudžiant mygtuką „Run“. Jeigu „Execution settings“ nustatymų grupėje esantys nustatymai nebus koreguojami su programa dirbančio vartotojo, jie bus parenkami pagal nutylėjimą. Tokiu atveju, siūlomi patys primityviausi nustatymai, kurie matomi ir iliustracijoje – 3.2 pav. Po „Run“ mygtuko paspaudimo, sulaukus vykdymo pabaigos, galima keisti „Execution settings“ nustatymų grupės parametrų reikšmes ir neišjungiant programos papildomai atlikti kitus skaičiavimus.

Toliau apibūdinsime kitus 3.2 pav. matomus langus, kuriose atsispindi tik paskutinioji iteracija. Kitaip tariant, esant daugiau nei vienai iteracijai, šiuose languose matysime tik paskutinę. Šiek tiek žemiau meniu juostos, skaitant iš kairės į dešinę matomi A, X, B ir Y langai. Visi jie išdėstyti iš eilės, pagal slaptų, užšifravimui skirtų SMC veiksmų tvarką: $A \cdot X_i \cdot B \ (mod\ p) = Y_i$, kai $i = 1, \dots n$. Prie Y lango yra papildomas žodis „Output“, kuriuo norima pabrėžti, jog tai yra atliktų užšifravimo veiksmų rezultatas – išvestis. X langas skirtas nebūtinai tik vienos įvesties matricos atspausdinimui, o Y langas skirtas nebūtinai tik vienos išvesties matricos atspausdinimui (priklausomai nuo įvestos n reikšmės). A ir B langai skirti slaptoms unikalioms raktinėms matricoms. Čia jų visuomet bus tik po vieną. Virš matricų A, B langų yra papildomi laukeliai $|A|$, $|B|$, kuriose atspaudinami atitinkamų matricų determinantai. Jie vartotojui matyti yra svarbūs, kadangi reikalaujama, kad $|A| \neq 0$, $|B| \neq 0$.

Po A, X, B ir Y matricomis vartotojo sąsajoje yra langai, skirti paskutinės iteracijos iššifravimo veiksmui $A^{-1} \cdot Y_i \cdot B^{-1} \ (mod\ p) = X_i$ atlikti, kai $i = 1, \dots n$. Čia, „Inverse of A“ ir „Inverse of B“ languose atspausdinamos slaptų raktinių matricų modulinės atvirkštinės matricos A^{-1} ir B^{-1}.

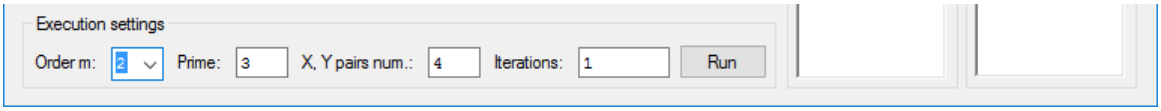
Svarbu atkreipti dėmesį, kad užšifruojant gauta išvesties matrica arba jų sąrašas Y_i, iššifruojant naudojami kaip įvestis, o gautas iššifravimo rezultatas sutampa su užšifravimo įvestimi X_i. Tokiam sutapimui patikrinti skirtas „Status“ langas, kuriame po programos įvykdymo išvedamas pranešimas „Success!” arba „Failure!“, informuojantis apie sėkmingą paskutinės iteracijos įvykdymą arba ne. Kadangi, programa veikia teisingai, natūralu, kad pranešimas „Failure!“ niekada nepasirodys, jei parametrai įvedami teisingai. Jo paskirtis tik užtikrinti sklandų vykdymo procesą. Be jo, vartotojui norint įsitikinti apie programos metu atliekamų veiksmų teisingumą, tektų atskirai tikrinti kiekvieną i-tąją X, Y porą tiek įvestyje, tiek išvestyje. Plačiau apie tai paaiškinsime eksperimentinius skaičiavimų skyriuje.

Dešinėje pusėje taip pat yra stačiakampio formos langas su pavadinimu „List of A', B' pairs“, kuriame atspausdinamos visos visiško perrinkimo algoritmo pagalba surastos alternatyvios raktinės matricos per paskutinę iteraciją. Tarp jų įtraukiamos ir pradinės raktinės matricos A, B.

Vartotojo sąsajos viršutinėje dešinėje pusėje – langai, kuriuose atsispindi informacija apie visas vykdymo metu atliktas iteracijas. „Execution run-time“ – bendras vykdymo laikas. Matuojant tokią trukmę įtraukiama visų procedūrų vykdymo trukmė, skaičiavimų atspausdinimo trukmė ir kt. Bendrai galima sakyti, kad tai visas užtrukęs laikas nuo „Run“ mygtuko paspaudimo pradžios iki vykdymo pabaigos. „Avg. alg. run-time“ – visiško perrinkimo algoritmo vienos iteracijos vykdymo trukmė. Jeigu vykdoma tik viena iteracija – „Execution run-time“ bus artima „Avg. alg. run-time“.

Pačioje dešiniausioje vartotojo sąsajos pusėje – atspausdinama statistika apie sugeneruotą A', B' porų skaičių kievienos sekančios iteracijos metu – „A', B' pairs number“ langas. Jis parodo, kiek skirtingų raktus atitinkančių matricų A', B' porų tenkina matricines lygtis kiekvienos kitos iteracijos metu. Šiame lange ir gaunami tinkami sprendiniai, kurie nagrinėjami atliekant tyrimus.

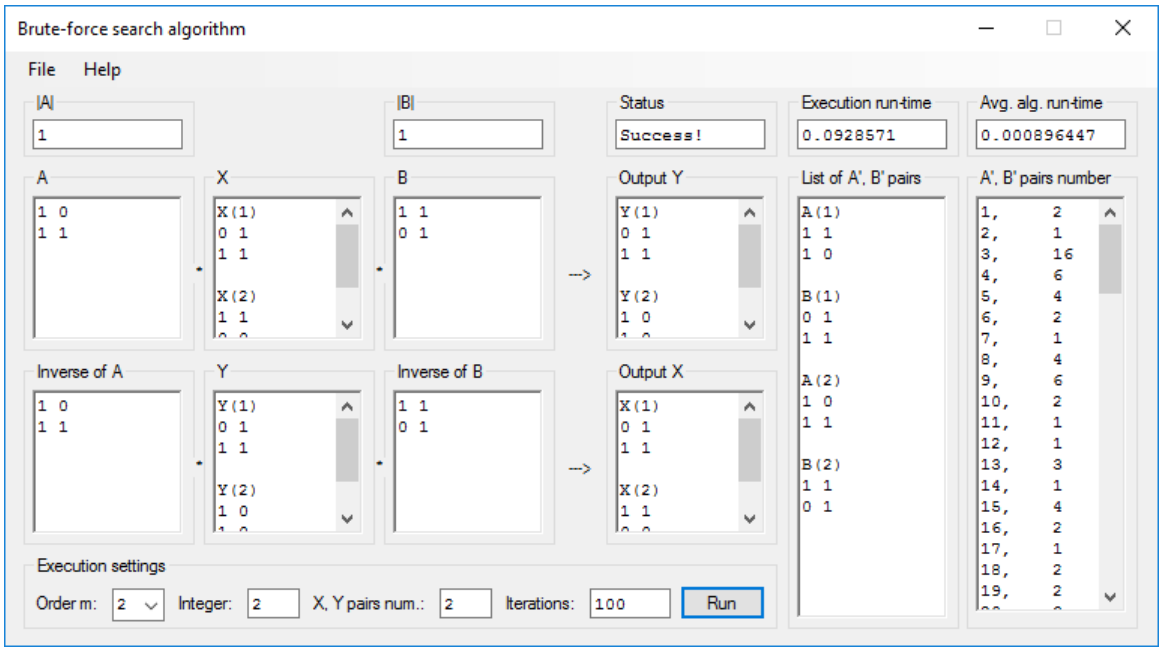
Norint atlikti matricinių lygčių virš baigtinio lauko sprendinių analizę su algebrinių lygčių sprendimo algoritmu pradiniame lange (3.1 pav.), renkamės antrą pasirinkimą. Atveriamas vartotojo sąsajos langas, kuris labai panašus į matricų visiško perrinkimo algoritmo 3.2 pav. Čia sakaičiavimai gali būti atlikti vadovaujantis beveik ta pačia tvarka, kaip ir matricų visiško perrinkimo atveju, taigi toliau paaiškinsime tik esminius skirtumus. Šiam tikslui pateikiame 3.4 pav. Čia „Order m” dalyje taip pat naudosime tik antros ir trečios eilės matricas. Tiesa, jei pasirenkamos antros eilės kvadratinės matricos, tuomet „X, Y pairs num.“ lange automatiškai parenkamos keturios įvesties matricos. Pasirinkus trečios eilės matricas „X, Y pairs num.“ lange automatiškai parenkamos devynios įvesties matricos. Čia kiti pasirinkimai negalimi, nes vykdoma tik pasirinktos tekstogramos ataka, dėl to ir matricų skaičius yra naudojamas pagal suformuluotus reikalavimus. Toliau matomas „Prime“ laukelis, kuriame įvestas sveikasis skaičius privalo būti pirminis. Tokiu būdu užtikriname, kad skaičiavimai bus atliekami su matricinėmis lygtimis virš baigtinio lauko. Algebrinių lygčių sprendimo atveju taip pat turime galimybę vienu metu vykdyti daugiau negu vieną iteraciją. Galiausiai algoritmo vykdymui spaudžiamas „Run“ mygtukas.



3.4 pav. Algebrinių lygčių sprendimo algoritmo nustatymai

3.2. EKSPERIMENTINIAI SKAIČIAVIMAI

Šiame skyriuje pateiksime matricų visiško perrinkimo ir algebrinių lygčių sprendimo algoritmų, skirtų lygčių sprendimui multiplikacinėse matricų pusgrupėse, eksperimentinius skaičiavimus ir jau aprašytos vartotojo sąsajos testavimo rezultatus. Tai galima sėkmingai padaryti teisingai užpildžius visus „Execution settings“ nustatymų grupės laukelius ir įvykdžius programą vieno „Run“ mygtuko paspaudimu.



3.5 pav. Programos langas įvykdžius programą

Pirmiausiai 3.5 pav. pateikiame programos lango iliustraciją po visiško perrinkimo algoritmo įvykdymo. Apatinėje programos lango dalyje matoma „Execution settings“ nustatymų grupė, kurioje jau įvesti nustatymai. Pristatysime, kokius pasirinkimus vartotojas parinko pateiktoje iliustracijoje. Matricų, su kuriomis dirbama eilės laukelyje „Order m“ įvesta 2 – tai nurodo, jog dirbama su antros eilės matricomis. Šis nustatymas darbo eigoje visada bus lygus dviem arba trim. „Integer“ arba sveikasis skaičius p lygus 2, kuris apibrėžia sveikųjų skaičių žiedą $\mathbb{Z}_2$. Atsižvelgiant į šiuos reikalavimus, elementai iš $\mathbb{Z}_2$ bus surašomi į antros eilės matricų eilutes ir stulpelius. „X, Y pairs num.“, kuris apibrėžia įvesties ir išvesties, t. y. X, Y matricų porų skaičių n, šiuo atveju parinktas 2. Pagal tai, ieškant alternatyvių raktinių matricų A', B' porų visiško perrinkimo algoritmu, tikrinama jau ne vieną, o dvi įvesties ir išvesties matricų poras X, Y. Suprantama, kad esant didesniam įvesties ir išesties matricų skaičiui, matricines lygtis tenkinančių matricų A', B' porų skaičius bus vis mažesnis. Mažesnis A', B' porų skaičius, tinkantis matricinėms lygtis išspręsti tuo pačiu reiškia didesnį saugumą. „Iterations“ laukelis skirtas nurodyti, kiek kartų prie fiksuotų raktinių matricų A, B bus vykdomas

algoritmas. Įvestas iteracijų skaičius lygus 100. Šiuo atveju, jis parinktas toks, kad rezultate būtų galima įžvelgti skirtumus tarp iteracijų. Paskutinis veiksmas, kurį turi atlikti vartotojas – tai programos paleidimas spaudžiant mygtuką „Run“. Primename, kad jeigu „Execution settings“ nustatymų grupėje esantys nustatymai nebus koreguojami su programa dirbančio vartotojo, jie bus parenkami pagal nutylėjimą. Po pirmojo „Run“ mygtuko paspaudimo, sulaukus vykdymo pabaigos, galima keisti „Execution settings“ nustatymų grupės parametrų reikšmes ir neišjungiant programos galima papildomai atlikti norimus skaičiavimus.

Dabar apibūdinsime kitus 3.5 pav. matomuose languose išvestus rezultatus, kuriuose atsispindi tik paskutinioji iteracija. Kitaip tariant, iš visų 100 įvykdytų iteracijų, šiuose languose matomi tik 100-tosios iteracijos rezultatai. Apibūdinsime šiek tiek žemiau meniu juostos esančius A, X, B ir Y langus, kurie išdėstyti iš eilės, pagal slaptų, užšifravimui skirtų veiksmų tvarką: $A \cdot X_i \cdot B \ (mod\ p) = Y_i$, kai $i = 1, 2$. X, Y languose matoma po dvi įvesties ir išvesties matricas, kadangi buvo parinktas n lygus 2. A ir B languose matomos slaptos unikalios raktinės matricos. Čia jų yra tik po vieną. Virš matricų A, B langų yra papildomi laukeliai $|A|$, $|B|$, kuriose atspaudinami atitinkamų matricų determinantai. Nesunku pastebėti, kad $|A| = 1 \neq 0$, $|B| = 1 \neq 0$, taigi reikalavimai išpildomi.

Žemiau A, X, B ir Y matricų vartotojo sąsajoje matomas paskutinės iteracijos metu atliktas tekstogramos iššifravimo veiksmas: $A^{-1} \cdot Y_i \cdot B^{-1} \ (mod\ p) = X_i$, kai $i = 1, 2$. Čia, „Inverse of A“ ir „Inverse of B“ languose jau atspausdintos slaptų raktinių matricų modulinės atvirkštinės matricos A^{-1} ir B^{-1}, kurios visų iteracijų metu išliks tokios pat.

Nesudėtinga pastebėti, kad per paskutinę iteraciją užšifruojant gautas išvesties matricų sąrašas Y_1, Y_2, iššifruojant naudojamas kaip įvestis, o gautas iššifravimo rezultatas sutampa su užšifravimo įvestimi X_1, X_2. Tokiam sutapimui patikrinti skirtas „Status“ praneša teigiamą rezultatą: „Success!“. Jei programa veiktų neteisingai, matytume pranešimą su neigiamu rezultatu: „Failure!“, tačiau kadangi parametrai įvesti teisingai, jis nepasirodo. Vartotojui norint patikrinti programos metu atliekamų veiksmų teisingumą savarankiškai, tektų atskirai tikrinti 1-mąją ir 2-ąją X, Y porą tiek įvestyje, tiek išvestyje ir įsitikinti, kad jos sutampa.

Dešinėje pusėje yra stačiakampio formos langas su pavadinimu „List of A', B' pairs“, kuriame atspausdinta matricų visiško perrinkimo algoritmo pagalba surastos A', B' matricų poros per paskutinę iteraciją. Šiuo atveju rasti du sprendiniai. Priklausomai nuo įvestų sisteminių parametrų čia gali pasirodyti skirtingas sprendinių skaičius.

Vartotojo sąsajos viršutinėje dešinėje pusėje – langai, kuriuose atsispindi informacija apie visas vykdymo metu atliktas iteracijas. „Execution run-time“ – bendras visos programos vykdymo laikas lygus 0,0928571 sekundės. Primename, kad į šią trukmę įtraukiama visų procedūrų vykdymo trukmė, skaičiavimų atspausdinimo trukmė ir kt. Kitaip tariant, tai visas užtrukęs laikas nuo „Run“ mygtuko paspaudimo pradžios iki vykdymo pabaigos. „Avg. alg. run-time“ trukmė 0,000896447 sekundės –

vidutinė visiško perrinkimo algoritmo vykdymo trukmė (per 100 iteracijų). Vykdant daugiau ar mažiau iteracijų vidutinė trukmė bus skaičiuojama pagal nurodytą iteracijų skaičių.

Dešiniausioje vartotojo sąsajos pusėje – renkama statistika apie surastą A', B' porų skaičių kiekvienos naujos iteracijos metu ir tai matoma „A', B' pairs number“ lange. Jis parodo, kiek skirtingų raktus atitinkančių matricų A', B' porų tenkina matricines lygtis kiekvienos kitos iteracijos metu: pirmos – 2, antros – 1 ir t. t.

Norint atlikti skaičiavimus algebrinių lygčių sprendimo algoritmu, eksperimentiniai skaičiavimai turėtų būti atliekami analogiškai, nes vartotojo vadovo sąsaja, yra praktiškai tokia pati. Čia tiesiog reikia nepamiršti suformuluotų reikalavimų ir pasirinkti teisingus sisteminius parametrus pagal vartotojo vadovo dalyje aprašytą tvarką. Taigi, pristatėme vartotojo sąsają, kuria dabar galėtų naudotis kiekvienas vartotojas. Trumpai apžvelgėme, kaip reikėtų atlikti norimus eksperimentinius skaičiavimus bei suprasti jų metu atspausdinamus rezultatus tam skirtose vietose. Panašiu būdu, abiem realizuotais algoritmais, rezultatai nustatinėjami viso tyrimo eigoje. Tiesiog pagal tam tikrą strategiją tikrinami įvairesni „Execution settings“ parametrų rinkiniai ir pagal tai atliekamas didesnės apimties tyrimas.

DISKUSIJA

Keliais sakiniais apibendrinsime svarbiausius rezultatus. Sukurtas ir programiškai realizuotas paprastas matricų šifras SMC suformuoja matricinių lygčių sistemą su nežinomais raktiniais A, B elementais virš baigtinio žiedo arba lauko, o ši sistema turi daugiau negu vieną sprendinį. Sprendiniams surasti egzistuoja daugiau negu vienas būdas, dėl to ištyrėme keletą pasirinktų. Pirmiausiai matricų visiško perrinkimo algoritmu spręsdami matricines lygtis suradome teisingų sprendinių skaičių prie skirtingų parametrų p, kurie apibrėžia žiedo $\mathbb{Z}_p$ dydį. Toks sprendinių radimas vadinamas žinomos tekstogramos ataka. Surastų sprendinių skaičių pagrindėme teoriškai. Atlikdami teorinį pagrindimą nustatėme, kad tinkamų sprendinių skaičius lygus atvirkštinių elementų skaičiui nagrinėjamame žiede.

Vykdydami matricų visišką perrinkimą įsitikinome, kad sprendinių skaičius prie tų pačių parametrų rinkinių priklauso nuo pačių įvesties ir išvesties matricų. Naudojome antros ir trečios eilės kvadratines matricas. Čia tyrimus būtų galima atlikti ir su dar didesnės eilės matricomis, tiesa vykdymo laikas augtų labai greitai ir greitai taptų nepriimtinas.

Tyrimo eigoje pastebėjome, kad su daugiau tekstogramą atitinkančių įvesties ir šifrogramą atitinkančių išvesties matricų, sprendinių skaičius matricinėse lygtyse mažėja. Galiausiai sprendinių skaičius prie konkrečių p nusistovi ir didinant įvesties ir išvesties matricų porų kiekį jis nebekinta. Tokiu būdu gaunami visiškai tinkačių bet kokioms įvesties ir išvesties matricų poroms sprendiniai. Dėl šios priežasties atliekant matricų visišką perrinkimą vertėtų naudoti didesnes tekstogramas ir

šifrogramas, jeigu norime gauti tik tinkamus sprendinius. Atliekant šį tyrimą išmatavome matricų visiško perrinkimo algoritmo efektyvumą, kurį įdomu palyginti su kitų egzistuojančių metodų efektyvumu.

Sukūrėme ir kitą kriptoanalizės metodą tyrimui atlikti – algebrinių lygčių sprendimo algoritmą. Šiuo atveju matricinės lygtys sprendžiamos tik baigtinio lauko atveju, kai p – pirminis skaičius. Tokio pobūdžio algoritmą vykdyti baigtinio žiedo atveju būtų komplikuota, nes kai kurie elementai gali netenkinti reguliarumo sąlygos arba tiesiog neturėti atvirkštinių. Jei taip nutiktų algoritmo surasti sprendiniai būtų klaidingi.

Algebrinių lygčių sprendimas atlieka pasirinktos tekstogramos ataką, o tai reiškia, jog įvesties matricas nurodome patys. Papildomai būtų galima pamėginti automatizuoti algebrinių lygčių sudarymą su bet kokiomis įvesties ir išvesties matricomis. Tada tyrimus galėtume atlikti platesniu atveju – vykdant žinomos tekstogramos ataką (kaip ir matricų visiško perrinkimo atveju).

Tyrimo eigoje alternatyviu algoritmu surasti antros ir trečios eilės matricų sprendiniai tinka bet kokiam baigtiniam laukui. Algebrinių lygčių sprendimo atveju formuluojama daugiau reikalavimų, tačiau jo efektyvumas ženkliai geresnis. Pastebėjome, kad efektyvesnių algoritmų kūrimui reikalingas geras kriptografijos ir kriptoanalizės išmanymas. Geras kriptoanalizės supratimas dažniausiai padeda sukurti saugesnius kriptografinius algoritmus.

Panašios į šiame tyrime sudaromas kelių kintamųjų lygtis (MQ) gali būti gautos matricinio laipsnio šifre, kuriame yra naudojama matricinio laipsnio funkcija (Lukšys ir Sakalauskas, 2012). Matricinio laipsnio funkcijos idėja yra panaši į sukurtą paprasto matricų šifro SMC šifravimo būdą, dėl to metodai kuriais galime surasti SMC paslėptą informaciją, bus tinkami ir matricinio laipsnio šifro sprendinių radimui.

IŠVADOS

Pagal atliktus tyrimus, suformulavome pačias svarbiausias išvadas:

- Paprastas matricų šifras SMC suformuoja matricines lygtis su nežinomais raktiniais A, B elementais virš baigtinio žiedo arba lauko, kurios turi daugiau negu vieną sprendinį.
- Matricų visiško perrinkimo algoritmu atlikdami žinomos tekstogramos ataką suradome sprendinių skaičių matricinėse lygtyse virš skirtingų baigtinių žiedų $\mathbb{Z}_p$. Didinant tekstogramas ir šifrogramas (įvesties ir išvesties matricų X, Y skaičių n), matricines lygtis tenkinančių sprendinių A', B' skaičius mažėja.
- Matricų visiško perrinkimo algoritmu nustatėme, kad prie konkrečių p sprendinių skaičius nusistovi naudojant $n = 7$ X, Y poras ir daugiau nebekinta. Tokiu būdu gauti sprendiniai yra visiškai tinkami bet kokioms tekstogramoms ir šifrogramoms.

- Tinkamų matricinėms lygtims išspręsti A', B' sprendinių skaičius priklauso ir nuo pačių tekstogramų bei šifrogramų (įvesties ir išvesties matricų X, Y) elementų reikšmių.
- Atlikdami sprendinių skaičiaus teorinį pagrindimą nustatėme, kad teisingų sprendinių skaičius lygus atvirkštinių elementų skaičiui tame žiede.
- SMC šifro raktus atitinkančius sprendinius įmanoma surasti algebriniu būdu, sprendžiant netiesinių lygčių sistemas. Kelių kintamųjų (MQ) lygtis galima pertvarkyti į tiesines ir tokiu būdu gauti teisingus sprendinius.
- Algebrinių lygčių sprendimo algoritmo atveju nustatytos sprendinių išraiškos (antros ir trečios eilės matricoms) tinka bet kokiam baigtiniam laukui.
- Algebrinių lygčių sprendimo algoritmas vykdant pasirinktos tekstogramos ataką (matricinėse lygtyse virš baigtinio lauko), yra ženkliai efektyvesnis nei matricų visiško perrinkimo algoritmas vykdant žinomos tekstogramos ataką (matricinėse lygtyse virš baigtinio žiedo).

REKOMENDACIJOS

Papildomai pateikiame keletą rekomendacijų, susijusių su tolimesniu šios temos plėtojimu arba čia aprašyto darbo patobulinimu.

Norint surasti visiškai tinkančių sprendinių skaičių, vykdant mažai ar tik vieną iteraciją, matricų visiško perrinkimo atveju rekomenduojama naudoti didelį įvesties ir išvesities matricų skaičių. Geriausiai, naudoti daugiau kaip 7 įvesties ir išvesties matricų poras.

Greitesniam veiksmų atlikimui matricinėse lygtyse, būtų galima naudoti Volker Strassen matricų daugybą, kuri laikoma spartesne bei galimai padidintų SMC šifro matricų daugybos efektyvumą (Strassen, 1969).

Šiame darbe sprendinių analizę atlikome su antros ir trečios eilės kvadratinėmis matricomis. Čia tyrimus būtų galima atlikti ir su dar didesnės eilės matricomis, tiesa vykdymo laikas augtų labai greitai ir tyrimas būtų labiau komplikuotas vykdymo trukmės atžvilgiu.

Algebrinių lygčių sprendimo algoritmo atveju galima pamėginti automatizuoti algebrinių lygčių sudarymą su bet kokiomis įvesties ir išvesties matricomis. Tuomet sprendinių analizės tyrimus galėtume atlikti vykdant žinomos tekstogramos ataką (kaip ir matricų visiško perrinkimo atveju).

Sukurtus algoritmus galėtume pritaikyti sprendinių paieškai matricinio laipsnio šifre, kuriame naudojama matricinio laipsnio funkcija.

Pagal gautus rezultatus ir suformuluotas išvadas rekomenduojama toliau plėtoti šią temą ir sukurti daugiau kriptoanalizės algoritmų, kurie padėtų užtikrinti saugesnius šifravimo metodus. Tuomet siūloma atlikti papildomą SMC šifro sprendinių analizę bei palyginti naujo algoritmo efektyvumą su jau realizuotais.

Dar keletas pastebėjimų. Nereikėtų pamiršti, jog skaičiavimai atlikti vieno personalinio kompiuterio galimybių ribose. Kuomet matuojama matricų visiško perrinkimo ar algebrinių lygčių sprendimo algoritmo vykdymo trukmė, reikėtų atkreipti dėmesį, kad tyrimus atliekant skirtinguose personaliniuose kompiuteriuose gali atsirasti paklaidos. Dėl to tyrimą vertėtų atlikti daugiau kaip vienu kompiuteriu ir palyginti rezultatus. Kita vertus, proporcijos išliktų.

Tyrimo metu stengėmės optimizuoti programos išeities tekstą, rezultatus siekiant gauti kaip įmanoma efektyviau. Įsitikinome, kad visiškas programos išeities teksto optimizavimas yra sudėtingas uždavinys.

ŠALTINIAI IR LITERATŪRA

1. Biham, E. ir Shamir, A. (1991). Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology, 4(1), 3-72.
2. Campbell, H. (1971). Linear Algebra With Applications.
3. Cauer, E., Wolfgang, M. ir Rainer, P. (2000). Life and work of Wilhelm Cauer. Perpignan, France.
4. Cheon, J. ir Lee, D. (2004). Resistance of S-boxes against Algebraic Attacks. Fast Software Encryption, LNCS, vol. 3017 (83-94). Springer-Verlag.
5. Dosinas, G. S. ir Navickas, Z. (2010). Algebrinių tiesinių operatorių struktūros.
6. Garey, M. R. ir Johnson, D. S. (1979). Computers and Intractability. A Guide to the Theory of NP-Completeness. W.H. Freeman and Company.
7. Glanville, R. (2009). Black Boxes, Cybernetics and Human Knowing.
8. Hill, L., S. (1929). Cryptography in an Algebraic Alphabet, The American Mathematical Monthly Vol.36, June–July 1929, p. 306–312.
9. Horn, R. A. ir Johnson, C. R. (2013). Matrix Analysis (2nd ed.). Cambridge University Press.
10. Long, C. T. (1972). Elementary Introduction to Number Theory (2nd ed.), Lexington.
11. Lukšys, K. (2013). Matricinio laipsnio šifras ir jo analizė.
12. Lukšys, K. (2014). Algebrinės struktūros, pratybų medžiaga.
13. Lukšys, K. ir Sakalauskas, E. (2012). Matrix Power Cipher. Information Technology and Control, 41(4), 349-355.
14. Mahmoud, A. Y. ir Chefranov, A. G. (2010). Secure Hill cipher modifications and key exchange protocol. In Proceedings of the 2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR) - Volume 02 (AQTR '10), vol. 2. IEEE Computer Society, 1-6.
15. Meier, W., Pasalic, E. ir Carlet, C. (2004). Algebraic attacks and decomposition of Boolean functions. Advances in Cryptology - EUROCRYPT 2004, LNCS 3027 474–491. Springer-Verlag.
16. Menezes, A. J., van Oorschot, P. C. ir Vanstone, S. A (2005). Handbook of Applied Cryptography.
17. Oppliger, R. (2005). Contemporary Cryptography. Norwood: Artech House Publisher.
18. Reynard, R. (1997). Secret Code Breaker II: A Cryptanalyst's Handbook. Jacksonville, FL.
19. Rosen, K. H. (1993). Elementary Number Theory and its Applications (3rd ed.), Addison-Wesley.
20. Sajavičius, S. (2003). Matematikos akiračiai. Lyginiai.

21. Sakalauskas, E., Listopadskis, N. ir Dosinas, G. S. (2008). Kriptografijos teorija.
22. Sakalauskas, E., Listopadskis, N., Dosinas, G. S., Lukšys, K. ir Katvickis, A. (2008). Kriptografinės sistemos.
23. Sakalauskas., E. (2012). The Multivariate Quadratic Power Problem over Zn is NP-complete. Information Technology and Control, 41(1), 33-39.
24. Sastry, V. U. K. ir Samson, Ch. (2012). Generalized Hill Cipher Involving Multiple Keys, Mixing and KeyDependent Substitution. International Journal of Computational Intelligence and Information Security, 3(6), 11-25.
25. Schaumüller-Bichl, I. (1983). Cryptanalysis of the Data Encryption Standard by the Method of Formal Coding. Cryptography: Proceedings of the Workshop on Cryptography, LNCS, vol. 149. Berlin: Springer, 235-255.
26. Schneier, B. (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C, (2nd ed.). Wiley.
27. Shannon, C. (1949). Communication Theory of Secrecy Systems. Bell System Technical Journal, vol.28-4, 656-715.
28. Strang, G. (2003). Introduction to linear algebra (3rd ed.).
29. Strassen, V. (1969). Gaussian Elimination is not Optimal, Numer. Math. 13, p. 354-356.
30. Thomae, E. ir Wolf, C. (2010). Solving Systems of Multivariate Quadratic Equations over Finite Fields or: From Relinearization to MutantXL. Cryptology ePrint Archive.
31. van Tilborg, H. C. (2005). Encyclopedia of Cryptography and Security. NY: Springer.

1 PRIEDAS. GRAFIKŲ DUOMENYS

1.1 pr. lentelės. Matricų visiško perrinkimo alg. grafikų duomenys

$m = 2, n = 4$			
A', B' sk.	p	Trukmė (s.)	Variantų sk.
1	2	0.0001441	256
2	3	0.002844673	6561
2	4	0.026598126	65536
4	5	0.144759602	390625
2	6	0.60495	1679616
6	7	1.9767332	5764801
4	8	5.9213895	16777216
6	9	13.8487825	43046721
4	10	32.4438953	100000000
10	11	67.5347575	214358881
4	12	152.6925528	429981696
12	13	270.3751165	815730721
6	14	456.9223931	1475789056
8	15	778.6677187	2562890625

$m = 3, n = 9$			
A', B' sk.	p	Trukmė (s.)	Variantų sk.
1	2	0.3073252	262144
2	3	454.1934176	387420489
2	4	80563.45723	68719476736
4	5	4472170.258	3.8147E+12
2	6	119064079.3	1.0156E+14
6	7	1909074916	1.62841E+15
4	8	21119226932	1.80144E+16
6	9	1.75964E+11	1.50095E+17
4	10	1.17235E+12	1E+18
10	11	6.51818E+12	5.55992E+18
4	12	3.12119E+13	2.66233E+19
12	13	1.31837E+14	1.12455E+20
6	14	5.00453E+14	4.26879E+20
8	15	1.73261E+15	1.47789E+21

1.2 pr. lentelės. Algebrinių lygčių sprendimo alg. grafikų duomenys

$m = 2, n = 4$			
A', B' sk.	p	Trukmė (s.)	Variantų sk.
2	3	1.11E-05	6561
4	5	1.86E-05	390625
6	7	2.55E-05	5764801
10	11	4.22E-05	214358881
12	13	5.17E-05	815730721
16	17	6.33E-05	6975757441
18	19	7.21E-05	16983563041
22	23	8.69E-05	78310985281
28	29	0.000111454	5.00246E+11
30	31	0.000120616	8.52891E+11
36	37	0.000140056	3.51248E+12
40	41	0.000155457	7.98493E+12
42	43	0.000162379	1.16882E+13
46	47	0.000176344	2.38113E+13

$m = 3, n = 9$			
A', B' sk.	p	Trukmė (s.)	Variantų sk.
2	3	5.63E-05	387420489
4	5	9.20E-05	3.8147E+12
6	7	1.30E-04	1.62841E+15
10	11	2.04E-04	5.55992E+18
12	13	2.36E-04	1.12455E+20
16	17	0.00031895	1.40631E+22
18	19	0.000360834	1.04127E+23
22	23	0.000430033	3.24415E+24
28	29	0.000556516	2.10457E+26
30	31	0.000585061	6.99054E+26
36	37	0.000696815	1.68901E+28
40	41	0.000762576	1.07179E+29
42	43	0.000813188	2.52599E+29
46	47	0.000887988	1.25245E+30

Pastaba: lentelėse juoda spalva pažymėti duomenys skirti algoritmų efektyvumo palyginimui.

1.3 pr. lentelė. Vidutinis sprendinių A', B' skaičius prie skirtingų n

$m = 2$, iteracijų skaičius: 100						
n	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 6$	$p = 7$
1	25.04	153.7	391.36	1045.6	2105.82	3750.6
2	8.09	12	15.96	20.04	22.52	46.08
3	3.69	3.18	4	4.8	4.82	6.3
4	1.1	2.28	2.2	4	2.3	6
5	1	2.02	2.12	4	2.1	6
6	1	2	2	4	2.06	6
7	1	2	2	4	2	6
8	1	2	2	4	2	6
9	1	2	2	4	2	6
10	1	2	2	4	2	6

2 PRIEDAS. PAGRINDINIŲ PROCEDŪRŲ IŠEITIES TEKSTAS

```
// Generate random matrix
public static double[,] RandomMatrix(int m, int p, Random rnd)
{
    double[,] no = new double [m, m];

    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < m; j++)
        {
            no[i, j] = rnd.Next(0, p);
        }
    }
    return no;
}


// Display square matrix
public static string DisplayMatrix(int m, double[,] M)
{
    string matrixString = "";
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < m; j++)
        {
            matrixString += M[i,j].ToString();
            matrixString += " ";
        }
        matrixString += Environment.NewLine;
    }
    return matrixString;
}


// Multiply matrices
public static double[,] MatrixMultiplication(int m, int p, double[,] M1, double[,] M2)
{
    double[,] R; // result

    R = new double[m, m];
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < m; j++)
        {
            R[i, j] = 0;
            for (int k = 0; k < m; k++)
            {
                R[i, j] = (R[i, j] + M1[i, k] * M2[k, j]) % p;
            }
        }
    }
    return R;
}
// Compute determinant
public static double Determinant(double[,] M)
{
    double det = 0;
    if (M.GetLength(0) == 1)
    {
        det = M[0, 0];
    }
    if (M.GetLength(0) == 2)
    {
        det = M[0, 0] * M[1, 1] - M[0, 1] * M[1, 0];
    }
    else
    {
        for (int i = 0; i < 1; i++)
        {
            for (int j = 0; j < M.GetLength(1); j++)
            {
                double subdet = Determinant(FillNewArray(M, i, j));
                if (j % 2 != 0) subdet *= -1;
                det += M[i, j] * subdet;
```

```
            }
        }
    }
    return det;
}


// Fill array
public static double[,] FillNewArray(double[,] originalArr, int row, int col)
{
    double[,] tempArray = new double[originalArr.GetLength(0) - 1, originalArr.GetLength(1) - 1];

    for (int i = 0, newRow = 0; i < originalArr.GetLength(0); i++)
    {
        if (i == row)
            continue;
        for (int j = 0, newCol = 0; j < originalArr.GetLength(1); j++)
        {
            if (j == col)
                continue;
            tempArray[newRow, newCol] = originalArr[i, j];
            newCol++;
        }
        newRow++;
    }
    return tempArray;
}


// Calculate modular inverse of an integer
public static int ModInverse(int a, int n)
{
    int i = n, v = 0, d = 1;
    if (a < 0)
    {
        a = (int)(Mod(a, n));
    }

    while (a > 0)
    {
        int t = i / a, x = a;
        a = i % x;
        i = x;
        x = d;
        d = v - t * x;
        v = x;
    }
    v %= n;
    if (v < 0)
        v = (v + n) % n;
    return v;
}


// Calculate modulus of an integer
public static double Mod(double a, int n)
{
    a %= n;
    if (a < 0)
    {
        a = (a + n) % n;
    }
    return a;
}

// Find next matrix in lexicographic order
public static bool NextMatrix(int p, double[,] M)
{
    int i = 0;
    int j = 0;
    int m = M.GetLength(0);
    bool ch = true;
    while (ch == true)
    {
        M[i, j] = M[i, j] + 1;
        if (M[i, j] < p)
        {
```

```
            ch = false;
        }
        if (M[i, j] == p)
        {
            M[i, j] = 0;
            j = j + 1;
            if (j == m)
            {
                j = 0;
                i = i + 1;
                if (i == m)
                {
                    return false; // back to initial matrix
                }
            }
        }
    }
    return true; // element is changed
}


// Check if two matrices are equal
public static bool MatricesAreEqual(double[,] M1, double[,] M2)
{
    int m = M1.GetLength(0);
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (M1[i, j] != M2[i, j])
            {
                return false;
            }
        }
    }
    return true;
}


// Check if two matrices are equal
public static string Verification(string S1, string S2)
{
    if (S1 == S2)
    {
        return "Success!";
    }
    return "Failure!";
}

// Assign second matrix to the first one
public static double[,] AssignMatrix(double[,] M1, double[,] M2)
{
    int m = M1.GetLength(0);
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < m; j++)
        {
            M1[i, j] = M2[i, j];
        }
    }
    return M1;
}


// Compute Greatest Common Divisor
public static int GCD(int a, int b)
{
    int Remainder;
    while (b != 0)
    {
        Remainder = a % b;
        a = b;
        b = Remainder;
    }
    return a;
}
```

```
        // Genereate matrix with needed requirements
        public static double[,] MatrixSatisfyRequirements(int p, int m, Random r)
        {
            double[,] M = RandomMatrix(m, p, r);
            while (GCD((int)Mod(Determinant(M), p), p) != 1)
            {
                M = RandomMatrix(m, p, r);
            }
            return M;
        }

        // Remove specific row and column of a matrix
        public static double[,] TrimArray(int rowToRemove, int columnToRemove, double[,] M)
        {
            double[,] result = new double[M.GetLength(0) - 1, M.GetLength(1) - 1];

            for (int i = 0, j = 0; i < M.GetLength(0); i++)
            {
                if (i == rowToRemove)
                    continue;

                for (int k = 0, u = 0; k < M.GetLength(1); k++)
                {
                    if (k == columnToRemove)
                        continue;
                    result[j, u] = M[i, k];
                    u++;
                }
                j++;
            }
            return result;
        }


        // Find modular inverse of a matrix
        public static double[,] ModularInverseMatrix(int p, double[,] M)
        {
            int m = M.GetLength(0);
            double[,] Minor = new double[m - 1, m - 1];
            double[,] Inversed = new double[m, m];
            double ModDet = ModInverse((int)Determinant(M), p);
            for (int i = 0; i < m; i++)
            {
                for (int j = 0; j < m; j++)
                {
                    Minor = TrimArray(i, j, M);
                    double Adjoint = Math.Pow(-1, (i + 1) + (j + 1)) * Determinant(Minor);
                    Inversed[j, i] = Mod(Adjoint * ModDet, p);
                }
            }
            return Inversed;
        }


        // Brute force and find A, B pairs of matrices
        public static void BruteForce(Random r, int p, double[,] A, double[,] B, int num,
            out List<double[,]> XMatrices, out List<double[,]> YMatrices, out List<double[,]> ABPairs, out int
ABCount)
        {
            int m = A.GetLength(0);
            double[][,] X = new double[num][,];
            double[][,] AX = new double[num][,];
            double[][,] Y = new double[num][,];
            double[,] KeyA = new double[m, m];
            double[,] KeyB = new double[m, m];
            double[][,] KeyAX = new double[num][,];
            double[,] Y_tmp;
            XMatrices = new List<double[,]>();
            YMatrices = new List<double[,]>();
            ABPairs = new List<double[,]>();
            ABCount = 0;

            for (int i = 0; i < num; i++)
            {
                X[i] = RandomMatrix(m, p, r);
                AX[i] = MatrixMultiplication(m, p, A, X[i]);
                Y[i] = MatrixMultiplication(m, p, AX[i], B);
```

```
            XMatrices.Add(X[i]);
            YMatrices.Add(Y[i]);
        }

        while (NextMatrix(p, KeyA) == true)
        {
            for (int i = 0; i < num; i++)
            {
                KeyAX[i] = MatrixMultiplication(m, p, KeyA, X[i]);
            }
            while (NextMatrix(p, KeyB) == true)
            {
                bool match = true;
                for (int i = 0; i < num; i++)
                {
                    Y_tmp = MatrixMultiplication(m, p, KeyAX[i], KeyB);
                    if (MatricesAreEqual(Y[i], Y_tmp) == false)
                    {
                        match = false;
                        break;
                    }
                }
                if (match == true)
                {
                    double[,] newKeyA = new double[m, m];
                    double[,] newKeyB = new double[m, m];
                    newKeyA = AssignMatrix(newKeyA, KeyA);
                    newKeyB = AssignMatrix(newKeyB, KeyB);
                    ABPairs.Add(newKeyA);
                    ABPairs.Add(newKeyB);
                    ABCount += 1;
                }
            }
        }
    }


    // Display matrices pairs list
    public static string DisplayMatrixPairsList(int m, List<double[,]> L, string ListString)
    {
        int Count = 1;
        int CountA = 1;
        int CountB = 1;
        ListString = null;
        foreach (double[,] mat in L)
        {
            if (Count % 2 != 0)
            {
                ListString += "A(" + CountA.ToString() + ")" + "\n";
                CountA += 1;
            }
            if (Count % 2 == 0)
            {
                ListString += "B(" + CountB.ToString() + ")" + "\n";
                CountB += 1;
            }
            ListString += DisplayMatrix(m, mat) + "\n";
            Count += 1;
        }
        Count = Count / 2;
        return ListString;
    }


    // Display matrices list
    public static string DisplayMatrixList(int m, List<double[,]> L, string name)
    {
        int CountA = 1;
        string ListString = null;
        foreach (double[,] mat in L)
        {
            ListString += name + "(" + CountA.ToString() + ")" + "\n";
            CountA += 1;
            ListString += DisplayMatrix(m, mat) + "\n";
        }
        return ListString;
    }
```

```
        // Shape list of output matrices
        public static List<double[,]> ShapeOutputList(int p, double[,] ModInvA, double[,] ModInvB,
List<double[,]> YMat)
        {
            int m = ModInvA.GetLength(0);
            List<double[,]> OutputList = new List<double[,]>();
            foreach (double[,] mat in YMat)
            {
                double[,] OutputX = MatrixMultiplication(m, p, MatrixMultiplication(m, p, ModInvA, mat),
ModInvB);
                OutputList.Add(OutputX);
            }
            return OutputList;
        }


        // Display numbers of A, B pairs
        public static string DisplayPairsNumbers(List<int> L)
        {
            string ListString = null;
            int index = 1;
            foreach (int num in L)
            {
                ListString += index.ToString() + ",\t" + num.ToString() + "\n";
                index += 1;
            }
            return ListString;
        }


        // execution button
        private void button1_Click(object sender, EventArgs e)
        {
            // start measuring execution time
            Stopwatch ExecutionTimer = new Stopwatch();
            Stopwatch AlgTimer = new Stopwatch();
            ExecutionTimer.Start();

            // parse matrix order and integer
            int m = int.Parse(comboBox1.Text);
            int p = int.Parse(textBox1.Text);

            Random r = new Random();
            // generate non singular matrix A
            double[,] A = MatrixSatisfyRequirements(p, m, r);
            // display matrix A and det(A)
            richTextBox2.Text = DisplayMatrix(m, A);
            textBox10.Text = Mod(Determinant(A), p).ToString();

            // generate non singular matrix B
            double[,] B = MatrixSatisfyRequirements(p, m, r);
            // display matrix B and det(B)
            richTextBox4.Text = DisplayMatrix(m, B);
            textBox11.Text = Mod(Determinant(B), p).ToString();

            // generate modular inverse of matrix A
            double[,] ModInvA = ModularInverseMatrix(p, A);
            richTextBox12.Text = DisplayMatrix(m, ModInvA);

            // generate modular inverse of matrix B
            double[,] ModInvB = ModularInverseMatrix(p, B);
            richTextBox14.Text = DisplayMatrix(m, ModInvB);

            // parse number of X and Y pairs
            int num = int.Parse(textBox7.Text);
            int iterations = int.Parse(textBox4.Text);

            // iterate brute force and find A, B pairs
            AlgTimer.Start();
            List<int> PairsNumbersList = new List<int>();
            for (int i = 0; i < iterations; i++)
            {
                BruteForce(r, p, A, B, num, out XMat, out YMat, out ABPairs, out ABCount);
                PairsNumbersList.Add(ABCount);
            }
            AlgTimer.Stop();
```

```
            richTextBox3.Text = DisplayPairsNumbers(PairsNumbersList);

            // measure average algorithm time
            double AvgTime = AlgTimer.Elapsed.TotalSeconds / iterations;
            textBox6.Text = AvgTime.ToString();

            // last iteration
            // display matrices lists of last iteration
            richTextBox7.Text = DisplayMatrixList(m, XMat, "X");
            richTextBox9.Text = richTextBox11.Text = DisplayMatrixList(m, YMat, "Y");

            // display list of A', B' pairs of last iteration
            string ABListString = null;
            richTextBox1.Text = DisplayMatrixPairsList(m, ABPairs, ABListString);

            // shape and display output X list of last iteration
            List<double[,]> OutputXList = ShapeOutputList(p, ModInvA, ModInvB, YMat);
            richTextBox13.Text = DisplayMatrixList(m, OutputXList, "X");

            // verify that output is correct
            textBox13.Text = Verification(DisplayMatrixList(m, XMat, "X"), DisplayMatrixList(m, OutputXList,
"X"));

            // finish measuring execution time
            ExecutionTimer.Stop();
            textBox12.Text = ExecutionTimer.Elapsed.TotalSeconds.ToString();
        }


        // Generate random matrix
        public static double[,] RandomKeyMatrix(int m, int p, Random rnd)
        {
            double[,] no = new double[m, m];

            for (int i = 0; i < m; i++)
            {
                for (int j = 0; j < m; j++)
                {
                    no[i, j] = rnd.Next(1, p);
                }
            }
            return no;
        }


        public static void SolveEquations2x2(Random r, int p, double[,] A, double[,] B,
            out List<double[,]> XMatrices, out List<double[,]> YMatrices, out List<double[,]> ABPairs, out int
ABCount)
        {
            int m = A.GetLength(0);
            double[][,] X = new double[4][,];
            double[][,] KeyAX = new double[4][,];
            double[,] Y_tmp;
            double[,] KeyA = new double[m, m];
            double[,] KeyB = new double[m, m];
            XMatrices = new List<double[,]>();
            YMatrices = new List<double[,]>();
            ABPairs = new List<double[,]>();
            ABCount = 0;

            X[0] = new double[,] { { 1, 0 }, { 0, 0 } };
            X[1] = new double[,] { { 0, 1 }, { 0, 0 } };
            X[2] = new double[,] { { 0, 0 }, { 1, 0 } };
            X[3] = new double[,] { { 0, 0 }, { 0, 1 } };

            double[][,] AX = new double[4][,];
            double[][,] Y = new double[4][,];
            for (int i = 0; i < 4; i++)
            {
                AX[i] = MatrixMultiplication(m, p, A, X[i]);
                Y[i] = MatrixMultiplication(m, p, AX[i], B);
                XMatrices.Add(X[i]);
                YMatrices.Add(Y[i]);
            }
            int j = 1;
            while (j < p)
            {
```

```
            A[0, 0] = j;
            B[0, 0] = (Y[0][0, 0] * ModInverse((int)A[0, 0], p)) % p;
            B[0, 1] = (Y[0][0, 1] * ModInverse((int)A[0, 0], p)) % p;
            B[1, 0] = (Y[1][0, 0] * ModInverse((int)A[0, 0], p)) % p;
            B[1, 1] = (Y[1][0, 1] * ModInverse((int)A[0, 0], p)) % p;
            A[0, 1] = (A[0, 0] * Y[2][0, 0] * ModInverse((int)Y[0][0, 0], p)) % p;
            A[1, 0] = (A[0, 0] * Y[0][1, 0] * ModInverse((int)Y[0][0, 0], p)) % p;
            A[1, 1] = (A[0, 0] * Y[2][1, 0] * ModInverse((int)Y[0][0, 0], p)) % p;

            bool match = true;
            for (int i = 0; i < 4; i++)
            {
                KeyAX[i] = MatrixMultiplication(m, p, A, X[i]);
                Y_tmp = MatrixMultiplication(m, p, KeyAX[i], B);
                if (MatricesAreEqual(Y[i], Y_tmp) == false)
                {
                    match = false;
                    break;
                }
            }
            if (match == true)
            {
                double[,] newA = new double[2, 2];
                double[,] newB = new double[2, 2];
                newA = AssignMatrix(newA, A);
                newB = AssignMatrix(newB, B);
                ABPairs.Add(newA);
                ABPairs.Add(newB);
                ABCount += 1;
            }
            j = j + 1;
        }
    }


    public static void SolveEquations3x3(Random r, int p, double[,] A, double[,] B,
        out List<double[,]> XMatrices, out List<double[,]> YMatrices, out List<double[,]> ABPairs, out int
ABCount)
    {
        int m = A.GetLength(0);
        double[][,] X = new double[9][,];
        double[][,] KeyAX = new double[9][,];
        double[,] Y_tmp;
        double[,] KeyA = new double[m, m];
        double[,] KeyB = new double[m, m];
        XMatrices = new List<double[,]>();
        YMatrices = new List<double[,]>();
        ABPairs = new List<double[,]>();
        ABCount = 0;

        X[0] = new double[,] { { 1, 0, 0 }, { 0, 0, 0 }, { 0, 0, 0 } };
        X[1] = new double[,] { { 0, 1, 0 }, { 0, 0, 0 }, { 0, 0, 0 } };
        X[2] = new double[,] { { 0, 0, 1 }, { 0, 0, 0 }, { 0, 0, 0 } };
        X[3] = new double[,] { { 0, 0, 0 }, { 1, 0, 0 }, { 0, 0, 0 } };
        X[4] = new double[,] { { 0, 0, 0 }, { 0, 1, 0 }, { 0, 0, 0 } };
        X[5] = new double[,] { { 0, 0, 0 }, { 0, 0, 1 }, { 0, 0, 0 } };
        X[6] = new double[,] { { 0, 0, 0 }, { 0, 0, 0 }, { 1, 0, 0 } };
        X[7] = new double[,] { { 0, 0, 0 }, { 0, 0, 0 }, { 0, 1, 0 } };
        X[8] = new double[,] { { 0, 0, 0 }, { 0, 0, 0 }, { 0, 0, 1 } };


        double[][,] AX = new double[9][,];
        double[][,] Y = new double[9][,];
        for (int i = 0; i < 9; i++)
        {
            AX[i] = MatrixMultiplication(m, p, A, X[i]);
            Y[i] = MatrixMultiplication(m, p, AX[i], B);
            XMatrices.Add(X[i]);
            YMatrices.Add(Y[i]);
        }
        int j = 1;
        while (j < p)
        {
            A[0, 0] = j;
            B[0, 0] = (Y[0][0, 0] * ModInverse((int)A[0, 0], p)) % p;
            B[0, 1] = (Y[0][0, 1] * ModInverse((int)A[0, 0], p)) % p;
            B[0, 2] = (Y[0][0, 2] * ModInverse((int)A[0, 0], p)) % p;
```

```
            B[1, 0] = (Y[1][0, 0] * ModInverse((int)A[0, 0], p)) % p;
            B[1, 1] = (Y[1][0, 1] * ModInverse((int)A[0, 0], p)) % p;
            B[1, 2] = (Y[1][0, 2] * ModInverse((int)A[0, 0], p)) % p;
            B[2, 0] = (Y[2][0, 0] * ModInverse((int)A[0, 0], p)) % p;
            B[2, 1] = (Y[2][0, 1] * ModInverse((int)A[0, 0], p)) % p;
            B[2, 2] = (Y[2][0, 2] * ModInverse((int)A[0, 0], p)) % p;
            A[0, 1] = (A[0, 0] * Y[3][0, 0] * ModInverse((int)Y[0][0, 0], p)) % p;
            A[0, 2] = (A[0, 0] * Y[6][0, 0] * ModInverse((int)Y[0][0, 0], p)) % p;
            A[1, 0] = (A[0, 0] * Y[0][1, 0] * ModInverse((int)Y[0][0, 0], p)) % p;
            A[1, 1] = (A[0, 0] * Y[3][1, 0] * ModInverse((int)Y[0][0, 0], p)) % p;
            A[1, 2] = (A[0, 0] * Y[6][1, 0] * ModInverse((int)Y[0][0, 0], p)) % p;
            A[2, 0] = (A[0, 0] * Y[0][2, 0] * ModInverse((int)Y[0][0, 0], p)) % p;
            A[2, 1] = (A[0, 0] * Y[3][2, 0] * ModInverse((int)Y[0][0, 0], p)) % p;
            A[2, 2] = (A[0, 0] * Y[6][2, 0] * ModInverse((int)Y[0][0, 0], p)) % p;

            bool match = true;
            for (int i = 0; i < 9; i++)
            {
                KeyAX[i] = MatrixMultiplication(m, p, A, X[i]);
                Y_tmp = MatrixMultiplication(m, p, KeyAX[i], B);
                if (MatricesAreEqual(Y[i], Y_tmp) == false)
                {
                    match = false;
                    break;
                }
            }
            if (match == true)
            {
                double[,] newA = new double[3, 3];
                double[,] newB = new double[3, 3];
                newA = AssignMatrix(newA, A);
                newB = AssignMatrix(newB, B);
                ABPairs.Add(newA);
                ABPairs.Add(newB);
                ABCount += 1;
            }
            j = j + 1;
        }
    }


    private void button2_Click(object sender, EventArgs e)
    {
        // start measuring execution time
        Stopwatch ExecutionTimer = new Stopwatch();
        Stopwatch AlgTimer = new Stopwatch();
        ExecutionTimer.Start();

        // parse matrix order and integer
        int m = int.Parse(comboBox1.Text);
        int p = int.Parse(textBox1.Text);

        Random r = new Random();
        // generate non singular matrix A
        double[,] A = MatrixSatisfyRequirements(p, m, r);
        // display matrix A and det(A)
        richTextBox5.Text = DisplayMatrix(m, A);
        textBox10.Text = Mod(Determinant(A), p).ToString();

        // generate non singular matrix B
        double[,] B = MatrixSatisfyRequirements(p, m, r);
        // display matrix B and det(B)
        richTextBox4.Text = DisplayMatrix(m, B);
        textBox11.Text = Mod(Determinant(B), p).ToString();

        // generate modular inverse of matrix A
        double[,] ModInvA = ModularInverseMatrix(p, A);
        richTextBox12.Text = DisplayMatrix(m, ModInvA);

        // generate modular inverse of matrix B
        double[,] ModInvB = ModularInverseMatrix(p, B);
        richTextBox14.Text = DisplayMatrix(m, ModInvB);

        int iterations = int.Parse(textBox4.Text);

        // iterate brute force and find A, B pairs
        AlgTimer.Start();
```

```
            List<int> PairsNumbersList = new List<int>();
            if (m == 2)
            {
                for (int i = 0; i < iterations; i++)
                {
                    SolveEquations2x2(r, p, A, B, out XMat, out YMat, out ABPairs, out ABCount);
                    PairsNumbersList.Add(ABCount);
                }
            }
            if (m == 3)
            {
                for (int i = 0; i < iterations; i++)
                {
                    SolveEquations3x3(r, p, A, B, out XMat, out YMat, out ABPairs, out ABCount);
                    PairsNumbersList.Add(ABCount);
                }
            }
            AlgTimer.Stop();
            richTextBox3.Text = DisplayPairsNumbers(PairsNumbersList);

            // measure average algorithm time
            double AvgTime = AlgTimer.Elapsed.TotalSeconds / iterations;
            textBox6.Text = AvgTime.ToString();

            richTextBox7.Text = DisplayMatrixList(m, XMat, "X");
            richTextBox9.Text = richTextBox11.Text = DisplayMatrixList(m, YMat, "Y");

            string ABListString = null;
            richTextBox2.Text = DisplayMatrixPairsList(m, ABPairs, ABListString);

            // shape and display output X list of last iteration
            List<double[,]> OutputXList = ShapeOutputList(p, ModInvA, ModInvB, YMat);
            richTextBox13.Text = DisplayMatrixList(m, OutputXList, "X");

            // verify that output is correct
            textBox13.Text = Verification(DisplayMatrixList(m, XMat, "X"), DisplayMatrixList(m, OutputXList,
"X"));

            // finish measuring execution time
            ExecutionTimer.Stop();
            textBox12.Text = ExecutionTimer.Elapsed.TotalSeconds.ToString();
        }
```