ЎЗБЕКИСТОН РЕСПУБЛИКАСИ АЛОҚА, АХБОРОТЛАШТИРИШ ВА ТЕЛЕКОММУНИКАЦИЯ ТЕХНОЛОГИЯЛАРИ ДАВЛАТ ҚЎМИТАСИ
ТОШКЕНТ АХБОРОТ ТЕХНОЛОГИЯЛАРИ УНИВЕРСИТЕТИ

Қўл ёзма ҳуқуқида

УДК 004.056

ХУДОЙҚУЛОВ ЗАРИФ ТЎРАҚУЛОВИЧ

Симметрик криптотизим калитини генерациялаш алгоритмларининг тадқиқи

5А330301 – Криптография ва криптоанализ
Магистр академик даражасини олиш учун ёзилган
диссертация

Илмий раҳбар:
т.ф.н. Мусаев А.И.

Тошкент – 2013

МУНДАРИЖА

КИРИШ

Диссертация мавзусининг асосланиши ва унинг долзарблиги. Замон тараққий этган сари жамиятда ахборотга бўлган талаб ва эҳтиёж ҳам ортиб бормоқда, айниқса, ахборот технологияларининг кун сайин ривожланиб бориши ахборотлар ҳажмининг ҳам ортиб боришига хизмат қилиб келмоқда. Бу каби ахборотлар ичида маълум маънода ҳимояни, маҳфийликни ва сир сақланишини талаб этадиганлари ҳам бўлади, негаки, бу тоифадаги маълумотларнинг ошкор бўлиши, ўғирланиши ёки йўқ қилиниши каби ҳолатлар ташкилот учун катта талофотларни, молиявий ёки моддий зарарларни олиб келиши мумкин. Бу каби ҳолатларни олдини олиш учун эса қандай соҳа бўлмасин албатта, ахборот хавфсизлигини, унинг ҳимояси ва муҳофазасини амалга ошириши лозим. Замонавий компютер тизимлари ва тармоқларида ахборотни ҳимоя қилиш деганда узатилаётган, сақланаётган ва қайта ишланаётган ахборотнинг ишончлилиги ва бутунлигини таъминлаш мақсадида турли хил восита ва усулларни ишлатиш, чораларни кўриш ва тадбирларни ўтказиш тушунилади. Бу борада мамлакатимизда сўнгги йилларда давлатимиз раҳбарияти томонидан ахборот хавфсизлигига катта аҳамият берилмоқда. Бунга қабул қилинган бир нечта қонун ва меъёрий ҳужжатлар, жумладан, № 611-ИИ 29.04.2004 й. “Электрон ҳужжат айланиши тўғрисида”ги, № 562-ИИ 11.12.2003 й. “Электрон рақамли имзо тўғрисида”ги қонунлар, вазирлар махкамасининг №242 21.11.2007й. “Ахборотнинг криптографик ҳимоя воситаларини лойиҳалаштириш, тайёрлаш, ишлаб чиқариш, реализатсия қилиш, таъмирлаш ва улардан фойдаланиш фаолиятини литсензиялаш тўғрисидаги низомни тасдиқлаш ҳақида”ги қарори, Президентимизнинг 2007 йил 3 апрелда қабул қилган “Ўзбекистон Республикасида ахборотнинг криптографик ҳимоясини ташкил этиш чора-тадбирлари” тўғрисидаги қарори мисол бўлиши мумкин.

Ҳозирги кунга қадар ахборот хавфсизлигини таъминлашда энг ишончли воситалардан бири ахборотни криптографик ҳимоя қилиш воситалари ҳисобланади. Республикамизда бу йўналиш жадал суръатлар билан ривожланмоқда. Янгидан янги криптографик тизимлар, алгоритмлар, стандартлар ишлаб чиқилмоқда ва турли соҳаларга тадбиқ этилмоқда. Жумладан, электрон рақамли имзо тўғрисида ўзимизнинг миллий стандарт O'z DSt 1092:2009, хэшлаш сохасида O'z DSt 1106:2009 миллий стандартининг ишлаб чиқилганлиги ва бу стандартнинг хар томонлама бошқа чет давлат стандартларидан хеч ҳам қолишмаслиги бунга мисол бўла олади.

Ахборот-коммуникация тизимларида локал ва корпоратив компьютер тармоқларининг кўпайиши ва бу тармоқларнинг глобал интернет тармоғида уланиши фойдаланувчилар ўртасида электрон маълумот алмашувининг кенг қўлланилишига олиб келди. Жамиятнинг ахборотга бўлган эҳтиёжи ахборот-ресурс марказларининг ташкил этилиши ва очиқ интернет тармоғи орқали фаолият юритишига олиб келиши билан бир вақтда алмашинувчи маълумотларнинг узатилиши давомида кафолатли мухофазасини таъминлаш масаласини келтириб чиқарди.

Ҳозирги замон криптографиясида бардошли калитларни ишлаб чиқиш долзарб муаммолардан биридир. Чунки хар бир криптотизимда калит параметри муҳим аҳамиятга ега. Криптотизим бардошлилиги калитнинг қанчалик тасодифийлигига боғлик бўлиб, тасодифий сонлар кетма-кетлигини ҳосил қилиш анча мураккаблиги, бунинг ўрнига эса псевдотасодифий кетма-кетликлардан фойдаланиш, псевдотасодифий кетма-кетликларни ишлаб чиқиш муҳим вазифа эканлигини англатади.

Илмий тадқиқот ишининг объекти ва предметининг белгиланиши. Тадқиқотнинг объекти криптографик аппарат ва аппарат-

дастурий воситаларга қўлланилувчи криптобардошли калит генерацияси алгоритмлари.

Калит генерацияси алгоритмлари ва таркибидаги акслантиришларнинг криптографик бардошлилиги, самарадорлик даражаларини баҳолаш усуллари тадқиқотнинг предмети ҳисобланади.

Илмий тадқиқот ишининг мақсади ва вазифалари. Ушбу магистрлик диссертациясида олиб борилган илмий тадқиқот ишининг мақсади мавжуд псевдотасодифий сонлар генераторлари ўрганиб чиқиб, шулар асосида бардошли калит генераторини ишлаб чиқиш.

Тадқиқот мақсадини амалга ошириш учун диссертация ишини бажаришда қуйидаги вазифалар қўйилди:

- мавжуд калит генерацияси алгоритмларини ва яратиш йўналишларини туркумлаш;
- сонларни тасодифийликка текширишнинг эмперик ва статистик усулларини ишлаб чиқиш ва шулар асосида мавжуд калит генераторларини таҳлил қилиш;
- аппарат ва аппарат-дастурий криптографик воситаларида самарали қўлланувчи криптобардошли акслантиришлардан фойдаланиб калит генерацияси алгоритмларини яратиш ва жараён босқичларининг функционал схемасини тузиш;
- яратилган алгоритмларнинг криптобардошлиги ва самарадорлиги кўрсаткичлари хақида аниқ натижаларга эришиш.

Илмий тадқиқотнинг асосий масалалари ва фаразлари. Ахборот-коммуникация тизимларидаги маълумотларни муҳофазасини таъминловчи аппарат-дастурий криптографик воситалар криптобардошлигини оширишга сабаб бўлувчи псевдотасодифий сонлар генераторини таҳлил этиш, алгоритмларини яратиш, тасодифийликка текширувчи тестлар алгоритмларини яратиш масалалари ечимларига эришиш.

Мавзу бўйича қисқача адабиётлар таҳлили. Компьютер тармоқларини, электрон маълумот алмашиниш тизимларини ривожланиши, таълим, ишлаб чиқариш, савдо-сотиқ ва банк соҳасида кенг қўлланилиши ахборотнинг криптографик ҳимоясини жадаллик билан ривожланишига ва оммавий тарзда фойдаланишга сабаб бўлмоқда.

Криптография, тасодифий ва псевдотасодифий калит генераторлари соҳасидаги манбалар сифатида Шнайер Б., Рон Р., Роберт Ж., Шеннон К.Е. томонидан олиб борилган тадқиқотларни келтириш мумкин. Ушбу олиб борилган тадқиқотлар шуни кўрсатдики мавжуд алгоритмлар криптобардошлиги уларнинг алгоритмларини махфий тутилишига боғлиқ бўлмай, фақатгина калитнинг сир тутилишиги боғлиқ бўлиши керак. Шуларни ҳисобга олиб криптографик тизимларда бардошли калитларни яратиш мақсадида калит генераторлари алгоритмлари яратилди.

Бу соҳада Ўзбекистон Республикаси олимлари томонидан ҳам етарли тадқиқот ишлари олиб борилмоқда, бунга Хасанов П.Ф., Арипов М.М., Каримов М.М., Акбаров Д.Е., Ғаниев С.К., Хасанов Х.П., Ахмедова О.П. томонидан эришилган натижаларни келтириш мумкин.

Криптографик тизимларга қўйилган талабларга асосан, ҳозирги замон криптографиясида бардошлиликни таъминлаш калитларнинг тасодифийлик даражасига боғлиқлигини ҳисобга олиб псевдотасодифий кетма-кетликлар ишлаб чиқарирувчи генераторларни хусусиятлари ва самарадорлигини чуқур таҳлил қилиниши ва етарли даражада ўрганилиши керак.

Илмий тадқиқот ишида қўлланилган услубларнинг қисқача тавсифи. Ушбу илмий тадқиқот ишида ахборотни криптографик ҳимоялаш тизимлари назарияси, эҳтимоллар назарияси, сонлар назарияси, математик мантиқ ва комбинаторика методларидан фойдаланилган. Булардан ташқари солиштириш, тестлаш ва қиёсий таҳлил усулларидан фойдаланилди.

Тадқиқот натижаларининг назарий ва амалий аҳамияти. Мавжуд ва яратилган псевдотасодифий сонлар генератори алгоритмларининг криптобардошлигини баҳолаш ва тизимли тадқиқлаш магистрлик диссертациясининг назарий аҳамияти ҳисобланади.

Магистерлик диссертация ишининг амалий аҳамияти – яратилган псевдотасодифий сонлар генератори алгоритмлари ва дастурий таъминотларини Ўзбекистон шароитида ишлаб чиқиладиган ишончли ва тезкор ишловчи аппарат-дастурий криптографик ҳимоя воситаларида фойдаланиш мумкин.

Илмий тадқиқот ишининг илмий янгилиги. Ушбу тадқиқот ишининг янгилиги мавжуд псевдотасодифий кетма-кетликлар кўриб чиқиб:

- улар асосида янги псевдотасодифий кетма-кетлик алгоритми яратилди ва унинг дастурий таъминоти ишлаб чиқилди;
- псевдотасодифий сонларни тасодифийликка текширишнинг комплекс мажмуи ишлаб чиқилди.

Диссертация таркибининг қисқача тавсифи. Ушбу тадқиқот иши қуйидаги: кириш, 3 та бўлим, хулоса, фойдаланилган адабиётлар рўйхати ва иловадан иборат. Ушбу тадқиқот ишида 9 та жадвал, 14 та расмдан иборат. Илмий тадқиқот ишининг умумий ҳажми 81 саҳифани ташкил этади.

I боб. Криптографик шифрлаш алгоритмларида калит генерацияси масаласи

1.1.Тасодифий ва псевдотасодифий сонларни генерациялаш принциплари ва усуллари

Бугунги жамият тараққиёти инсоният тафаккурининг маҳсули бўлган ривожланган илм-фан ютуқларига асосланган техника ва технологиялар билан бир қаторда, кенг маънода, ахборотларнинг муҳим аҳамиятга эгалиги орқали ҳам белгиланади. Инсон тафаккури ривожининг манбаи эса маълумотлар (ахборотлар) мажмуидан иборатдир. Шак-шубҳасиз ўз вақтида олинган тўла ва ишончли маълумот, шу маълумот билан боғлиқ бўлган ҳолатдан келиб чиқадиган амалий фаолиятларнинг мақсадли кечишларини мувофиқлаштиришда муҳим аҳамият касб этади. Фаолият мақсадларининг турлича бўлиши табиий равишда ахборотлардан турли мақсадларда фойдаланиш асосларига сабаб бўлади. Шунинг учун бугунги, ахборотларни сақлаш ва узатиш тизимлари бир томондан такомиллашиб мураккаблашган ва иккинчи томондан ахборотлардан фойдаланувчилар учун кенг қулайликлар вужудга келган даврда, ахборотларни мақсадли бошқаришнинг қатор муҳим масалалари келиб чиқади. Бундай масалалар қаторига катта ҳажмдаги ахборотларнинг тез ва сифатли узатиш ҳамда қабул қилиш, ахборотларни ишончлилигини таъминлаш, ахборотлар тизимида ахборотларни бегона шахслардан (кенг маънода) муҳофаза қилиш каби кўплаб бошқа масалалар киради. Ахборот ва ахборот тизимидан фойдаланиш инсоният фаолиятининг барча соҳаларига кириб бориб, муҳим аҳамият касб этиб, ривожланиб бораётган бугунги жамиятда ахборотларни мақсадли бошқариш фаоллашмоқда. Компьютерлар ва компьютер тизимлари ахборот тизимининг муҳим бўғимидир. Интернет тармоқлари жамият фаолиятининг барча соҳаларини қамраб олиб, ахборотни тез ва сифатли алмашинувини таъминлаш технологияларининг

ривожланишига ижобий манба бўлиб келмокда. Юқоридаги келтирилган асосли мулоҳазалардан келиб чиқиб, ахборотларни асли ҳолидан ўзгартирилган ҳолда, яъни шифрланган ҳолда, сақлаш ва узатиш масалаларининг муҳим эканлигига шубха йўқдир. Ахборотларнинг муҳофазасини таъминлаш масалалари инсоният жамиятида қадимдан мухим бўлиб келган. Айтиш мумкинки, ахборотни муҳофаза қилиш услублари жамиятда дастлабки пайдо бўлган муомила тили ва ёзуви билан узвий боғлиқ ҳамда тенгдошдир. Ҳақиқатдан ҳам, қадимда ёзув муомила воситасидан фақат айрим юқори табақадаги жамият аъзоларигина фойдаланганлар. Қадимий Миср ва Ҳиндистоннинг илоҳий китоблари бунга мисол бўла олади. Эрамиздан аввалги бешинчи асрда яшаб ўтган грек олими Геродотнинг ҳабар беришича, қадимий Мисрда шифрланган ахборотлар ролини, жрецлар, яъни юқори табақадаги етук фикрли кишилар томонидан яратилган муомила тили бажарган. Бунда учта алфбо асосланилган: ёзув, илоҳий ва маҳфий. Ёзув алфбоси оддий ўзаро муомилада қўлланилган, илоҳий алфавит диний муомила воситаси сифатида қўлланилган, маҳфий алфбо эса маълумотларнинг асл маъносини бегоналардан муҳофаза қилишда ва астриологлар томонидан қўлланилган [5].

Турли ёзув алфболарининг вужудга келиши ва ривожланиши натижасида *криптография* мустақил йўналишда ривожлана борди.

Криптография - ахборотни аслидан ўзгартирилган ҳолатга акслантириш услубларини топиш ва такомиллаштириш билан шуғулланади. Дастлабки системалашган криптографик услублар эрамиз бошида, Юлий Цезарнинг иш юритиш ёзишмаларида учрайди. У, бирор маълумотни маҳфий ҳолда бирор кишига етказмоқчи бўлса, алфбонинг биринчи ҳарфини тўртинчи ҳарфи билан, иккинчи ҳарфини бешинчиси билан ва ҳоказо тартибда алмаштириб матнни асл ҳолатидан шифрланган матн ҳолатига ўтказган.

Криптографик системалар йўналишидаги изланишлар айниқса биринчи ва иккинчи жаҳон уруши йиллари даврида муҳим аҳамият касб этди ва жадал ривожланди. Урушдан кейинги йилларда, ҳисоблаш техникаларининг яратилиши, уларнинг такомиллашиб, инсоният фаолиятининг барча соҳаларига чуқур ва кенг маънода кириб бориши, криптографик услубларни табиий равишда ривожланиб ва такомиллашиб боришини таъқозо этмоқда [5,6].

Криптографик услубларнинг ахборот тизими муҳофазаси масалаларида қўлланилиши, айниқса, ҳозирги кунда фаоллашиб бормоқда. Ҳақиқатан ҳам, бир томондан компьтер тизимларида интернет тармоқларидан фойдаланган ҳолда катта ҳажмдаги давлат ва ҳарбий аҳамиятга эга бўлган, ҳамда, иқтисодий, шаҳсий ва бошқа турдаги ахборотни тез ва сифатли узатиш, қабул қилиш кенгайиб бормоқда. Иккинчи томондан эса бундай ахборотларнинг муҳофаза қилиниши таъминлаш масалалари муҳимлашиб бормоқда.

Ахборотни мухофаза қилиш масалалари билан *криптология* (kryptos-маҳфий, logos- илм) фани шуғулланади. Криптология мақсадлари ўзаро қарама-қарши иккита йўналишига эга бўлган – *криптография* ва *криптоанализ* [5].

Криптографиянинг очиқ маълумотларни шифрлаш масалаларининг математик услублари билан шуғулланиши тўғрисида юқорида айтиб ўтилди.

Криптоанализ эса шифрлаш услуби (калити ёки алгоритми)ни билмаган ҳолда шифрланган маълумотни асл ҳолатини (мос келувчи очиқ маълумотни) топиш масалаларини ечиш билан шуғулланади.

Ҳозирги замон криптографияси қуйидаги тўртта бўлимни ўз ичига олади:

1. Симметрик криптотизимлар.

2. Очиқ услубга ёки яна бошқача айтганда очиқ калитлар алгоритмига асосланган криптотизимлар.
3. Электрон рақамли имзо криптографик тизимлари.
4. Криптотизимлар учун криптобардошли калитларни ишлабчиқиш ва улардан фойдаланишни бошқариш.

Симметрик блокли шифрлаш алгоритмлари каби, узуликсиз (оқимли) шифрлаш алгоритмларини яратилиши ҳам табиий зарурият асосида вужудга келган. Нисбатан кичик узунликка эга бўлган, яъни кафолатланган криптобардошлиликни таъминловчи узунликга эга бўлган - бугунги кунда 128 битдан кам бўлмаган калит билан бир томонли криптографик акслантиришлар асосида, етарли даражада катта узунликдаги псевдотасодифий сонлар кетма-кетлик (ПТСКК) гаммаси ишлаб чиқарувчи генераторлар негизида узуликсиз шифрлаш алгоритмлари яратилади. Узунлиги 128 битдан кам бўлмаган калитларнинг мумкин бўлган барча вариантлари сони 2^{128} тадан кам бўлмай, уларни ҳаммасини танлаб чиқиш жараёнини амалга ошириш, бугунги кун ҳисоблаш техника ва технологияларининг мавжуд илғор имкониятларидан фойдаланилганда ҳар доим ҳам самарали натижалар беравермайди. Ана шундай генераторлар ишлаб чиқарган гамма кетма-кетликни ташкил этувчи алфавит белгиларини очиқ маълумот мос алфавити белгилари билан бирор амал бажариш орқали шифр маълумот алфавити белгиларига алмаштириш – гаммалаштириш амалга оширилади. Бундай шифрлаш жараёни кўп алфавитли ўрнига қўйиш шифрлашни амалга оширишни самарали усулуни ифодалайди – кафолатли криптобардошлиликни таъминловчи кичик узунликдаги калит билан, очиқ маълумотнинг частотавий хусусиятларини шифрмаълумотга кўчирмайдиган етарли криптобардошлиликни таъминловчи шифрлашни амалга оширади [9].

Узлуксиз шифрлаш алгоритмлари асосини ПТСКК ишлаб чиқарувчи генераторлар ташкил этади. Бундай генераторларнинг асосий

криптобардошлилик характеристикаси ушбу генераторлар ҳосил қилган кетма-кетликнинг тасодифийлигидадир. Ҳосил қилинган кетма-кетликлар блокларининг тасодифийлик даражаси маълум бир критерийлар орқали баҳоланади. Тасодифийлик даражаси юқори бўлган псевдотасодифий кетма-кетликни ишлаб чиқарувчи генераторлар замонавий криптотизимларнинг ажралмас қисми ҳисобланади. Тасодифий кетма-кетликлар криптографияда қуйидаги мақсадларда қўланилади [5,9,20]:

-симметрик криптотизимлар учун тасодифийлик даражаси юқори бўлган сеанс калитлари ва бошқа калитларни генерация қилишда;

-асимметрик криптотизимларда қўлланиладиган катта қийматлар қабул қилувчи параметрларнинг тасодифий бошланғич қийматлари генерациясида;

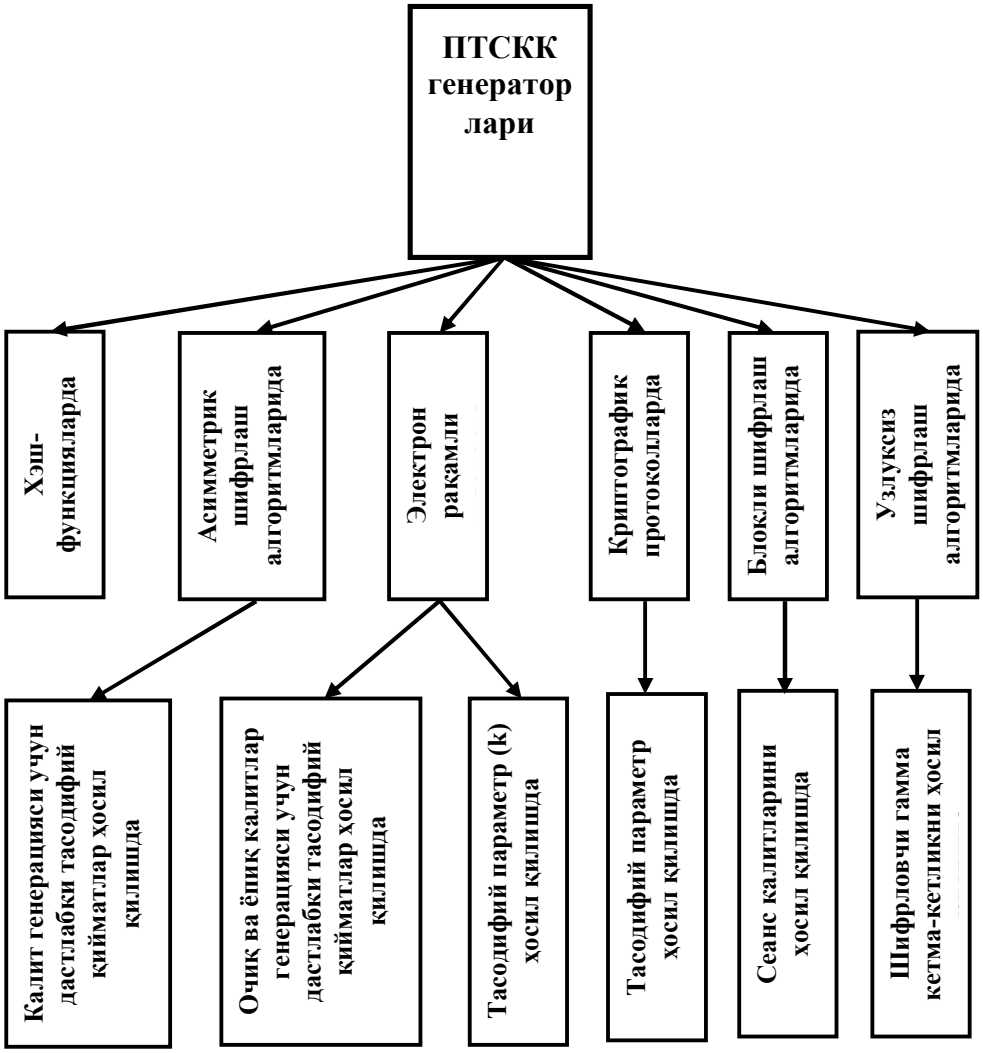
-блокли шифрлаш алгоритмларининг бошланғич тасодифий қиймат талаб қилувчи CBC, OFB ва бошқа қўлланиш тартиб-қоидалари учун тасодифийлик даражаси юқори бўлган бошланғич векторлар ҳосил қилишда;

-электрон рақамли имзо тизимларида катта қийматга эга параметрлар учун дастлабки тасодифий қийматларни генерациясида;

-битта протокол орқали бир хил маълумотларни ҳар-хил калитлар қўллаш билан шифрлаб ҳар-хил кўринишда узатиш учун талаб қилинадиган ҳолатларда калит учун етарли узунликдаги тасодифий кетма-кетлик ҳосил қилишда, масалан SSL ва SET протоколларида.

Ташкил этувчилари тенг эҳтимоллик билан тақсимланган тасодифий кетма-кетлик ҳосил қилиш муаммосини ечиш кетма-кетликни ташкил этувчиларининг текис тақсимланган генерацияси муаммосини ечиш билан боғлиқ. Бирор кетма-кетликни ташкил этувчи элементлар, шу кетма-кетликда деярли тенг миқдорда қатнашган бўлса, бу кетма-кетлик текис тақсимотга эга дейилади: агар A -кетма-кетликни ташкил этувчи $x_t \in A$ -элементлари сони N та бўлса, у ҳолда иҳтиёрий $t \in N$ учун, A -кетма-

кетликни ташкил этувчи $x_t \in A$ -элементнинг, шу кетма-кетликдаги частотаси, бошқа элементларнинг частотаси билан деярли бир хил бўлади, яъни ҳар бир $x_t \in A$ –элемент шу кетмакетликда деярли бир хил эҳтимоллик билан қатнашади [5].



1-расм. Псевдотасодифий сонлар кетма-кетлиги генераторларини қўллаш соҳалари

Тасодифий кетма-кетликлар хақиқий тасодифий кетма-кетликларга ва псевдотасодифий кетма-кетликларга бўлинади.

Тасодифий кетма-кетликни: физик генераторлар ва дастурий генераторлардан фойдаланиб ҳосил қилиш мумкин.

Физик ҳодисаларнинг ўзгариш мажмуига асосланган генераторлар орқали ишлаб чиқилган кетма-кетлик ҳақиқий тасодифий бўлиб, бу кетма-кетликни бир мартагина ишлаб чиқилиб, уни кейинчалик бирор бир усул ёки восита билан худди шундай тарзда такрорланишини бошқариш мураккаб ҳисобланади. Шу сабабли маълумотларни шифрлаш жараёнида бевосита физик генераторлар билан ишлаб чиқилган кетма-кетликни калитлар гаммаси сифатида қўллаш мақсадга мувофиқ эмас. Чунки, дешифрлаш жараёнида қўлланиладиган физик генераторнинг айнан шифрлаш жараёнида қўлланилган кетма-кетликни ишлаб чиқиши кафолатланмайди.

Бирор номаълум параметрга (калитга) боғлиқ бўлган математик модел асосида псевдотасодифий кетма-кетлик ишлаб чиқувчи дастурий генераторлар ҳосил қилган псевдотасодифий кетма-кетликни, номалум параметр қийматини билган ҳолда, худди шу математик модел ва унинг дастурий таъминоти асосида кетма-кетликнинг қайта такрорланишини бошқариш мумкин. Бундай ҳолат, маълумотларни шифрлаш жараёнида бевосита дастурий генераторлар билан ишлаб чиқилган псевдотасодифий кетма-кетликни калитлар гаммаси сифатида қўллаш мақсадга мувофиқлигини англатади ва дешифрлаш жараёнида қўлланиладиган дастурий генераторнинг айнан шифрлаш жараёнида қўлланилган псевдотасодифий кетма-кетликни ишлаб чиқиши кафолатланади.

Юқорида кўрсатиб ўтилган амалий масалаларни ечишда хақиқий тасодифий кетма-кетликлар ишлаб чиқувчи тасодифий физик ходисаларга асосланган генераторлар олдиндан калитлар блоклари мажмуини яратишда, генераторларнинг бошланғич параметрлари қийматларини ўрнатишда ва бошқа шу каби масалаларни ечишда самарали натижалар беради [19].

Етарли катта давр узунлигига эга ва тасодифийлик даражаси юқори бўлган кетма-кетликлар ҳосил қилувчи дастурий ПТСКК генераторини

амалда қўланишлари самарали ва қулай бўлиб, криптографик воситаларда кенг қўлланилади.

Мавжуд дастурий генераторлар ва улар асосидаги узлуксиз шифрлаш тизимлари маълум бир ёндашувлар асосида яратилган.

Алгоритмларни криптобардошлилигини етарли даражада таминланганлигини кафолатлаш ёки исботлаш асослари нуқтаи - назаридан мавжуд ПТСКК генераторлари асосан учта йўналишга ажратиш мумкин [5,9]:

1. Тизимли-назарий ёндашув асосида қурилган ПТСКК генераторлар;
2. Мураккабликка асосланган ПТСКК генераторлар;
3. Комбинациялаш ёндошуви асосида қурилган ПТСКК генераторлар.

Тизимли-назарий ёндашув асосида яратилган ПТСКК генераторларини яратилиш асосларига кўра:

- элементар реккурент ҳисоблашларга;
- силжитиш регистрларига;
- бир томонли функцияларга;
- байтлар ва битлар блокларининг ўрнини боғлиқсиз алмаштиришга асосланган генераторларга ажратиш мумкин.

Мураккабликка асосланган ПТСКК генераторларини яратилиш асосига кўра:

- катта сонларни туб кўпайтувчиларга ажратиш мураккаблигига асосланган;
- квадратик чегирма усулига асосланган;
- дискрет логарифмлаш масаласининг мураккаблигига асосланган генераторларга ажратиш мумкин.

Комбинациялаш ёндошуви асосида қурилган ПТСКК генераторларини яратилиш асосларига кўра:

- полиномиал комбинациялашга асосланган;
- тасодифий параметрли комбинациялашга асосланган;

- макларен-марсали усулига асосланган генераторларга ажратиш мумкин.

1.2. Чизиқли ва чизиқсиз конгруэнт генераторлар

Тизимли-назарий ёндашув асосида яратилган узлуксиз шифрлаш алгоритмларининг бардошлилиги, бу алгоритмларда қўлланилган акслантиришларнинг назарий ва амалий бир томонлилик хусусиятларининг қай даражада ишончлилигини баҳолаш билан исботланади.

Элементар реккурент ҳисоблашларга асосланган псевдотасодифий кетма-кетлик генераторлари уларда қўлланилган акслантиришларга кўра *чизиқли, мультипликатив, чизиқсиз* туркумларга бўлинади [5].

Чизиқли ва мультипликатив конгруэнт генераторлар. Чизиқли конгруэнт генераторлар умумий ҳолатда $x_{i+1}=(ax_i+c)\bmod N$ формула билан аниқланувчи реккурент ҳисоблашга асосланган. Дастлабки берилган кириш параметрлари асосида кетма-кетликлар ҳосил қилинади.

Кириш параметрлари:

N – чекли майдон характеристикасини ифодаловчи сон, a ва c - ўзгармас мусбат бутун сонлар, x_0 – бошланғич бутун қийматли сон;

Кетма-кетликни ташкил этувчи чиқиш қийматлари :

$$x_{i+1}=(ax_i+c)\bmod N,\quad i=0,1,2,3,\ldots;$$

Чизиқли конгруэнт генераторнинг кириш параметри c=0 бўлса, яъни

$$x_{i+1}=(ax_i)\bmod N,\quad i=0,1,2,3,\ldots;$$

бўлса, бу генератор чизиқли мультипликатив генератор дейилади.

Тасдиқ 1. Ушбу $x_{i+1}=(ax_i+c)\bmod N,\quad i=0,1,2,3,\ldots;$ рекурент формула билан аниқланган псевдотасодифий кетма-кетлик максимал N даврга эга бўлиши учун қуйидаги:

1) c ва N -ўзаро туб сонлар, яъни ЭКУБ(c,N)=1;

2) p сони N сонининг бўлувчиси ва a-1 сони p сонига каррали;

3) N сони 4 каррали бўлса, a-1 сони ҳам 4 га каррали;

шартларнинг бажарилиши зарур ва етарли [5].

Қуйида эса чизиқли ва мультипликатив генераторларнинг максимал даврга эга бўлиши билан боғлиқ айрим хоссалар келтириб ўтилади [5].

Бевосита ҳисоблаш билан ишонч ҳосил қилиш мумкинки, ушбу $x_{i+1}=(ax_i+c)\bmod N$, $i = 0,1,2,3, \ldots$; тенглик билан аниқланувчи кетма-кетлик умумий ҳади учун :

$$x_t = \left(a^t x_0 + \frac{a^t - 1}{a - 1} c \right) \bmod N, \quad t \geq 1$$

формула ўринли.

Чизиқли ва мультипликатив конгруэнт генераторларнинг камчилиги шундаки, ПТКК бирор биграммасини $(z_1; z_2)$: $z_1 = x_t$, $z_2 = x_{t+1}$, билган ҳолда, унинг бошқа ташкил қилувчиларини топиш имконияти мавжуд . Ҳақиқатан ҳам, кетма-кетликни барча ташкил қилувчи қийматлари $z_2 = a z_1 + c - k N$, k=0,1,… чизиқлар оиласида ётади.

Чизиқли конгруент генератор ҳисобланган Фишман ва Море генераторида кирувчи параметр z_0 қуйидагича олинган [21]:

$$z_0=23482349.$$

ПТСКК эса қуйидаги амал орқали ҳисобланади: $z_{i+1}=a*z_i mod(2^{31}-1)$. Бу ерда a ҳолат функцияси.

Чизиқсиз конгруэнт генераторлар.

Кириш параметрлари:

N – чекли майдон характеристикасини ифодаловчи сон;

d, a ва c - ўзгармас мусбат бутун сонлар, x_0 – бошланғич қиймат;

Кетма-кетликни ташкил этувчи чиқиш қийматлари :

$x_{i+1}=(dx_i^2+ax_i+c)\bmod N$, бу ерда i=0,1,2,… .

Бу генератор квадратик генератор деб ҳам аталади.

Тасдиқ 2. Квадратик генераторлар ҳосил қилган ПТКК ўзининг $T_{\max} = N$ максимал даврига эга бўлиши учун қуйидаги шартларнинг:

1. c ва N –ўзаро туб сонлар;

2. d, a-1 -сонлари бирор p –туб сонга каррали бўлиб, бу p – сони N нинг бўлувчиси;

3. d – жуфт сон бўлиб,

$$d = \begin{cases} (a-1) \bmod 4, \text{ агар } N \text{ сони } 4 \text{ га каррали бўлса}; \\ (a-1) \bmod 2, \text{ агар } N \text{ сони } 2 \text{ га каррали бўлса}; \end{cases}$$

2. агарда N сони 9 га каррали бўлса, у ҳолда $d \bmod 9 = 0$, ёки $d \bmod 9 = 1$ ва $cd \bmod 9 = 6$;

бажарилиши зарур ва етарли.

Шунингдек $N=2^q$ бўлса, максимал даврни таъминлаш учун d-тоқ бўлиши ва $a=(d+1) \bmod 4$ бўлиши етарлидир.

Квадратик конгруент генератор $x_{i+1}=x_i^2 mod\ p (i \geq 0)$ учун 512 бит узунликка эга бўлган p ва x_0 параметрлар қуйидагича олиниши тавсия этилади [21]:

p = 987b6a6bf2c56a97291c445409920032499f9ee7ad128301b5d0
254aa1a9633fdbd378d40149f1e23a13849f3d45992f5c4c6b7104
099bc301f6005f9d8115e1;

x_0 = 3844506a9456c564b8b8538e0cc15aff46c95e69600f084f0657c24
01b3c244734b62ea9bb95be4923b9b7e84eeaf1a224894ef0328d44bc3e
b3e983644da3f5.

Кириш бит узунлиги 512 бит бўлган кубик конгруент генератор($x_{i+1}=x_1^3 mod 2^{512}$) учун еса кириш битини

x_0 = 7844506a9456c564b8b8538e0cc15aff46c95e69600f084f0657c2401b3c2
44734b62ea9bb95be4923b9b7e84eeaf1a224894ef0328d44bc3eb3e983644
da3f5 кўринишда танлаш мақсадга муофиқдир [21].

Чизиқли ва мультипликатив конгруэнт генераторлар каби чизиқсиз генераторлар ҳам киптотаҳлил усулига бардошсиз.

Хозиргача чизиқсиз силжитиш регистрларига асосланган генераторлар ҳосил қилган кетма-кетликларни етарлича таҳлил қилувчи математик усуллар ишлаб чиқилмаган. Шу сабабли чизиқсиз тескари боғланишли регистрлар орқали амалга оширилган генераторлар билан боғлиқ бўлган қуйидаги муаммоларни келтириш мумкин [5]:

- ҳосил қилинган псевдотасодифий кетма-кетликда текис тақсимотдан четланиш бўлиши мумкин, яъни “0” ва “1” белгиларнинг ишлаб чиқилган гамма кетма-кетлик блокларидаги миқдори деярли тенг бўлмаслиги мумкин;
- кетма-кетликнинг даври кутилганидан қисқа бўлиши мумкин;
- кетма-кетлик даври ҳар-хил бошланғич қийматлар учун ҳар-хил бўлиши мумкин, яъни маълум бир талабга жавоб берувчи параметрлар танланганда ҳар қандай иҳтиёрий бошланғич қиймат учун генератор ҳосил қилган кетма-кетлик даври максимал бўлади деб бўлмайди;
- ҳосил қилинган гамма кетма-кетлик текшириш ҳисоб-китобларисиз тасодифийга ўхшаб кўриниши мумкин, лекин регистрнинг маълум бир ҳолатидан сўнг, чизиқсизлик амалининг маҳсули сифатида, кейинги ҳосил бўлган гамма кетма-кетлик элементлари фақат “0” ёки “1” лардан иборат бўлиб қолиши мумкин.

Чизиқсиз силжитиш регистрларининг криптографик самарали тарафи бундай регистрларга асосланган узлуксиз шифрларнинг криптографик таҳлили усуллари камлигидир.

1.3. Байтлар ва битлар ўрнини боғлиқсиз алмаштиришга асосланган генераторлар

ПТСКК генераторлари яратишда байтлар ва битлар ўрнини алмаштиришга асосланган усуллар муҳим аҳамият касб этади. Чунки ушбу

усул асосида яратилган генераторлар бошқа турдаги ПТСКК генераторларга қараганда криптобардошли саналади. Ушбу бўлимда ушбу турдаги машҳур бўлган RC4 ва ISAAC генераторлари билан танишамиз.

RC4. RC4 – узлуксиз шифрлаш алгоритми бўлиб, у SSL(Secure Sockets Layer) пратаколи ва WEP (симсиз тармоқларда хавфсизликни таъминлашда) кенг фойдаланилади. RC4 узлуксиз шифрлаш алгоритми Ron Rivest томонидан 1987 йилда яратилган ва шунинг учун RC4(Rivest Cipher 4) деб номланган [6,22].

RC4 псевдотасодифий битлар кетма-кетлигини ҳосил қилади ва ҳосил қилишда икки қисмдан иборат бўлган махфий оралиқ ҳолатидан фойдаланилади:

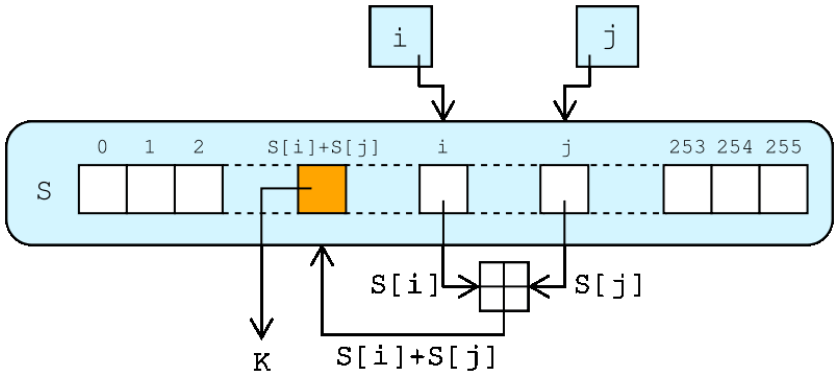
- барча мумкин бўлган 256 байтнинг жойлашишдаги ўрни(S ни топиш);
- иккита 8 – битли индекслар(*i* ва *j* ларни топиш).

Байтларнинг келиш тартиби калит узунлиги билан амалга оширилади, одатда 40-256 бит оралиғида бўлиб, калит жадвали(key-scheduling) алгоритми орқали ҳосил қилиниди. Бу жараён тугагандан сўнг псевдотасодифий сонлар генератори алгоритми ёрдамида битлар кетма-кетлиги ҳосил қилинади [22].

Калит жадвали алгоритми қуйидагича:

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

Псевдотасодифий сонлар генератори алгоритми орқали ҳосил бўлган кетма-кетлик танланган S(i) ва S(j) ўзгарувчиларни mod256 бўйича қўшишдан ҳосил бўлади (2- расм).



2-расм. RC4 генератори алмаштириши

Псевдотасодифий сонлар генератори алгоритми қуйидагича:

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    k := inputByte XOR S[(S[i] + S[j]) mod 256]
    output K
endwhile
```

Алгоритмда i ўзгарувчини қиймати ортиши билан ҳосил бўлган байтлар сони ҳам ортиб боради.

Бу ерда алмаштириш функцияси *swap* қуйидаги кўринишга ега:

```
byte temp = array[ind1];
array[ind1] = array[ind2];
array[ind2] = temp;
```

Ушбу генератор криптобардошли саналиб, ушбу хусусият кирувчи калит тасодифийлик даражаси билан белгиланади. Ҳозирда ушбу алгоритмнинг бир нечта вариантлари мавжуд бўлиб(RC4A, VMPC, RC4+), уларда дастлабкиларида мавжуд камчиликлар бартараф этилган.

ISAAC. Ушбу ПТСКК генератори 1966 йилда Роберт Женкинс томонидан яратилган бўлиб, RC4 алгоритмига ўхшашдир. Кирувчи параметр сифатида 32 бит ўлчамдаги сўзлардан иборат бўлган 256 узунликдаги массивдир. Чиқишнинг ҳар бир босқичида худди шу ўлчамдаги массив ҳосил бўлади. Ушбу ПТСКК генераторида ^(XOR), +(mod2^k) ва чапга ва ўнга суриш амаллари(<<, >>)дан ташкил топган [16,23].

f(a,i) функция эса қуйидагича ҳисобланади:

$$f(a,i)= \begin{cases} a<<13 \text{ if } i=0 \bmod 4 \\ a>>6 \text{ if } i=1 \bmod 4 \\ a<<2 \text{ if } i=2 \bmod 4 \\ a>>16 \text{ if } i=3 \bmod 4 \end{cases}$$

бу ерда i {0,…..,255} ораликқа тегишли сон.

Ушбу генераторнинг алгоритми қуйидагича:

Кирувчи параметрлар: a, b, c ва s ҳолат массиви, 256 ўлчамга ега бўлган 32 битли сўзлардан ташкил топган.

Чиқиш r массив, 256 ўлчамли 32 битли сўздан иборат бўлади.

1: $c \leftarrow c + 1$

2: $b \leftarrow b + c$

3: for $i = 0, \ldots, 255$ do

4: $x \leftarrow s_i$

5: $a \leftarrow f(a, i) + s_{i+128 \bmod 256}$

6: $s_i \leftarrow a + b + s_{x>>2 \bmod 256}$

7: $r_i \leftarrow x + s_{si>>10 \bmod 256}$

8: $b \leftarrow r_i$

9: end for

10: return r

Ушбу генератор бардошли генератор саналиб, ундаги мавжуд камчиликлар ISAAC+ генератор алгоритмида тузатилган. Ушбу генераторда бир неча марта назарий ҳужумлар амалга оширилган, аммо амалий томондан ҳужумга учрамаган .

1.4. Бир томонли фукцияларга асосланган калит генерациялаш алгоритмлари

ПТСКК генераторлари яратишда бир томонли функциялар кенг қўлланилади. Бир томонли функцияларнинг характерли хоссаларидан бири шундан иборатки, бу функциянинг қийматини аргументнинг берилган

қиймати бўйича ҳисоблаш полиниомиал-мураккабликкга эга бўлиб, функциянинг берилган қиймати бўйича бу қийматга мос бўлган аргумент қийматини ҳисоблаш экспоненциал мураккабликга эга ёки ҳисоблашнинг рационал алгоритми мавжуд эмас (ёки номаълум) [5].

FIPS-186 генератори. Бу алгоритм АҚШ миллий стандарти сифатида қабул қилинган бўлиб, DSA электрон рақамли имзо алгоритми учун махфий параметр ва калит ишлаб чиқиш учун мўлжалланган.

Бу алгоритмда бир томонли функциялар сифатида DES ва SHA-I алгоритмларидан фойдаланилган [5,14].

Кириш параметрлари: m – бутун сон, q -160 битли туб сон, b=160,

t = 67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0$_{16}$ – 160 битли сон;

$y_i = 0$;

s -бошлағич 160 битли тасодифий ва маҳфий сон.

Чиқиш параметрлари:

$x_1, x_2, x_3, \ldots\ldots, x_m$

Алгоритм қадамлари:

1. z_i=SHA-1((s +y_i)mod2^b);
2. x_i = G(t, z_i);
3. s = (1+s+x_i) mod2^b.

G: - бир томонли функция сифатида ишлатилган DES-шифрлаш алгоритми

кириш: $t = t_0||t_1||t_2||t_3||t_4$; $z_i = z_0||z_1||z_2||z_3||z_4$;

чиқиш: $w = w_0||w_1||w_2||w_3||w_4$.;

Функция:

1. $u_i = t_i \oplus z_i$, бу ерда i=0 дан 4 гача ўзгаради;
2. $b_1 = z_{(i+4)\text{mod}5}$, $b_2 = z_{(i+3)\text{mod}5}$, бу ерда i = 0 дан 4 гача ўзгаради;

   $a_1 = u_2$, $a_2 = u_{(i+1)\text{mod}5} \oplus u_{(i+4)\text{mod}5}$;

$A = a_1 || a_2$, $B = b_1 || b_2$;

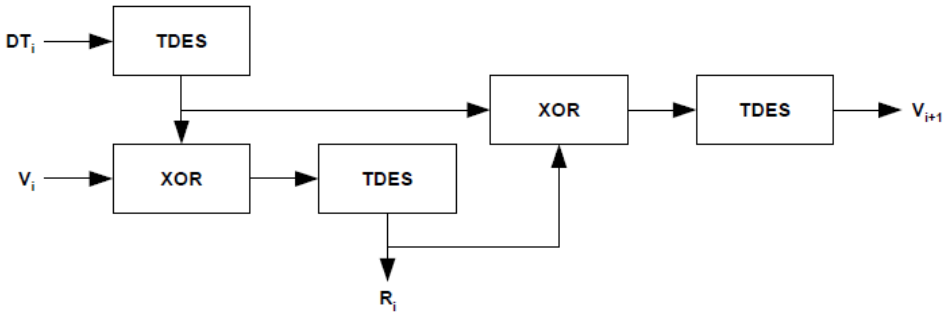
$y_i = E_B(A)$;

$y_i = L_i || R_i$;

3. $w_i = w_i + L_i \oplus R_{(i+2) \bmod 5} \oplus L_{(i+3) \bmod 5}$, бу ерда $i = 0$ дан 4 гача ўзгаради.
4. Натижа: $G(t, z_i) = w_0 || w_1 || w_2 || w_3 || w_4$.

Ушбу калит генераторининг бардошлилиги унда ишлатилган бир томонлама функциялар DES ва SHA-I га боғлиқ бўлади [5].

ANSI X9.17 генератори. Бу алгоритм АҚШ да псевдатасодифий кетма-кетлик ишлаб чиқувчи Миллий стандарт ҳисобланиб, FIPS (USA Federal Information Processing Standart) таркибига киради. Алгоритмда бир -томонлама функция сифатида учлик DES иккита K1, K2∈V64 калит ишлатилади: DESK1DESK2DESK1(64 бит) . Ушбу генератор учлик DES асосида ишлаб чиқилган бўлиб, уч марта учлик DES муолажасини қўллаш орқали амалга оширилади [14].



3- расм. ANSI X.9.17 генератори

Бу ерда:

TDES – учлик DES шифрлаш алгоритми бўлиб, шифрлашда иккита K1, K2∈V64 калитлардан фойдаланилади;

DT_i – кирувчи вақт параметри;

V_i - кирувчи 64-битли махфий калит бўлиб, у қуйидаги қуйидагича ҳосил қилинади:

$$V_{i+1}=TDES(TDES(DT_i)XOR(R_i));$$

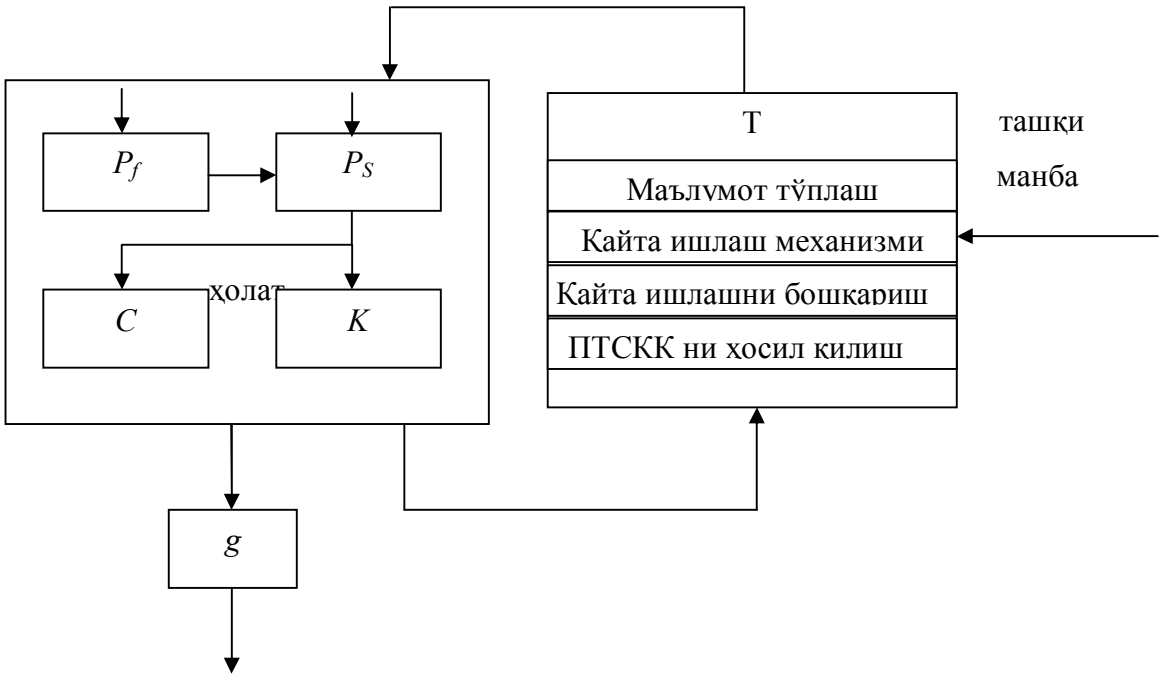
R_i – чиқувчи псевдотасодифий кетма-кетлик бўлиб қуйидагича ҳосил қилинади:

$$R_i=TDES(TDES(DT_i)XOR(V_i));$$

Yarrow-160 генератори. Yarrow-160 псевдатасодифий кетма-кетлик ишлаб чиқарувчи генератори Келси, Шнайер ва Фергюсон томонидан таклиф қилинган. Бу ерда учлик DES ва SHA-1 хэшлаш алгоритми ишлатилган [5,18].

Yarrow алгоритмида кирувчи тасодифий маълумотлар икки, тезкор қисм P_f ва секин йиғилувчи P_s қисмларга бўлиниб, хешланади. Кирувчи маълумотлар ҳақиқий тасодийиф бўлган маълумотлар манбаларидан олинади. Қисмларга маълумотлар тўплангандан сўнг, улар орқали *K* махфий калит ҳосил қилинади. Тасодифий сонлар кетма-кетлиги еса *C* ҳисоблагични *K* махфий калит билан шифрлаш орқали амалга оширилади.

Yarrow алгоритми қуйидаги тўртта ташкил этувчидан иборат [11]:



4-расм. Yarrow алгоритмининг умумий структураси

Yarrow калит генераторида хэш функция ва блокли шифрлашдан фойдаланилганлиги сабабли, криптоҳужумларга бардошли ҳисобланади.

Yarrow-160 алгоритмида хеш функция сифатида SHA-1 (Secure Hash Algorithm) ва блокли шифрлаш алгоритми сифатида эса учлик -DES (Triple Data Encryption Standard) фойдаланилган [11,18].

1. *Маълумотларни тўплаш* жараёнида ташқи манбадан кирган маълумотларни икки қисмга жамлаш амалга оширилади. Ташқи манба сифатида эса қуйидагиларни олиш мумкин:
   - махсус манбалар, яъни шахсий компьютерларнинг механик қурилмалар орқали ҳосил қилинади. Буларга сичқончанинг ҳолати, микрафон орқали қабул қилинадиган овоз частотаси, клавиатура тугмаларининг босилиш вақтлари ва ҳ.к;
   - компьютерда бажарилинаётган жараёнлар ҳақида маълумотлар;
   - шахсий компьтерга тегишли бўлган маълумотлар(компьтер номи, операцион тизим ҳақидаги ва ҳ.к);
   - Windows регисторлари ҳақидаги маълумотлар ва ҳ.к.

Ушбу манбалар орқали қабул қилинган маълумотлар SHA-1 хэшлаш алгоритми ёрдамида 160 битга келтирилади.

2. *Қайта ишлаш механизми* ёрдамида жамланган маълумотлар орқали махфий калит ҳосил қилинади ва қуйидаги босқичлардан иборат:

Дастлабки таёргарлик. Дастлаб $V_0:=P_f$ олинади. Бу ерда $P_f=h(i_f)$ тезкор қисмдаги маълумотнинг хеш қиймати.

Шакллантириш. Бу иккинчи босқич бўлиб, хеш қиймат қуйидаги йўл билан шакллантирилади:

$$V_i=h(V_{i-1}|V_0|i),$$ бу ерда $i=1,....,N_t$

Бу ерда | бирлаштириш белгиси ва $N_t \geq 0$ эса шаклланиш даврини кўрсатади. Бу жараён ҳужумга бардошли бўлиб қолмасдан, шунингдек генераторни маҳсулдорлигини камайтиради.

Калитни ҳосил қилиш. Бу босқичда дастлабки калит K ва хешлаш функцияси ёрдамида янги K' калит ҳосил қилинади. h' функцияси m

узунликдаги хеш қийматлардан k узунликдаги қисмини олиш учун фойдаланилади(Yarrow-160 да $m=160$ ва $k=192$).

$$K' = h'(h(V_{Nt}|K),k)$$

бу ерда $h'(M,k)$ қуйидагича ҳисобланади:

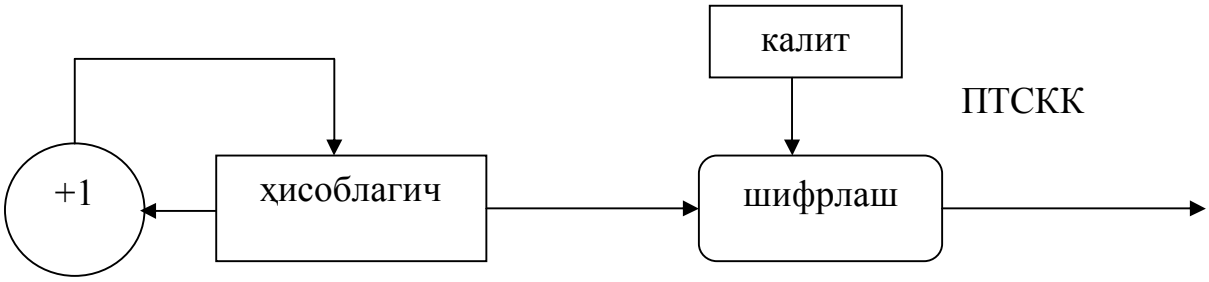
$$s_0 := M$$

$$s_i := h(s_0|\ldots|s_{i-1})$$

$h'(m,k)=(s_0|s_i|\ldots)$ нинг дастлабки k бити.

Ҳисоблагични янгилаш. Енг охирги қадам бу ҳисоблагични $C=E_K(0)$ га тенглашдир. Бу ерда E_K шифрлаш алгоритми бўлиб, K махфий калит билан шифрланади.

3. *Қайта ишлашни бошқариш* қисми эса ҳар сафар қисмларга(P_f ва P_s) маълумотлар тўпланганда калит ҳосил қилишни бошқаради.
4. *ПТСККни ҳосил қилиш* қисмида эса махфий калитдан фойдаланган ҳолда ҳисоблагич кўрсаткичи шифрланади ва ПТСКК ҳосил қилинади.



5- расм. Генерация жараёни

Yarrow алгоритмида $n(64)$ битли ҳисоблагичдан фойдаланилади ва кейинги қиймат қуйидагича олинади:

$$C_{m+1}=C_m+1(\mathrm{mod}\ 2^n)$$

ПТСКК эса қуйидагича ҳосил қилинади:

$$R_m=E_K(C_m).$$

Ушбу ПТСКК алгоритмида ТСКК генераторларидан фойдаланилган бўлиб, бу имконият алгоритмни ихтиёрий платформада фойдаланиш имконини беради. Ушбу алгоритм 160-битли ПТСКК ҳосил қилади. Ушбу

алгоритмнинг бир камчилиги кам қуваттли бўлиб, бу кўрсаткич унда ишлатилган хешлаш алгоритми ва блокли шифрлаш алгоритми бардошлигига боғлиқ бўлади.

Бир томонли мантиқий функцияларга асосланган генератор. Бу генератор алгоритмининг асосини тўртта 4 аргументли мантиқий функция:

1) $F_i = X_i\ Y_i\ (Z_i \oplus W_i) \oplus W_i$ - байтлар устида F мантиқий акслантириш;

2) $G_i = W_i\ (X_i Z_i \oplus Y_i) \oplus Z_i W_i$ - байтлар устида G мантиқий акслантириш;

3) $R_i = X_i Y_i Z_i \oplus Z_i W_i \oplus X_i W_i \oplus Y_i W_i$ - байтлар устида R мантиқий акслантириш;

4) $V_i = Y_i W_i Z_i \oplus Z_i W_i \oplus X_i$ -байтлар устида V мантиқий акслантириш;

1-жадвал

Сиқиш жадвали

Y/X	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	5	13	6	11	1	10	15	8	0	4	7	9	2	12	3	14
1	8	7	2	14	15	3	11	6	1	12	13	10	5	4	9	0
2	14	2	13	4	12	7	1	11	6	9	0	5	3	10	8	15
3	0	14	9	12	3	13	7	4	15	6	5	1	11	2	10	8
4	3	10	7	2	4	12	9	1	14	13	15	8	0	5	11	6
5	2	3	1	8	0	14	5	9	12	11	6	7	10	15	13	4
6	10	4	14	15	9	5	8	2	11	0	1	3	12	6	7	13
7	11	9	10	1	6	4	13	15	3	5	14	0	8	7	2	12
8	1	0	3	7	13	11	10	12	9	14	4	6	15	8	5	2
9	4	8	11	9	14	6	2	5	10	3	12	15	7	13	0	1
10	9	12	15	0	2	1	14	10	5	8	11	13	4	3	6	7
11	6	11	8	13	7	9	0	3	4	15	10	2	14	1	12	5
12	15	1	0	5	10	8	3	7	13	2	9	12	6	14	4	11

13	12	5	4	10	11	2	6	13	8	7	3	14	1	0	15	9
14	7	15	12	6	5	0	4	14	2	10	8	11	13	9	1	3
15	13	6	5	3	8	15	12	0	7	1	2	4	9	11	14	10

5) Ўлчами 16x16 бўлиб, элементлари ярим байтдан иборат бўлган (0 дан 15 гача сонларни текис тақсимотидан иборат) сиқиш жадвали (СЖ)(1-жадвал) ташкил этади. [5]

Бир байт $b = b_1b_2b_3b_4b_5b_6b_7b_8$ иккита $x = b_1b_2b_3b_4$ ва $y = b_5b_6b_7b_8$ ярим байтларга ажратилади, сўнгра бу ярим байтларнинг қийматлари бўйича мос равишда сатр ҳамда устун тартиб сонлари (номерлари) топилиб, шу сатр ва устунлар кесишган жойдаги соннинг иккилик саноқ системасидаги ярим байтли ифодасини бир байтли $b = b_1b_2b_3b_4b_5b_6b_7b_8$ ифоданинг сиқиш жадвали орқали акслантириш натижаси сифатида қабул қилинади.

Босқичлари:

1. K – 256 битли калит 4 байтли 8 та қисмга ажратилади;

2. Ҳар бир 4 байтли калитдан мантиқий акслантиришлар орқали 4 байтли блок ҳосил қилинади;

3. Ҳосил қилинган 4 байтли блок 2 марта СЖ орқали акслантирилиб, 1 байтли блок ҳосил қилинади;

4. Генератор алгоритмида кўрсатилган қоида бўйича K - калитнинг байтлари аралаштирилиб, 1-4 босқичлар такрорланади.

Алгоритм акслантиришлари кетма-кетлиги.

Тасодифий бўлган 256 битли калит $K = k_1, k_2, \ldots k_{255}, k_{256}$ киритилади ва бу калит 32 битли ёки 4 байтли 8 та блокка ажратилади:

1) $K = k_1, k_2, \ldots k_{128}, \ldots, k_{128+32L}$(бит)$= x_1, x_2, \ldots, x_{16}, \ldots, x_{16+4L}$(байт)$=$

$= y_1, y_2, \ldots, y_{4+L}$(32 битли) -блоклар, бу ерда L=0, 1, 2, … .

2) Ҳар бир $x_i = k_{1+8(i-1)}\, k_{2+8(i-1)p}\, k_{3+8(i-1)} \ldots k_{8+8(i-1)}$, i=1,2,…16+4L, бўлиб,

агар L=4 бўлса, у ҳолда

$$K = k_1, k_2, \ldots k_{255}, k_{256} = x_1, x_2, \ldots, x_{31}, x_{32} = y_1, y_2, \ldots y_7, y_8 .$$

3) Киритилган калит массиви 4 байтли (32 битли) блокларга ажратилади:

$$y_{j4x1} = (x_{1+4(j-1)}, x_{2+4(j-1)}, x_{3+4(j-1)}, x_{4+4(j-1)}) .$$

Биринчи 4 байтли блокнинг (X_i , Y_i, Z_i , W_i) ҳар бир байтлари устида мантиқий акслантиришлар бажариб янги 4 байтли ($F_i \parallel G_i \parallel R_i \parallel V_i$) блокка эга бўламиз. Бу блок байтларини сиқиш жадвалидан ўтказилади ва икки баробар сиқилган 2 байтли блокка (A||B) эга бўлинади. (A||B)-блокни ҳам сиқиш жадвалидан ўтказилиб, 1 байтли D-блок ҳосил қилинади. D-блокни гамма кетма-кетлик элементи сифатида қабул қилинади. Бошланғич берилган 256 битли калитдан шу тариқа 8 байт (64 бит) гамма кетма-кетлик элементлри олинади. 256 битли калитни ўзгартириш тескари боғланиш акслантириши билан амалга оширилади.

Калитни ўзгартирувчи тескари боғланиш акслантириши қуйидагича бажарилади:

1) Гамма кетма-кетлик ҳосил қилиш жараёнида ҳосил қилинган оралиқ A ва B -блокларни mod 256 бўйича S ўзгарувчига йиғиб борилади:

$$S=(S+A+B)\text{mod}256.$$

2) Сўнгра, 64 бит гамма кетма-кетлик олиниб бошланғич калитнинг барча байтлари ишлатилгандан сўнг K -калитнинг биринчи b_0-байти орқали

$$Q=(b_0+S)\text{mod}32$$

ҳисобланади.

3) “Q” индексда турувчи b_Q қийматини ўзгартирилади:

$$b_Q=(b_Q+S)\text{mod}256.$$

4) Шундан сўнг ўзгартирилган 256 битли K -калитни S қийматга циклик сурилади:

$$K = K << S$$

ва натижада ўзгартирилган янги 256 битли калитга эга бўлинади.

Бу калит кейинги 8 байтли гаммаларни ҳосил қилишда ишлатилади.

K-калит 2^{256} та ҳар-хил қийматга эга бўлиши мумкин. Ҳар бир қийматдан 2^{64} та гамма кетма-кетлик битлари ҳосил қилинади ва гамма кетма-кетлик даври узунлиги $2^{256} \cdot 2^{64}=2^{320}$ битни ташкил қилади.

Алгоритм асосида қуйидаги акслантиришлар ётади:

$(b_0+S)\text{mod}32$ - mod32 бўйича йиғинди;

$(b_Q+S)\text{mod}256$ - mod256 бўйича йиғинди;

$K <<S$ - S нинг қиймати бўйича K-калитни циклик суриш;

$F_i=X_iY_i(Z_i \oplus W_i) \oplus W_i$ - байтлар устида мантиқий акслантириш бажариш;

$G_i= W_i (X_iZ_i \oplus Y_i) \oplus Z_iW_i$ - байтлар устида мантиқий акслантириш;

$R_i=X_iY_iZ_i \oplus Z_iW_i \oplus X_iW_i \oplus Y_iW_i$ - байтлар устида мантиқий акслантириш;

$V_i=Y_iW_iZ_i \oplus Z_iW_i \oplus X_i$ - байтлар устида мантиқий акслантириш;

СЖ(F_i), СЖ(G_i), СЖ(R_i), СЖ(V_i) - 1 байтли блокларни ярим байтларга сиқиш;

A=СЖ(F_i) || СЖ(G_i) ва B=СЖ(R_i) || СЖ(V_i) - ярим байтли блоклар конкатенацисидан бир байтли блоклар олиш;

D=СЖ(A)||СЖ(B) - ярим байтли сиқиш натижаларининг конкатенацияси(1 байт);

$C_L= M_L \quad D \bigoplus$ XOR амали орқали гамма блок битларини очиқ маълумот блоки мос битларига қўшиш.

Кириш параметрлари:

K [256] - 256 битли калит;

M[d]- очиқ маълумот блоклари;

СЖ[16x16] - сиқиш жадвали.

Натижа: C_L - шифрланган маълумотдан иборат.

Бир томонлама мантиқий функцияларга асосланган генераторларнинг бардошлилик даражаси ўрта ҳисобланади.

1.5. Комбинациялашган псевдотасодифий сонларни генерациялаш алгоритмлари

Юқорида ПТКК ишлаб чиқарувчи генераторларнинг тизимли-назарий ёндашув асосида ва мураккабликка асосланган назарий ёндашув йўналишлари таҳлил қилинди. Шу йўналишларда яратилган базавий генераторлардан фойдаланиб уларни комбинациялаш асосида янги генераторлар яратиш усуллари, комбинациялаш асосидаги псевдотасодифий кетма-кетлик генераторлари яратиш йўналиши деб аталади.

Бу ёндашувда мавжуд ПТКК ишлаб чиқувчи генераторлар асосидаги акслантиришларнинг (алгоритмларнинг) бирлаштирилиши (комбинацияси) асосида янги генератор яратилади. Бу генераторнинг криптобардошлилиги унинг таркибидаги ҳар бир акслантиришнинг ва алгоритмларнинг мураккаблиги билан экспоненциал боғлиқдир.

Комбинациялаш асосида қурилган ПТКК генераторларини яратиш: полиномиал мураккабликка эга акслантиришларни комбинациялаш, тасодифий параметрли алгоритмларни комбинациялаш, Макларен-Марсальи ва бошқа шулар каби усуллар орқали амалга оширилади.

Хозирги пайтгача қўлланилиб келинаётган силжитиш регистрларига асосланган ПТКК ишлаб чиқувчи генераторларнинг криптобардошлилиги полиномиал мураккабликка асосланган силжитиш регистрларини комбинациялаш орқали ишлаб чиқилган.

Макларен-Марсальи усули. Бу усул ёрдамида комбинациялаш учун иккита ПТКК ишлаб чиқувчи генераторлар (G_1, G_2) олинади. Бу генераторлар мос равишда $\{x_i\}$ ва $\{y_j\}$ кетма-кетликни ҳосил қилсин. Бирор k (яъни $1\times k$) етарли катта ўлчамга эга, T жадвални мос равишда G_1 генератор натижаси бўлган x_i $(i=1,2,...,k)$ қийматлар билан тўлдириб чиқилади. Шундан сўнг тўлдириб чиқилган жадвалдаги қийматлар бўйича

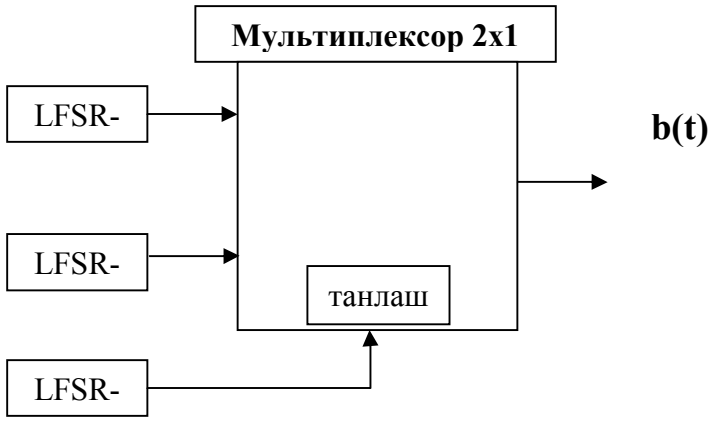
G_2 генератор натижаси бўлган $\{y_j\}$ индексда турувчи x_{y_j} $(1 \le y_j \le k)$ элементни ПТКК гаммаси элементи сифатида узатиш мумкин. Демак бу усулда «тасодифий» $\{x_i\}$ сонлар билан тўлдирилган T жадвалдан «тасодифий» $\{y_j\}$ индекс буйича қийматларни танланади. Бу усулда таклиф қилинган комбинациялашган генераторлардан бири Chameleon генераторидир. Бу генератор асосида яратилган узуликсиз шифрлаш алгоритмининг хусусиятли томони шундан иборатки, бардошли криптотизимни ишлатган ҳолда калитнинг кам қисми ўзгариши очиқ маълумотнинг ҳам оз ўзгаришига олиб келишидадир. Криптотизимга бундай талаб қўйилиши хозирги пайтда тижорат телевидениеси ва тижорат радиотармоқларининг кўпайиб бориши билан келиб чикди. Тижорат телевидениеси сотиб олинган ва интеллектуал мулк ҳисобланган қўшиқ ва кинофильмларни шу канал орқали шифрлаб узатади. Тижорат канали абонентлари қабул қилиб олинган қўшиқ ва кинофильмларни эшитишлари ва кўришлари мумкин, дискларга ёзиб олиб кўпайтириш ва сотиш имкониятларини чегаралашга имкон беради. Баъзи абонентлар қонунга хилоф равишда дешифрланган қўшиқ ва кинофильмларни дискларга ёзиб олиб сотувга чиқаришлари мумкин. Тижорат телевидениеси сотувга чиқарилган диск қайси абонент томонидан чиқарилганини ўзлари берган калитдаги битлар ўзгаришига қараб дарҳол қайси абонент томонидан дешифрланганини аниқлаб олишга муваффақ бўлади [5].

Иҳтиёрий узлуксиз шифрлаш алгоритми ҳамда 2^{16} та 64 битли элементи бўлган умумий ўлчами 512 килобайтли B массив олинади. Танлаб олинган узлуксиз шифрлаш алгоритмидан чиққан 64 битли кетма-кетлик 4 қисмга бўлиниб, ҳар бир 16 битли қисмга мос келувчи индексларда турувчи бошланғич B массив элементлари ўзаро XOR амали билан қўшилиб, 16 битли гамма ҳосил қилинади. Бу гамма очиқ маълумотнинг 16 битли блоки билан XOR амали билан қўшилиб, натижада

шифрмаълумот ҳосил қилинади ва эфирга узатилади. Бу ерда B массив калит сифатида ҳар бир абонентга алоҳида-алоҳида берилади. Лекин ҳар бир берилган 512 килобайтли B массивнинг фақат иҳтиёрий битта бити ўзгартирилиб берилади. Натижада ҳар бир абонентга бир хил шифрматн келгани билан 512 килобайтли B массивдаги бир бит ўзгариши ҳисобига дешифрланган ҳар бир 512 килобайтли очиқ маълумотларда 4 та бит ўзгарган ҳолатда дешифрланади. Мана шу ўзгаришлар орқали қайси абонент дешифрлагани аниқланади [5].

Тасодифий параметрли комбинациялашган генераторлар. Бу усулда генераторларни комбинациялаш учун иккита G_1, G_2 генератор олинади. G_1 генераторни гамма ҳосил қилувчи ва G_2 ни параметр ўзгартирувчи деб олинади. Агар бу G_1 генераторнинг бошлангич параметрларини a_1, b_1, c_1 деб олинадиган бўлса ва бошланғич қиймат x_0 орқали $G_1^1(x_0) = x_1$ ҳисобланади, x_2 ни ҳисоблаш учун эса G_1 генераторнинг бошланғич параметрларини иккинчи G_2 генераторнинг генерация қилган қийматларига қараб $a1=e_i$, $b1=f_i$, $c1=g_i$ ўзгартирилади, бу ерда$\{e_i , f_i , g_i \}=G_2(x)$. Натижада, параметрлари тасодифий ўзгарувчи комбинациялашган генератор G_1 га эга бўлинади [5].

Полиномиал комбинациялаш. Бу турда кўпроқ силжитиш регистрларини комбинациялаш қўлланилган.



6-расм. Мултиплексорли комбинациялашган регистр

Бир нечта силжитиш регистрлари олинади ва шу регистрлардан бири бошқарувчи регистр сифатида чиқиш бити қайси регистрдан олинишини аниқлаб беради. Геффе генераторида учта регистр ишлатилган бўлиб биринчи регистр қолган регистрларни бошқаради.

Геффе генератори назарий жиҳатдан бардошли бўлганлиги билан корреляцион хужум турига бардошли эмас. Амалий кузатувлар шуни кўрсатадики, 75% гамма битлар битта силжитиш регистрининг чиқишига тенгдир. Агар тескари боғланиш примитив кўпҳади маълум бўлса бу генераторнинг бошланғич қийматини аниқлаш мумкин бўлади. Агар тескари боғланиш примитив кўпҳади учҳад бўлиб, силжитиш регистларининг узунлиги n бўлса $37n$ битли чиқиш гаммасига эга бўлган ҳолда учта силжитиш регистрларининг ҳолатини аниқлаш мумкин бўлади. Полиномиал комбинациялаш усулида корреляцион хужум турига бардошлилигини ошириш учун, бошқарувчи мультиплексор ёки генератор ҳосил қилган ПТКК тасодифийлик даражаси юқори ва статистик кўрсаткичлари текис тақсимланган бўлишини таъминлаш зарур [5].

1.6. Симметрик шифрлаш алгоритмларида фойдаланилган калит генерациялаш алгоритмларининг қиёсий таҳлили

Юқоридаги бобларда ҳозирда симметрик шифрлаш тизимларида бардошли калитларни ишлаб чиқарувчи ПТСКК генераторлари кўриб чиқилди.

Келтирилган фикр ва мулохазаларнинг якуни сифатида амалга қўлланилиб келинаётган мавжуд генераторларнинг туркумлари 7-расмда келтирилган. Ушбу туркумланиш ПТСКК генераторларини математик асоси ва ундаги функция, амаллардан келиб чиқишга асосланган.

Кўриб ўтилган генераторлар бардошлилиги таркибидаги математик амаллар орқали белгиланади. 2 - жадвалда эса псевдотасодифий сонлар

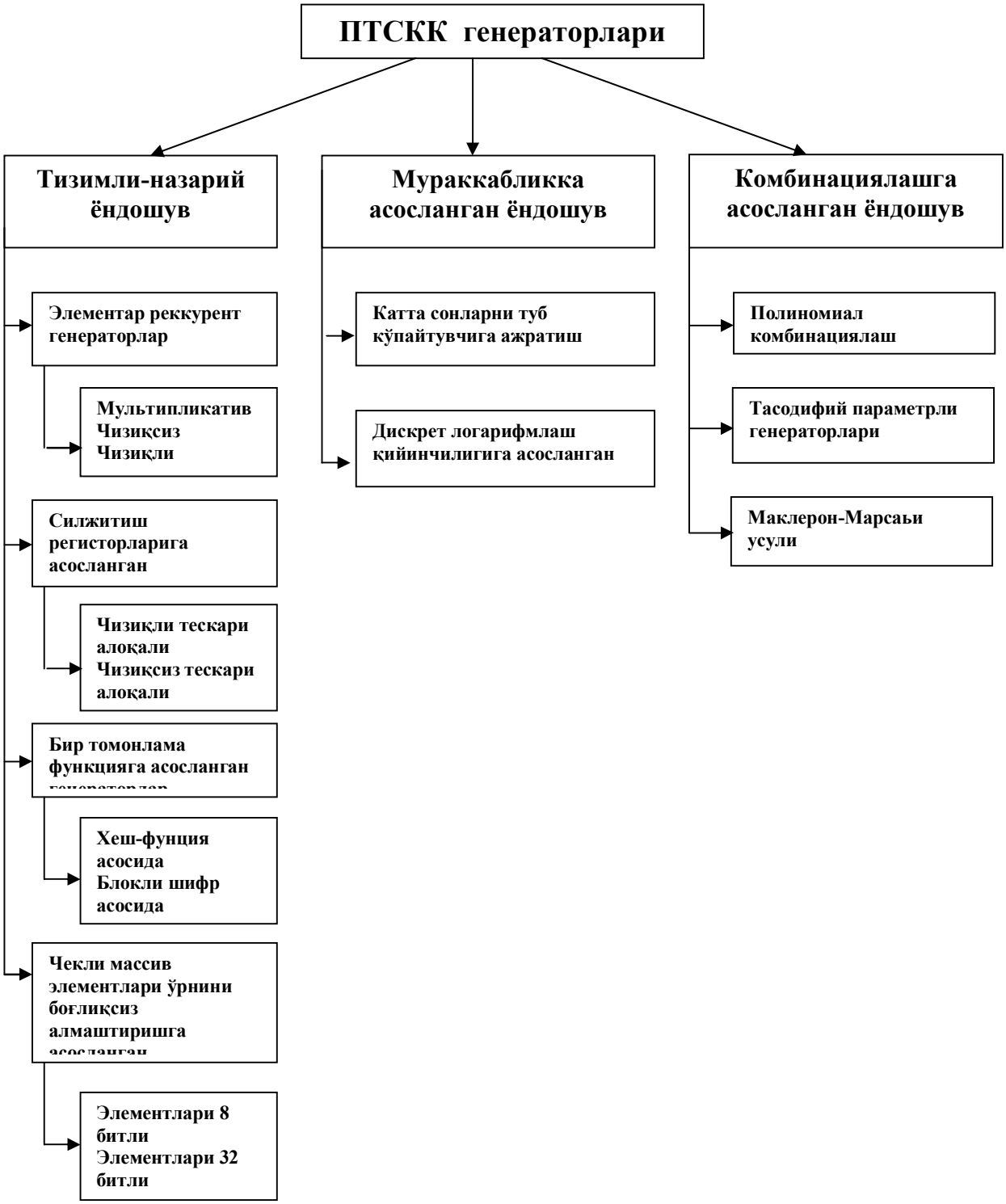
генераторининг таркибидаги амаллар, калит узунлиги ва криптобардошлиги хусусятлари бўйича таҳлили кўрсатилган.

2 – жадвал

ПТСКК генераторлари хусусиятлари

Алгоритм номи	**Калит узунлиги (бит)**	**Таркибидаги амаллар**	**Крипто-бардошлиги**	**Хусусятлари**
Конгруэнт генераторлар	<64	Кўпайтириш, mod N	$<2^{64}$, кичик	Параметрлар танлаш мураккаб, тезлиги паст
Реккурент генераторлар	<64	Кўпайтириш, mod N	$<2^{64}$, кичик	Параметрлар танлаш мураккаб, тезлиги паст
Гиффорд алгоритми	64	Силжитиш регистри	2^{64}, кичик	Такрорланмас даври кичик
RC-4, ISAAC	2048 битгача	Mod256, ўрин алмаштириш	2^{1700}, юқори	Тезлиги юқори, патентга ега
SEAL, WAKE	128	Ўнга суриш, ХОР	2^{160}, ўрта	Тезлиги юқори
A5, Геффе	64	Силжитиш регистрлари комбинатсияси	2^{64}, кичик	Тезлиги юқори
RSA, BBS, Шамир, Блюм-Микали	1024-2048	Катта сонларни даражага ошириш	2^{1024}, юқори	Тезлиги паст, фақат калит ишлаб чиқиш учун
ANSI X9.17 FIPS-186 YARROW-160	64-160	3DES, 2DES, SHA-1 алгоритмлари	2^{128}, ўрта	Тезлиги паст, фақат калит ишлаб чиқиш учун

Ушбу жадвалдан кўриниб турубдики, суриш амалларига, ўрин алмашиниш амалларига ва бир томонлама функцияларга асосланган генераторлар бошқаларига нисбатан тезлиги ва бардошлилик хусусиятлари бўйича яхшироқдир.



7 - расм. Псевдотасодифий сонлар генератори туркумлари

Юқоридаги кўрсаткичлардан кўриниб турибдики симметрик тизимлар учун псевдотасодифий кетма-кетликлар ишлаб чиқарувчи бир томонлама функцияларга асосланган калит генерациялаш алгоритмлари бошқаларига қараганда криптобардошли ҳисобланар экан. Ушбу ҳусусият эса қуйидаги параметрлар орқали таъминланган:

- таркибидаги математик бир томонлама функциялар (DES, 3DES, SHA1 ва ҳ.к);
- кирувчи параметрларнинг ҳар калит генерациялаш вақтида янгиланиши;
- кирувчи параметрлар эса компьютер хотирасидаги тасодифий саналган параметрлар орқали ҳосил қилиниши ва ҳ.к.

Юқорида келтирилган бир томонлама функциялар орқали яратилган ПТСКК генераторлари, ANSI X9.17, FIPS-186 ва Yarrow-160 ларда мавжуд камчиликларни таҳлил қилсак.

ANSI X9.17 генераторида қуйидаги камчиликларлар мавжуд [13,14]:

- амалий томондан исботланмаган бўлсада шифрлаш алгоритми ҳисобланган TDES даги камчиликлар;
- вақтни синхронизациялашга асосланганлиги ва уни ҳосил қилиш мумкинлиги;
- шифрлашда фойдаланилган калитларни ўзгармаслиги;
- $seed_{i+1}$ ни ҳосил қилиш $seed_1$ га боғлиқлиги ва ҳ.к;

Ушбу санаб ўтилган камчиликларни бартараф этишда қуйидаги усуллардан фойдаланиш мақсадга муофиқ [14]:

- криптобардошли саналган симметрик шифрлаш алгоритмларидан фойдаланиш (ГОСТ 28147-89, AES ва ҳ.к);
- вақтни синхронизациялашни яхшилаш ёки бу жараённи бошқа тасодифий бўлган кириш параметр билан алмаштириш;
- доимий равишда шифрлаш алгоритмлари калитларини янгилаш;
- seed параметрни ҳосил қилишда тасодифий ҳисобланган манбалардан фойдаланиш.

FIPS-186 генераторида мавжуд камчиликлар [14]:

- DES шифрлаш алгоритмидан фойдаланилганлиги (ҳозирда бу алгоритмни амалий томондан ҳужум қилиш мумкинлиги);

- кирувчи параметр ҳисобланган y_i ни бошқариш мумкинлиги ва бошқа параметрлар билан боғлиқ камчиликлар.

Бу камчиликларни бартараф этишда қуйидаги ўзгартиришлар мақсадга муофиқдир:

- шифрлаш алгоритмини криптобардошли ҳисобланган бошқа симметрик шифрлаш алгоритми билан алмаштириш;
- криптобардошли хэш фукнциялардан фойдаланиш;
- дастлабки кирувчи параметрларни тасодифий бўлган тизим параметрлари билан алмаштириш;
- математик амалларни криптобардошли бўлган ифодалар билан алмаштириш.

Yarrow-160 генераторида мавжуд камчиликлар Bruce Schneier ва Niels Ferguson томонидан бартараф этилиб, янги Фортуна номи билан патентланди [11].

Фортуна генератори ҳам yarrow-160 да мафжуд босқичларни ўзида сақлаб қолган ва шунингдек қуйидаги ўзгаришлар киритилган:

- тасодифий ҳисобланган манбалар миқдори ортирилган, яъни кирувчи параметрлар барчаси тизим ресурсларидан тўпланган маълумотлар орқали ҳосил қилинади;
- DES шифрлаш алгоритми 21 асрнинг энг криптобардошли алгоритми ҳисобланган AES билан алмаштирилган ва баъзи математик ифодаларда ўзгаришлар киритилган.

ПТСКК ҳосил қилишда генераторларга кирувчи дастлабки маълумотлар (калитлар) тасодифийлиги муҳим аҳамиятга эга. Чунки фойдаланувчи томонидан киритилган дастлабки калитлар қандайдир қонуниятга бўйсинади ёки хотирада сақлаб қолинади ва оқибатда бузғунчи томонидан тизим бузилиши мумкин. Бу камчиликни олдини олиш мақсадида кирувчи параметрларни компьютер манбасидан олинган тасодифий қийматлардан фойдаланиш катта самара беради. Бу усулнинг

бардошлилиги бузғунчи тизимни тўлиқ ҳолда бошқара олмаслик имконяти ёқлиги билан белгиланади. Одатда ПТСКК генераторларида кирувчи маълумот сифатида операцион тизим ҳақидаги маълумотлар, қурилмалар ҳақидаги маълумотлар ва ҳ.к.лар олиниши мумкин.

Ушбу илмий тадқиқот ишида кўриб чиқилган ПТСКК генераторларида кирувчи параметрлар сифатида тизимдаги қуйида кўрсатилган параметрлар олинган [11].

Юқори ўзгарувчили вақт(high performance timer). Ҳозирда замонавий ШК да юқори ўлчамли аниқ вақтни санаш имконяти мавжуд. Бу қиймат маълум вақт оралиғидаги вақтни санаш частотасини ҳисоблайди. Ушбу қиймат ШК процессорининг частотаси билан бир хил эмас.

WINDOWS регистерларидаги маълумотлар. WINDOWS операцион тизимида регисторлар жуда кўп маълумотларни ўзида сақлайди. Ушбу маълумотлар тасодифий бўлмасада, бузғунчи томонидан ушбу маълумотларни барчасини қўлга киритиш ва қайси жараёнда қайси регистер қийматидан фойдаланишни аниқлаш жуда мураккаб.

Жараёнлар ҳақидаги маълумотлар(process information). WINDOWS операцион тизимида жараёнлар кўплаб маълумотларни ўзида сақлайди. Ушбу маълумотларни сиз Task Manager ойнаси орқали билишингиз мумкин. Улар қуйидаги маълумотлар бўлиши мумкин:

- Process Name;
- Process id;
- Memory Usage;
- Page Faults;
- Handles;
- Threads;
- I/O Reads;
- I/O Read Bytes;
- I/O Writes;

- I/O Write Bytes;
- I/O Other;
- I/O Other Bytes.

Ушбу маълумотларни тўплаш маълум вақт оралиғи билан тизим тингланади ва маълумотлар тўпланади.

Тизим ҳақидаги маълумотолар. Ушбу маълумотлар ўзгармас маълумотлар саналиб, ҳар бир жараёнда фойдаланилади. Булар қуйидагилар:

- Шахсий компьютер (ШК) номи;
- WINDOWS версияси;
- Тизим маълумотлари;
- Мавжуд қурилмалар ҳолатлари;
- MAC адрес маълумоти;
- Автоматик юкланувчи дастурлар ҳақидаги маълумот;
- Хотира маълумотлари.

Пинглаш. Бошқа ШКни тармоқ орқали пинглаш натижасида олинган маълумотлар, пинглаш вақтлари тасодифий ҳисобланиб, уларни юқори ўзгарилувчили вақт билан ҳисоблаш билан бардошли манбага эга бўлиш мумкин.

Ушбу олинган параметрлар тасодифий бўлмасада, бузғунчи томонидан бошқариш қийин бўлган манбалардир. Ҳар бир кирувчи параметрлар хэшланиб, олинган барча параметрлар бириктирилиб, кейин яна хэшланади. Бу амаллар кираётгпган параметрларни тасодифийлик даражасини яна бир марта оширади.

I боб бўйича хулосалар

Магистрлик диссертциясининг биринчи бўлими ПТСКК генераторлари таҳлили ва уларнинг криптографик тизимларда тутган ўрни келтирилган. Ушбу бўлимда қуйидаги натижалар олинди:

- калит генерациялаш алгоритмлари криптографик акслантиришларининг хусусятларига ва қўлланиладиган амаллардан фойдаланиш қоидаларига кўра, уларни синфлаш масаласи кўриб ўтилди;
- тасодифий ва псевдотасодифий сонларни генерациялаш принцплари ва усуллари келтириб ўтилди;
- мавжуд калит генерациялаш алгоритмлари классификацияси келтирилди;
- симметрик шифрлаш алгоритмларида фойдаланилган калит генерациялаш алгоритмларининг қиёсий таҳлили келтирилди;
- келтирилган таҳлил натижаларга кўра бир томонлама фунуциялар асосида қурилган ва байтлар ўрнини алмаштиришга асосланган ПТСКК генераторлари бошқаларига кўра криптобардошли топилди ва шунингдек улардаги камчиликлар келтирилиб ўтилди;
- бир томонлама функцияларга асосланган алгоритмларнинг камчиликларини бартараф этиш усуллари ва алгоритмлари келтирилди;
- ПТСКК ҳосил қилишда генераторларга кирувчи параметрларни ҳосил қилишда, тизимнинг тасодифий параметрларидан фойдаланиш усуллари ва уларни жамлаш усуллари келтирилди.

II боб. Симметрик шифрлаш алгоритмлари учун ишлаб чиқилган калитларни тасодифийликка текширувчи тестларнинг таҳлили

2.1. Статистик тестлар

Ушбу бўлимда ПТСКК тасодифийликка синовчи статистик тестлар ва уларнинг қиёсий таҳлили келтирилган.

Тасодифийликка текширувчи тестлар 2 хил бўлади [5]:

График тестлар - График тестлар фойдаланувчига текширилаётган кетма-кетликнинг маълум бир график боғлиқлиги ҳақидаги маълумотни бериб, у бўйича текширилаётган кетма-кетлик хоссалари тўғрисида хулоса чиқариш имкониятини беради.

Баҳолаш тестлари - Баҳолаш тестлари текширилаётган кетма-кетлик статистик хоссаларини таҳлил қилиб, унинг чин тасодифийлик даражаси ҳақида хулоса чиқариш имкониятини беради.

8-расм. Тасодифийликка текшириш тестлари турланиши

ПТСКК тасодифийликка текширишда тестлар ва тестлар тўпламидан фойдаланилади. Буларга DIEHARD, NIST Special Publication 800-22 тестар тўпламини мисол тариқасида олишимиз мумкин. Қуйидаги бўлимда кенг тарқалган Хи-квадрат, Колмогорф-Смирнов тести ва NIST Special Publication 800-22 тўпламига оид тестлар кўриб ўтилган.

Хи – квадрат тақсимоти ва унинг дастурий таъминоти. Бирор ўтказилаётган тажриба натижаларининг барча мумкин бўлган ҳолатлари $y_1, y_2, ..., y_k$, дан иборат ва уларнинг сони *k* га тенг бўлиб, бу тажриба бир-бирига боғлиқсиз ҳолда *n* марта ўтказилсин. Шунда, $y_1, y_2, ..., y_k$ - ҳолатларни, уларнинг *n* марта ўтказилган тажрибада, бир хил сонда такрорланишидан (текис тақсимотдан ёки бир хил частотага эга бўлишдан) қанчалик четланганлигини баҳолаш масаласини ечилишини кўриб ўтилади [5].

Бунинг учун қуйидагича белгилашлар киритилади:

p_s - эксперимент натижаси y_s бўлишининг эҳтимоллик қиймати;

Y_s - эксперимент натижаларининг y_s ҳолатга тегишлилари (тенглари) сони.

У ҳолда, бу белгилашларга нисбатан “Хи-квадарат” деб аталувчи тақсимот критерийси ушбу

$$V = \sum_{s=1}^{k} \frac{(Y_s - np_s)^2}{np_s},$$

формула орқали аниқланади.

Агар тажриба *n* мартадан бир неча марта ўтказилганда, ҳар доим $y_1, y_2, ..., y_k$ - ҳолатлар тенг Y_i мартадан такрорланса (текис тақсимланган ёки бир хил частотали бўлса), яъни $Y_1 = Y_2 = ... = Y_k$ бўлса, у ҳолда $p_1 = p_2 = ... = p_k = \frac{1}{k}$, деб ҳулоса қилинади ва

$$V=\sum_{s=1}^{k}\frac{\left(Y_s-\frac{n}{k}\right)^2}{\frac{n}{k}}=\sum_{s=1}^{k}\frac{\left(\frac{n}{k}-\frac{n}{k}\right)^2}{\frac{n}{k}}=0$$

тенглик ўринли бўлади. Бундай жараённинг илмий тадқиқот учун қизиғи йўқ. Аммо, амалдаги аксарият жараёнларда бундай ҳолат кузатилмайди, яъни бирор тажриба бир-бирига боғлиқсиз равишда *n* марта ўтказилиганда: $Y_1=Y_2=\ldots=Y_k=\frac{n}{k}$ ҳолат кўзатилмайди. Шунинг учун $y_1, y_2, \ldots, y_k$ - ҳолатларни рўй бериш эҳтимолликлари бир хил $p_1=p_2=\ldots=p_k=\frac{1}{k}$ бўлиб, тажриба бир-бирига боғлиқ бўлмаган равишда *n* марта ўтказилганда, бу ҳолатларнинг рўй бериши сони мос равишда $Y_1, Y_2, \ldots, Y_k$ бўлса, у ҳолда ушбу

$$V=\sum_{s=1}^{k}\frac{\left(Y_s-\frac{n}{k}\right)^2}{\frac{n}{k}}=\frac{k}{n}\sum_{s=1}^{k}\left(Y_s-\frac{n}{k}\right)^2$$

формула $Y_1=Y_2=\ldots=Y_k=\frac{n}{k}$ бўлган тенг тақсимотдан $Y_1, Y_2, \ldots, Y_k$ -тенг бўлмаган тақсимотни ўртача квадратик четланишини ифодалайди. Бу охирги формуладаги $\left(Y_s-\frac{n}{k}\right)$ - ифода бирор ўзгармас сон билан чегараланган, яъни

$$\left|Y_s-\frac{n}{k}\right|\le C=const\,.$$

Шунинг учун

$$V=\sum_{s=1}^{k}\frac{\left(Y_s-\frac{n}{k}\right)^2}{\frac{n}{k}}=\frac{k}{n}\sum_{s=1}^{k}\left(Y_s-\frac{n}{k}\right)^2 \qquad \le\frac{k}{n}\sum_{s=1}^{k}C^2=\frac{(kC)^2}{n}\to 0\,, \qquad \text{агар}$$

$n\to\infty$ бўлса.

Бу охирги формуладан, бирор генератор орқали ҳосил қилинган псевдотасодифий кетма-кетликнинг даври етарли узун бўлиб, барча мумкин бўлган битлар, байтлар ва қисм блокларининг тақсимоти деярли текис (тенг тақсимланган) бўлса, у ҳолда “Хи-квадарат” тақсимот критерийсининг бу кетма-кетликка нисбатан қиймати нолга яқин бўлиб, унинг тасодифийлик даражаси юқори ҳисобланади.

Қуйида эса стандарт DES, ГОСТ 28147-89 и AES-FIPS-197 ва бошқа симметрик шифрлаш алгоритмлари учун махфий калитни тасодифий қилиб генерация қилишнинг Хи-квадрат тақсимоти орқали қандай амалга оширилишини кўриб ўтамиз.

Берилган калит блоки бўйича қуйидаги жадвални тузиб оламиз:

Қиймат (s): 0 1 ;

Эҳтимоллик (p_s): $\frac{1}{2}$ $\frac{1}{2}$;

Кузатилаётган сон (Y_s): N_0 N_1 ,

бу ерда N_0 ва N_1 мос равишда калит блокида иштирок этувчи ноллар ва бирлар, $N_0 + N_1 = n$, орқали калит узунлигини белгилайди, масалан $n = 256$;

Кутилаётган сон (np_s): $\frac{n}{2}$ $\frac{n}{2}$;

Хи- квадрат тақсимоти формуласи бўйича:

$$V = \sum_{s=0}^{k-1} \frac{(Y_s - np_s)^2}{np_s}$$ ҳисобланади

Ушбу қаралаётган ҳолатда : $k = 2; s = 0,1; p_0 = p_1 = \frac{1}{2}; Y_0 = N_0; Y_1 = N_1; n = 256$;

у ҳолда, қуйидагича катталикка эга бўламиз:

$$V = \frac{(N_0 - 128)^2 + (N_1 - 128)^2}{128}.$$

Бу катталикни ҳисоблаш учун бизга Хи-квадрат тақсимотининг критик нуқталари жадвалидан фойдаланамиз [5].

3- жадвал

Хи-квадрат тақсимотининг критик нуқталари жадвали

	p=1%	p=5%	p=25%	p=50%	p=75%	p=95%	p=99%
N=*1*	**0.00016**	**0.00393**	**0.1015**	**0.4549**	**1.323**	**3.841**	**6.635**
N=*2*	**0.02010**	**0.1026**	**0.5754**	**1.386**	**2.773**	**5.991**	**9.210**
N =*3*	**0.1148**	**0.3518**	**1.213**	**2.366**	**4.108**	**7.815**	**11.34**
N =*4*	**0.2971**	**0.7107**	**1.923**	**3.357**	**5.385**	**9.488**	**13.28**
N =*5*	**0.5543**	**1.1455**	**2.675**	**4.351**	**6.626**	**11.07**	**15.09**
N =*6*	**0.8721**	**1.635**	**3.455**	**5.348**	**7.841**	**12.59**	**16.81**
N =*7*	**1.239**	**2.167**	**4.255**	**6.346**	**9.037**	**14.07**	**18.48**
N =*8*	**1.646**	**2.733**	**5.071**	**7.344**	**10.22**	**15.21**	**20.09**
N =*9*	**2.088**	**3.325**	**5.899**	**8.343**	**11.39**	**16.92**	**21.67**
N =*10*	**2.558**	**3.940**	**6.737**	**9.342**	**12.55**	**18.31**	**23.21**
N =*11*	**3.053**	**4.575**	**7.584**	**10.34**	**13.70**	**19.68**	**24.72**
N =*12*	**3.571**	**5.226**	**8.438**	**11.34**	**14.85**	**21.03**	**26.22**
N =*15*	**5.229**	**7.261**	**11.04**	**14.34**	**18.25**	**25.00**	**30.58**
N =*20*	**8.260**	**10.585**	**15.45**	**19.34**	**23.83**	**31.41**	**37.57**
N =*30*	**14.95**	**18.49**	**24.48**	**29.34**	**34.80**	**43.77**	**50.89**
N =*50*	**29.71**	**34.76**	**42.94**	**49.33**	**56.33**	**67.50**	**76.15**
N >*30*	$\nu + \sqrt{2\nu}x_p + \frac{2}{3}x_p^2 - \frac{2}{3} + O(\frac{1}{\sqrt{\nu}})$						
x_p =*8*	**-2.33**	**-1.36**	**-0.674**	**0.00**	**0.674**	**1.64**	**2.33**

Хи-квадрат критерийси жадвали $\nu = k - 1 = 2 - 1 = 1$, сатридан V қиймат жойлашиш оралиғини топамиз. Агар V қиймат жадвал устунининг $p = 25\%$ дан $p = 75\%$, оралиғида бўлса, у ҳолда псевдотасодифий генератор ёрдамида ҳосил қилинган калит блок битлари кетма-кетлиги тасодифий деб олинади.

Гарчанд псевдотасодифий генератор ёрдамида ҳосил қилинган калит блок битлари кетма-кетлиги тасодифийликка “Хи-квадрат” критерийси бўйича текширилганда ижобий жавоб олинган бўлса ҳам, ундан кўра ишончли ва мукаммал бўлган жавоб олиш учун қаралаётган битлар кетма-кетлигини бошқа мавжуд тасодифийлик тестларига ҳам текшириб кўриш лозим. Бу критерийларга текширув натижаларида қанчалик кўп ижобий

жавоблар олинса, критерий шунчалик яхши натижа деб қаралади. Бундан ташқари қуйидаги жараён ҳам тасодифийликка текширишда чиқариладиган хулосанинг ижобийлигига сезиларли даражада таъсир кўрсатади, яъни псевдотасодифий генератор ёрдамида ишлаб чиқилган калитларнинг амалиётда ўрнатилган бардошсиз калитлардан ўртача квадрат четланишининг ўртача қийматини ифодаловчи жараён.

Айтайлик, псевотасодифий генератор ёрдамида ҳосил қилинган калит блоки:

$k = k_1 k_2 ... k_n = k_1 k_2 ... k_{256}$, бу ерда $k_i \in \{0;1\}$, i=1,2, …, n = 256,

юқорида келтирилган критерий бўйича тасодифийликка текширилган ва қониқарли жавоб олинган. Амалиёт жараёнида шифртизимлар билан ишлашда аниқланган бардошсиз калитларни $k_{н1}, k_{н2}, ..., k_{нт}$, каби белгилаймиз.

Псевотасодифий генератор ёрдамида ҳосил қилинган калит блоки: $k = k_1 k_2 ... k_n = k_1 k_2 ... k_{256}$ ва амалиёт жараёнида бардошсиз деб топилган $k_{н1}, k_{н2}, ..., k_{нт}$, калитларнинг фарқи кўриб ўтилади:

$r_1 = k_{н1} \oplus k = r_1(1) r_2(1) .. r_{256}(1)$, бу фарқ бўйича мос равишда 0 ва 1 битлар сони $N_0(1)$, $N_1(1)$;

$r_2 = k_{н2} \oplus k = r_1(2) r_2(2) .. r_{256}(2)$, бу айирма бўйича мос равишда 0 ва 1 битлар сони $N_0(2)$, $N_1(2)$;

$r_m = k_{нт} \oplus k = r_1(m) r_2(m) .. r_{256}(m)$, бу айирма бўйича мос равишда 0 ва 1 битлар сони $N_0(m)$, $N_1(m)$; бу катталиклардан фойдаланган ҳолда, қуйидагиларни ҳисоблаймиз:

$$V_1 = \frac{(N_0(1) - 128)^2 + (N_1(1) - 128)^2}{128};$$

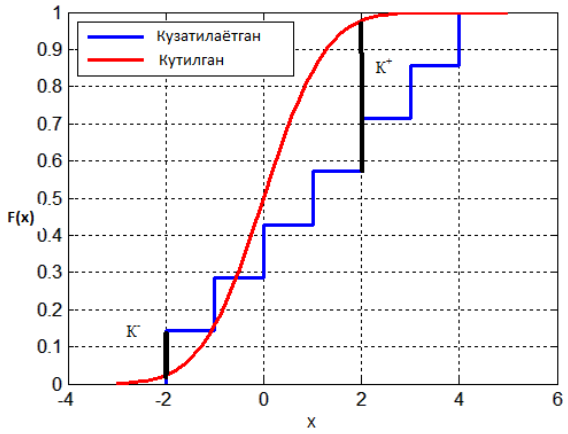
$$V_2 = \frac{(N_0(2) - 128)^2 + (N_1(2) - 128)^2}{128};$$

$$V_m = \frac{(N_0(m)-128)^2 + (N_1(m)-128)^2}{128};$$

$$V = \frac{V_1 + V_2 + ... + V_m}{m}.$$

Хи-квадрат критерийси жадвали $\nu = k - 1 = 2 - 1 = 1$, сатридан V - қиймат жойлашиш оралиғини топамиз. Агар V қиймат жадвал устунининг $p = 25\%$ дан $p = 75\%$, оралиғида бўлса, у ҳолда псевдотасодифий генератор ёрдамида ҳосил қилинган калит блок битлари кетма-кетлиги тасодифий деб олинади.

Калмагорф-Смирнов тести ва унинг дастурий таъминоти. Колмогороф-Смирнов(КС) тести дастлаб 1933 йилда А.Н.Колмогорф томонидан яратилди ва Н.В.Смирнов томонидан 1939 йилда ўзгартиришлар критилди. КС тести хи-квадрат тести ўринсиз бўлган ҳолларда қўлланилади, ва хи-квадрат тести билан биргаликда ишлатилади. Шунинг учун бу тест кўплаб амалий тестларнинг асоси ҳисобланади [23].



9 - расм. Колмогорф-Смирнов тести

Ушбу тест давомий бўлган жараёнларни текширишга мўлжалланган бўлиб, кузатилаётган $F_0(x)$ ва кутилган кўрсаткич $F_e(x)$ орасидаги фарқни текширади. Ушбу фарқ қанчалик кичкина бўлса икки ҳодиса ўртасидага фарқ шунчалик кичик бўлади.

Бу ерда: K^+ - кутилган ҳолатдан юқори бўлган максимал қиймат;

K^- - кутилган ҳолатдан паст бўлган максимал қиймат.

$$K^+ = \sqrt{n}\max_{x}(F_{куз}(x) - F_{кут}(x))$$

$$K^- = \sqrt{n}\max_{x}(F_{куз}(x) - F_{кут}(x))$$

Агар $K^+ < K_{[1-\alpha,n]}$ ва $K^- < K_{[1-\alpha,n]}$ кузатилётган ҳолат тесташдан ўтади. Бу ерда α муҳимлик даражаси ҳисобданади.

Юқоридаги формулага қуйидаги ўзгартиришларни киритамиз:

K^- ҳисоблаш учун $F_{кут}(x_{i+1})$-$F_{куз}(x_i)$

U(0,1) оралиқда $F_{кут}(x)=x$ ва $F_{куз}(x)=j/n$; $x>x_1,x_2,\ldots,x_{j-1}$ ҳоллар учун.

Юқоридаги ўзгартиришларни ҳисобга олган ҳолда қуйидагиларга эга бўламиз:

$$K^+ = \sqrt{n}\max_{j}(\frac{j}{n} - x_j)$$

$$K^- = \sqrt{n}\max_{j}(x_j - \frac{j-1}{n})$$

Мисол тариқасида қуйидаги кетма-кетликни оламиз.

30 та псевдотасодифий сон x_0=15 қиймат билан қуйидаги формула орқали ҳисобланди:

$$x_n = 3 * x_{n-1} \, mod 31$$

ва қуйидагилар сонлар ҳосил қилинди: 14, 11, 2, 6, 18, 23, 7, 21, 1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15.

Ушбу сонларни 31 га бўлиш орқали қуйидаги натижаларни оламиз:

0.45161, 0.35484, 0.064452, 0.19355, 0.58065, 0.74194,
0.22581, 0.67742, 0.03226, 0.09677, 0.29032, 0.87097,
0.61290, 0.83871, 0.51613, 0.54539, 0.64516, 0.93548,
0.80645, 0.41935, 0.25806, 0.77419, 0.32258, 0.96774,
0.90323, 0.70968, 0.12903, 0.38710, 0.16129, 0.48387.

$K_{[0.9;n]}$ қиймат n=30 ва α=0.1 учун 0.22 га тенг(2.2.жадвал).

4 - жадвал

Колмогорф-Смирнов тестининг ҳисоблашлар жадвали

j	x_j	$\frac{j}{n} - x_j$	$x_j - \frac{j-1}{n}$
1	0.03226	0.00108	0.03226
2	0.06452	0.00215	0.03118
3	0.09677	0.00323	0.03011
4	0.12903	0.00430	0.02903
5	0.16129	0.00538	0.02796
6	0.19355	0.00645	0.02688
7	0.22581	0.00753	0.02581
8	0.25806	0.00860	0.02473
..	..	..	..
29	0.93548	0.03118	0.00215
30	0.96774	0.03226	0.00108
	Max	0.03226	0.03226

$$K^- = \sqrt{n}\max_j\left(x_j - \frac{j-1}{n}\right) = \sqrt{30} * 0.03226 = 0.1767$$

$$K^+ = \sqrt{n}\max_j\left(\frac{j}{n} - x_j\right) = \sqrt{30} * 0.03226 = 0.1767$$

Кузатилаётган K^+ ва K^- қийматлар мос ҳолда 0.1767 га тенг бўлди. Бу эса кутилган 0.22 қийматидан кичик. Демак кетма-кетлик ушбу ҳолатдан мувофақиятли ўтди.

5 - жадвал

Колмогорф-Смирнов тестининг критик нуқталари жадвали

Сони	**α ўзгарувчининг қийматлари**				
N	**0.20**	**0.15**	**0.10**	**0.05**	**0.01**
1	0.900	0.925	0.950	0.975	0.995
2	0.684	0.726	0.776	0.842	0.929
3	0.565	0.597	0.642	0.708	0.828
4	0.494	0.575	0.564	0.624	0.733
5	0.446	0.424	0.510	0.454	0.669
6	0.410	0.436	0.470	0.521	0.618

7	0.381	0.405	0.438	0.486	0.577
8	0.358	0.381	0.411	0.457	0.543
9	0.339	0.360	0.388	0.432	0.514
10	0.322	0.342	0.368	0.410	0.490
11	0.307	0.326	0.452	0.391	0.468
12	0.295	0.313	0.338	0.375	0.405
13	0.284	0.302	0.325	0.361	0.433
14	0.274	0.292	0.314	0.349	0.418
15	0.266	0.293	0.304	0.338	0.404
16	0.258	0.274	0.295	0.328	0.392
17	0.250	0.266	0.286	0.318	0.381
18	0.244	0.259	0.278	0.309	0.371
19	0.237	0.252	0.272	0.301	0.363
20	0.231	0.246	0.264	0.294	0.365
25	0.21	0.22	0.24	0.27	0.32
30	0.19	0.20	0.22	0.24	0.29
35	0.18	0.19	0.21	0.23	0.27
>35	$\frac{1.07}{\sqrt{N}}$	$\frac{1.14}{\sqrt{N}}$	$\frac{1.22}{\sqrt{N}}$	$\frac{1.36}{\sqrt{N}}$	$\frac{1.63}{\sqrt{N}}$

NIST Special Publication 800-22 нашрида статистик тестлар тўплами ва улурнинг дастурий таъминоти. Ушбу бўлимда National Institute of Standards and Technology (NIST) институти томонидан мақул деб топилган статистик тестлар ҳақида маълумот берилган. Ушбу статистик тестларни тасодифий ва псевдотасодифий битлар кетма-кетлигини тасодифийликка текшириш учун фойдаланилади [15].

Chastota (Frequency,Monobit) тести. Ушбу тест умумий кетма-кетликдаги ноллар ва бирлар миқдорини кўрсатади. Ушбу тестнинг мақсади берилган кетма-кетмакетликдаги ноллар ва бирлар миқдорини ҳақиқий тасодиф бўлган кетма-кетликдаги бирлар ва ноллар миқдорига қанчалик яқинлигини ҳисоблайди. Ушбу тест кетма-кетликдаги бирлар миқдори умумий узунлик миқдорини ½ улушуни, яни бирлар ва ноллар

миқдори бир хил миқдорда бўлишини талаб етади. Шу еслатиб ўтиш керакки барча статистик тестлар ушбу тестдан ўтишга боғлиқ бўлиб, агар ушбу тестдан ўтмаса у тасдофий деб ҳисобланмайди. Ҳисоблашлар натижасида олинган *P* параметр қиймати *0.01* дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади. Ушбу тестда кирувчи кетма-кетлик узунлиги камида *100* битни ташкил этиши шарт [15].

Блоклар учун частота (Frequency Test within a Block) тести. Ушбу тест *M-бит* узунликдаги блокда бирлар миқдорини кўрсатади. Ушбу тестнинг мақсади еса *M-бит* узунликдаги бирлар миқдори *M/2* га, яъни кутилган тасодифийлик даражасига қанчалик яқинлигини текширади. Агар блок узунлиги *M=1* бўлса бу тест биринчи, частотали тест натижасини билан бир хил натижага эга бўлади. Ҳисоблашлар натижасида олинган *P* параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади. Ушбу тестда кирувчи кетма-кетлик узунлиги камида 100 битни ташкил этиши шарт. Ҳар бир блок узунлиги эса умумий узунликни 10% дан кам бўлмаслиги керак [15].

Югиришлар (Runs) тести. Ушбу тест умумий кетма-кетликдаги бир хил битли бўлинмаган кетма-кетликлар оралиғини умумий сонини кўрсатади. К оралиқ узунлиги бир хил бўлган К битлардан иборатдир ва у боши ва охиридан қарама-қарши битли ўзгарувчилар билан чегараланган. Оралиқлар тестининг мақсади турли хил узунликдаги кетма-кетликларнинг бирлар ва ноллар оралиқлари сонини кутилган тасодифий кетма-кетликдаги кўрсаткичларга ўхшашлигини ҳисоблашдир. Хусусан, бу тест бирлар ва ноллар орасидаги кўчиш қанчалик тез ёки секинлигини ҳисоблайди. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади.

Қолган ҳолларда эса тасодифий эмас деб топилади. Ушбу тестда кирувчи кетма-кетлик узунлиги камида 100 битни ташкил этиши шарт [15].

Блок ичидаги енг узун югуришни ҳисоблашга асосланган (Longest Run of Ones in a Block) тест. Ушбу тест М-бит узунликдаги блок ичидага кема-кет келган бирларни узунлигини ҳисоблашга асосланган. Ушбу тестнинг мақсади, ҳар бир блокдаги кетма-кет келган бирлар узунлигини кутилган кетма-кетликдаги миқдорига қанчалик яқинлигини ҳисоблашдир. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади. Ушбу тестда кирувчи кетма-кетлик узунлиги камида 128 битни ташкил этиши шарт [15].

Бирлик матрица рангини ҳисоблашга асосланган (Binary Matrix Rank) тест. Ушбу тест берилган кетма-кетликдан тузилган бир-бирига боғлиқ бўлмаган матрицаларни рангини ҳисоблашга асосланган. Ушбу тест орқали тестлашдан асосий мақсад, берилган кетма-кетликдан ҳосил қилинган белгиланган узунликка эга бўлган қисмларнинг бир-бирига чизиқли боғлиқлигини текширдир. Ушбу тест шунингдек DIEHARD тестида ҳам қўлланилган. Ушбу тесда кирувчи параметр ҳисобланган M=Q=32 га тенг бўлиши шарт, яни матрицанинг ўлчами 32x32 бўлиши шарт ва матрицалар сони камида 38 тани ташкил этиши шарт. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади [15].

Дискрет фурье алмаштиришлари спектрал (Discrete Fourier Transform, Spectral) тести. Ушбу тест кетма-кетликнинг дискрет фурье алмаштиришларининг энг юқори кўрсаткичини кўрсатади. Ушбу тестнинг мақсади текшириладиган кема-кетликдаги даврийлик хусусиятини текширишдир, ёки бошқача айтганда тасодифийликдан чекиниш даражасини кўрсатишдир, яни юқори миқдорлар сони 95% дан юқори, паст

миқдорлар эса 5% дан фарқли еканлигини кўрсатишдир. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади. Ушбу тестда кирувчи кетма-кетлик узунлиги камида 1000 битни ташкил этиши шарт [15].

Даврий бўлмаган қисмлар бўйича (Non-overlapping Template Matching) тест. Ушбу тест текшириш учун олинган бирламчи белгиларни кўринишлар сонини кўрсатади. Ушбу тестнинг мақсади генераторларнинг ишлаб чиқарадиган кетма-кетликларида даврий бўлмаган қисмларини топишга асосланган. Ушбу тестлашда текшириладиган тасодифий кетма-кетлик m-бит бўлаклари устида амаллар бажарилади. Агар даврий бўлмаган қисм топилмаса m бир битга силжитилиб, кейинги битлар кетма-кетлиги олинади. Агар бундай ёқотиш топилса, даврий бўлмаган қисмдан кейинги бит тозаланиб, тестлаш давом эттирилади. Тесташда m бўлак узунлиги 9 ёки 10 битга тенг бўлиши ва кирувчи кетма-кетлик узунлиги эса 10^6 бит бўлиши талаб этилади. Бўлаклар сони эса M>0.01n ни ташкил этиши шарт. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади [15].

Даврий бўлган қисмлар бўйича (Overlapping Template Matching) тест. Ушбу тест текшириш учун олинган бирламчи белгиларни кўринишлар сонини кўрсатади. Ушбу тестда текширилаётган кетма-кетлик M бўлакка бўлиниб, ҳар бир бўлак устида m бит қисми олиниб, улар устида текширишлар олиб борилади. Агар m бит даврий бўлса, бир битга сурилиб амаллар давом эттирилади. Ушбу тест ва даврий бўлмаган қисмлар бўйича тестлашнинг фарқи агар даврий бўлмаган қисм топилса, текширилаётган бўлак бир бит олдинга сурилиб давом эттирилади. Тесташда m бўлак узунлиги 9 ёки 10 битга тенг бўлиши ва кирувчи кетма-кетлик узунлиги эса 10^6 бит бўлиши талаб этилади. Бўлаклар сони эса

M>0.01n ни ташкил этиши шарт. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади [15].

Маурернинг "Универсал статистик" (Maurer's "Universal Statistical") тести. Ушбу тестнинг мақсади берилган кетма-кетликни кам ёқотилиш билан қисилишини текширади. Агар кетма-кетлик кўп ёқотилиш билан қисилса, ундай кетма-кетлик тасодифий деб топилмайди. Ушбу тест катта узунликдаги кетма-кетликни талаб этади($n \geq (Q+K)L$) ва L бит блокдан қилиниб икки қисмга бўлинади. Бу ерда L қуйидаги ораликда бўлиши керак: $6 \leq L \leq 16$. Биринчи қисм Q та блокдан иборат бўлиб, Q қуйидагича танланиши шарт: $Q=10*2^L$. Кейинги қисм эса К($K=[n/L]-Q \approx 1000*2^L$). L, Q ва n ўзгарувчилар 6-жадвал асосида танланади.

Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади [15].

6 - жадвал

Маурернинг универсал статистик тестида қийматларни олиш жадвали

n	**L**	**$Q = 10 \cdot 2^L$**
≥387,840	6	640
≥904,960	7	1280
≥2,068,480	8	2560
≥4,654,080	9	5120
≥1,342,400	10	10240
≥22,753,280	11	20480
≥49,643,520	12	40960
≥107,560,960	13	81920
≥231,669,760	14	163840
≥496,435,200	15	327680
≥1,059,061,760	16	655360

Lempel-Ziv сиқиш тести. Ушбу тестнинг мақсади кетма-кетликни қанчалик сиқилишини кўрсатади. Агар кетма-кетлик кўп сиқилса, тасодифий ҳисобланмайди. Ушбу тестда кирувчи кетма-кетлик узунлиги камида 10^6 бўлиши талаб этилади. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади [15].

Чизиқли мураккаблик(Linear Complexity) тести. Ушбу тест чизиқли сурулувчи регистер(linear feedback shift registor) узунлигини кўрсатади. Ушбу тестнинг мақсади текширилувчи кетма-кетликни натижавий тасодифийликка тўлиқ этарлилигини текширади. Тасодифий кетма-кетлик чизиқли сурулувчи регистер узунлиги билан характерланади. Чизиқли сурулувчи регистер қанчалик қисқа бўлса, у шунчалик тасодифий емас деб ҳисобланади. Ушбу тестда кирувчи кетма-кетлик узунлиги камида 10^6 га тенг бўлиши шарт. Ушбу тестдан ижобий натижа олиши учун, M ва N қуйидагича танланиши шарт: $500 \le M \le 5000$ ва $N \ge 200$. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади [15].

Давомийлик(Serial)тести. Ушбу тест берилган кетма-кетликда *m*-бит қисмни пайдо бўлиш частотасини кўрсатади. Ушбу тестнинг мақсади *m*-битли текширишда 2^m пайдо бўлиш даражасини тасодифий кетма-кетликдаги кўрсаткичига нисбатан таққослашдир. Тасодифий кетма-кетликда ўхшашлик бор, яъни ҳар бир *m*-битда бир хил даражада пайдо бўлиш кўрсаткичи бор. Шуни эслатиб ўтиш керакки, агар $m=1$ бўлса бу тест натижаси частоталар тести билан бир хил натижага эга бўлади. Ушбу тестда кирувчи параметр ҳисобланган *m* ва *n* ўртасидаги боғлиқлик қуйидагича бўлиши керак: $m<[\log_2 n]-2$. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик

тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади [15].

Тахминий энтропия (Approximate Entropy) тести. Ушбу тест берилган кетма-кетликда *m*-бит қисмни пайдо бўлиш частотасини кўрсатади. Бу тест икки қисм(m ва m+1узунликка эга бўлган)нинг пайдо бўлиш частоталарини тасодифий деб олинган кетма-кетлик натижаси билан таққослаш мақсадида ишлатилади. Бу тестда кирувчи *m* ва *n* параметрлар $m<[\log_2 n]-2$ кўринишида танланиши шарт. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади [15].

Ортиб борувчи йиғинди (Cumulative Sums) тести. Ушбу тест берилган кетма-кетликни максимал йиғиндисини (+1 ва -1 рақамлари тарзида олиб) кўрсатади. Ушбу йиғинди қанчалик катта ёки қанчалик кичик бўлиши билан ушбу кетма-кетликни тасодифийлик даражаси белгиланади. Тасодифий кетма-кетликда ушбу йиғинди нол қийматига яқинроқ бўлади ёки бошқача айтганда йиғинда қанчалик нолга яқин бўлса кетма-кетликнинг тасодифийлик даражаси шунча юқори бўлади. Ушбу тестда кирувчи кетма-кетлик узунлиги камида 10^2 бўлиши талаб этилади. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади [15].

Тасодифий ташрифлар (Random Excursions) тести. Ушбу тест берилган кетма-кетлик йиғиндиси ичида мавжуд *K* ҳодисани такрорланиш сонини кўрсатади. Ушбу йиғинди 0 ни -1 га ва 1 ни +1 га алмаштириш орқали жамланади. Тасодифий кетма-кетлик нолга тенг бўлган соҳасида қисмларга ажратилади ва ҳар бир қисм учун тестлаш амалга оширилади. Ушбу тестнинг мақсади тасодифий кетма-кетликдаги ҳолат берилган кетма-кетликнинг қайси қисмида учрашини текширади. Ушбу тест ҳар бир

оралиқ саккизта ҳолатга бўлиниб текширилади. Булар: -4, -3, -2, -1 ва +1, +2, +3, +4. Ушбу тестда кирувчи кетма-кетлик узунлиги камида 10^6 бўлиши талаб этилади. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади [15].

Тасодифий ташрифлар варианти (Random Excursions Variant) тести. Ушбу тест тасодифий кетма-кетликда аниқ бир ҳолатнинг такрорланиш сонини кўрсатади. Тасодифий ташрифлар варианти тестининг мақсади тасодифий кетма-кетликда ҳар хил ҳолатларни пайдо бўлиш сонини бир хилда емаслигини топиш ҳисобланади. Бу тест 18 кетма-кет тестлар тўпламидир, битта тест ва ҳар бир ҳолат(-9, -8, …, -1 ва +1, +2, …, +9.) учун хулосалардан иборат бўлади. Ушбу тестда кирувчи кетма-кетлик узунлиги камида 10^6 бўлиши талаб этилади. Ҳисоблашлар натижасида олинган P параметр қиймати 0.01 дан катта бўлса, берилган кетма-кетлик тасодифий деб ҳисобланади. Қолган ҳолларда эса тасодифий эмас деб топилади [15].

Тестлашни ташкил қилиш ва натижаларни олиш. Ушбу жараён учта босқичдан иборатдир [15]:

- ПТСКК генераторини статистик тестлашни ташкил қилиш;
- NIST статистик тестларни эмперик натижаларини аниқлаш;
- тестни ўтказишда керакли бўлган тавсиялар.

Статистик тестларни ўтказишни бир неча усуллари мавжуд. NIST да статистик тестлаш беш босқичда амалга оширилади.

1-қадам. Генераторни танлаш. Бу босқичда дастурий ёки аппарат генератор орқали иккилик кетма-кетлик ҳосил қилинади(0 ва 1 дан иборат n бит узунликдаги).

2-қадам. Ҳосил қилинган кетма-кетликдан m бит узунликдаги кетма-кетликлар ҳосил қилинади ва файлга сақланади.

S_1={0,1,1,1,0,1,0,0,1..1}

S_2={0,1,1,1,0,1,1,0,1..0}

S_3={0,1,1,1,0,1,1,0,1..1}

S_4={0,1,1,1,0,1,0,0,0..1}

……

S_m={0,1,1,1,0,1,0,0,0..0}

3-қадам. Бу қадамда статистик тестларга кирувчи параметрлар танланади(кириш бит узунлиги, блок узунлиги ва ҳ.к).

	$Test_1$	**$Test_2$**	**….**	**$Test_{16}$**
S_1	$P_{1,1}$	$P_{1,2}$	….	$P_{1,16}$
S_2	$P_{2,1}$	$P_{2,2}$	….	$P_{2,16}$
.	.	.	….	
.	.	.	….	
S_m	$P_{m,1}$	$P_{m,2}$	….	$P_{m,16}$

4-қадам. P - параметрни ҳисоблаш. Бу босқичда ҳар бир тестдан ҳисоблаш орқали *p* – қиймат топилади.

$P_{1,1}$=0.0572	$P_{1,2}$=0.0392	…….	$P_{1,16}$=0.8532
.	.	…….	.
.	.	…….	.
$P_{m,1}$=1.000	$P_{m,2}$=0.4634	…….	$P_{m,16}$=0.9999

5-қадам. Олинган натижалар орқали кетма-кетликни тестлашдан ўтган/ўтмаганлиги аниқланади. Агар P-қиймат [0,1] оралиқда бўлса тестлашдан ўтган бўлади.

PASS={$P_{1.1}$,$P_{1.2}$,$P_{1.3}$,….} FAIL={$P_{1.10}$, $P_{2,4}$,$P_{6.8}$….. }

Статистик тестлар натижалари тестлашдан ўтган кетма-кетликлар сони орқали эмперик қийматлари топилади. Масалан, агар биз олган 1000 та кетма-кетлик қаторлари орасидан (*m=1000*), *α=0.01* еркинлик даражаси билан 996 таси ўтган бўлсин. Яъни, 996 тасини қиймати *P≥0.01* шартни қаноатлантиради, у ҳолда нисбат 996/1000=0.9960.

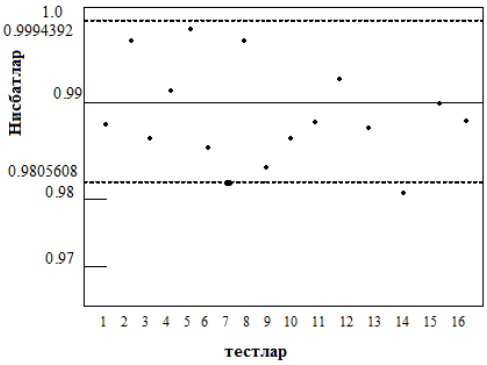
Статистик тестлардан олинган натижаларни тесташдан ўтиш ораликда тегишли бўлганларини аниқлаймиз. Бу оралиқ қуйидаги ифода билан аниқланади.

$\hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$ бу ерда, $\hat{p} = 1 - \alpha$

Ушбу ораликда биз олган ҳол учун *0.99±0.0094392* га тенг бўлади ва тестлар натижалари 0.9805607 дан катта ва 0.9994392 дан кичик бўлса, ушбу кетма-кетликлар тасодифий деб топилади. P қиймат график тарзда қуйидагича тасвирланади (10- расм).

Ушбу натижаларни олишда кирувчи *n* қиймат узунлиги (кирувчи кетма-кетлик узунлиги) миллиондан катта қилиб олиниши шарт.

Ушбу графикдан кўриниб турибдики агар тестлаш натижалари кўрсатилган ораликда тегишли бўлса, кетма-кетлик тасодифий деб тан олинади.



10- расм. P қийматни график тарзда ифодаланиши

2.2. Тасодифийликка текширувчи тестларнинг қиёсий таҳлили

Юқорида тасодифий ва псевдотасодифий кетма-кетликларни тасодифийликка текширувчи тестлар кўриб ўтилди. ПТСКК танлашда тасодифийликка текширувчи тестлар муҳим аҳамият касб этади. Шунга

кўра ПТСКК қанчалик кўп тасодифийликка текширувчи тестлардан ўтса, генератор шунча криптобардошли деб топилади.

Қуйидаги жадвалда ушбу тасодифийликка текширувчи тестларнинг қиёсий таҳлили келтирилган.

7-жадвал

Тасодифийликка теширувчи тестларнинг қиёсий таҳлили

Тест/ Хусусиятлар	**Кирувчи кетма-кетлик узунлиги**	**Асосланиши**	**Мақсади**	**Ҳисоблаш даражаси**
Monobit test	100 бит камида	Ноллар ва бирлар миқдорини ҳисоблашга	Ноллар ва бирлар орасидаги фарқни ҳисоблаш	Тахминий
Block Frequency Test	100 бит камида	М бит узунликда бирлар сонини топишга	Олинган миқдорни кутилган миқдорга тенглигини (м/2) ҳисоблайди	Тахминий, блок узунлигига боғлиқ
Cumulative Sums (Cusum) Test	100 бит камида	Бирлар ва ноллар миқдорини ҳисоблашга	Бир ва ноллар миқдорини тенглигини ҳисоблаш	Тахминий
Runs Test	100 бит камида	Бирлар ва ноллар миқдорини ҳисоблашга	Бирлар ва ноллар ораликлари сонини кутилган миқдорга тенглигини	Тахминий
Long Runs of Ones Test	128 бит камида	Кетма-кет келган бирлар сонини ҳисоблашга	Ҳар бир қисмдаги бирлар миқдорини кутилганига қанчалик яқинлигини	Тахминий
Rank Test	Камида 38 та 32х32 матрица	Матрица рангини ҳисоблашга асосланган	Ҳар бир қисмни бир-бирига боғлиқлик даражасини	Тахминий
Discrete Fourier Transform	1000 бит камида	Дискрет фуре алмаштириш	Даврийлик хусусиятини текшириш	Тахминий

(Spectral) Test		Ларига		
Non-overlapping Template Matchings Test	1 мил. бит камида	Киритилган кетма-кетликни ҳосил бўлиш сонини ҳисоблаш	Танланган кетма-кетликни кўринмаслик миқдорини	Тахминий, текширилувчи қисм битга боғлиқ
Overlapping Template Matchings Test	1 мил. бит камида	Киритилган кетма-кетликни ҳосил бўлиш сонини ҳисоблаш	Танланган кетма-кетликни кўринишлар миқдорини	Тахминий, текширилувчи қисм битга боғлиқ
Universal Statistical Test	Катта узунликдаги	Бир хил қисмларни топишга асосланган	Кетма-кетликни қисилиш узунлигини кўрсатади	Тахминий
Approximate Entropy Test	Катта узунликда	Қисмларни пайдо бўлиш частотасини	икки қисм(m ва m+1узунликка эга бўлган)нинг пайдо бўлиш частоталарини	Тахминий
Serial Test	Катта узунликда	Қисмларни пайдо бўлиш частотасини	m-битли текширишда 2^m пайдо бўлиш даражасини топиш	Тахминий
Random Excursions Test	1 мил. Бит	Ораликда кутилган ҳодисани кўринишлар сонини ҳисоболаш	Қисм кетма-кетликларни тасодифийликка текшириш	Аниқ
Random Excursions Varian Test	1 мил. бит	Ораликда кутилган ҳодисани кўринишлар сонини ҳисоболаш	Қисм кетма-кетликларни тасодифийликка текшириш	Аниқ
Lempel-Ziv Test	1 мил. бит камида	Ўхшаш қисмларни топишга	Кетма-кетликни қисилишлар миқдорини кўрсатади	Тахминий

Linear Complexity Test	1 мил. бит камида	Кетма-кетлик оралиғини ҳисоблаш	Ўхшаш қисмлар орасидаги узунликни ҳисоблаш	Тахминий, блок узунлигига боғлиқ
Хи-квадрат	Катта узунликдаги ПТСКК	Олинган ва кутилган натижалар фарқини ҳисоблашга	ПТСКК бирлар ва ноллар миқдори улушини ҳисоблаш	Тахминий, бардошсиз калитга боғлиқ
Кол-Смирнов	Кичик узунликдаги ПТСКК	Олинган ва кутилган натижалар фарқини ҳисоблашга	Кичик ва катта оғишларни ҳисоблаш	Аниқ

Ушбу тасодифийликка текшириш тестлари бир-биридан кириш параметрлари ва текшириш усулига кўра фарқ қилади. Бу ўринда шуни ҳисобга олиш керакки барча тестларнинг асоси ва тестлашнинг бирламчи шарти сифатида Хи-квадрат тести олинади.

ПТСКК ни тасодифийликка текширишда ушбу тестлар тўпламидан фойдаланишда шуни эътиборга олиш керакки агар псевдотасодифий кетма-кетлик қанчалик кўп теслашдан ўтса унинг тасодифийлиги шунча юқори ҳисобланади.

II боб бўйича хулосалар

Магистрлик диссертация ишининг иккинчи бўлими ПТСКК ни тестлашга бағишланган бўлиб, дастлаб псевдотасодифий кетма-кетлик блоклари белгиларини тасодифийликка текширувчи тестлар ҳақида фикр ва мулоҳазалар билдирилиб, уларнинг график ва баҳолаш тестларидан иборатлиги изоҳланди

Ҳозирда кенг қўлланиладиган тасодифийликка текширувчи тестлар бўлмиш, “Хи-квадрат” ва “Калмагорф-Смирнов” критерийсининг амалий қўлланилиш усули кўрсатилди.

Бундан ташқари NIST ташкилотининг тасодифийликка текширувчи тестлар тўплами билан танишилди. Унинг ишлаш усули ва ундан натижани олиш ва таҳлил қилиш кўриб ўтилди. Ушбу тестлаш комплекси 16 та тестлардан ташкил топган бўлиб, уларнинг ҳар бири ПТСКК ни алоҳида-алоҳида хусисиятлар бўйича текширади.

Келтирилган тестларнинг ишлаш принципи, кириш кетма-кетлик узунлиги ва ҳисоблашнинг аниқлиги бўйича таҳлил қилинди.

Якуринй хулоса сифатида шуни айтиш мумкинки, ушбу тестлаш усуллари ПТСКК тестлашда муҳим аҳамият касб этиб, улардан фойдаланиш сифати, кирувчи кетма-кетлик узунлиги ва сонига боғлиқ бўлади.

III боб. Симметрик криптотизимлар учун ишлаб чиқилган калит генерациялаш алгоритмларининг дастурий таъминотлари

3.1. Ўзгартирилган RC4 калит генератори алгоритмининг дастурий таъминоти

Биринчи бўлимда RC4 узлуксиз шифрлаш алгоритми учун калит ишлаб чиқарувчи генератор билан танишилди. Ундаги мавжуд камчиликлар топилиб улар бартараф этилиб янги генератор алгоритми келтирилди. Қуйида алгоритмнинг ўзгартиришлар киритилган қисмлари ва алгоритми келтирилган.

Унга кўра

- кирувчи параметрларни ҳосил қилишда тасодифий параметрлардан фойдаланилган;
- калитни ҳосил қилиш қисмида ўзгартиришлар киритилган.

Қуйида калитни генерация қилиш келтирилган:

```
while GeneratingOutput:
    i := i + 1
    a := S[i]
    j := j + a
    b := S[j]
    S[i] := b      (Swap S[i] and S[j])
    S[j] := a
    c := S[i<<5 ⊕ j>>3] + S[j<<5 ⊕ i>>3]
    output (S[a+b] + S[c⊕0xAA]) ⊕ S[j+b]
endwhile
```

Бундан кўриниб турибдики янги киритилган *c* параметр орқали чиқувчи калит битлари ўзгартириляпти. Дастлабки калитни шакллантириш жараёни эса қуйидаги кўринишга эга:

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

Бу жараёнда дастлаб *S* массив элементлари ҳосил қилиниб, сўнгра улар массив элементларини алмаштирувчи *swap* функцияси ёрдамида

массив элементлари ўрни ўзгартирилди. Ўзгартириш функцияси эса қуйидаги кўринишга эга:

```
byte temp = array[ind1];
array[ind1] = array[ind2];
array[ind2] = temp;
```

Ушбу яратилган калит генераторлари учун тасодифий кирувчи параметрларни ҳосил қилишда ва уларни йиғишда *source* классдан фойдалилди. Ушбу класс ҳар бири алоҳида тасодифий параметрларни ташкил этувчи қуйидаги функциялардан иборат.

machine_name(). Ушбу функция фойдалинаётган ШК номини қайтаради.

getOSInfo(). Ушбу функция операцион тизим ҳақидаги маълумотларни қайтаради.

memory_info(). Ушбу функция тизим хотираси ҳақидаги умумий маълумотларни беради.

mouse_position(). Сичқонча қурилмасининг ҳолатини қайтарувчи функция.

process_info(). Ушбу функция тизим жараёнлари ҳақидаги маълумотларни қайтаради.

HiPerfTimer. Бу юқори аниқлик билан вақтни аниқловчи класс бўлиб, у жараённинг бажарилиш вақтини кўрсатади.

highperformer_timer(string name). Ушбу функция кирувчи параметрга кўра бажарилувчи жараённи аниқлайди. Бунда кирувчи параметр тасодифий равишда танланади.

get(). Ушбу функция класс ичида бажарилувчи барча функцияларни бирлаштириб якуний натижани ҳисоблайди.

Кирувчи параметрнинг тасодифийлик даражасини ошириш мақсадида ҳар бир кирувчи параметрлар тўғридан-тўғри эмас балки хэш қиймати олиниб, якуний қиймат ҳам хэшланади. Олинган натижа эса калит генераторлари учун киручви тасодифий катталикни беради.

3.2. Ўзгартирилган ANSI X9.17 калит генератори алгоритмининг дастурий таъминоти

ANSI X9.17 калит генерациялаш стандарти бир томонлама функцияга асосланган алгоритм ҳисобланиб, унда бир томонли функциялар сифатида учлик DES шифрлаш алгоритми фойдаланилган. Ушбу стандарт алгоритмида мавжуд камчиликлар эса биринчи бўлимда келтирилган эди. Ушбу камчиликларни бартараф этиш учун қуйидаги ишлар амалга оширилди:

- шифрлаш алгоритми AES стандартига алмаштирилди;
- шифрлаш алгоритмлари калитларини доимий янгилаш амалга оширилди;
- вақтни синхронлашдан фойдаланилди;
- кирувчи параметрларни ҳосил қилишда тасодифий параметрлардан фойдаланилди.

Кирувчи параметрлар тизим манбаси асосида ҳосил қилиниб, генераторнинг дастлабки калити сифатида фойдаланилди. Қуйида дастурнинг асосий функцияси келтирилган. Кўриниб турубдики, шифрлашда ишлатиладиган калитлар ҳаб бир калитни ҳосил қилиш даврида янгиланади. Яна шуни этиборга олиш керакки учлик DES да фақат иккита калитдан фойдаланилган бўлса, бу алгоритмда шифрлашнинг ҳар бир жараёнида ҳар ҳил калитлардан фойдаланилди.

3.3. Ўзгартирилган FIPS-186 калит генератори алгоритмининг дастурий таъминоти

FIPS-186 стандарти учун калит ҳосил қилувчи генератор алгоритми ва ундаги камчиликлар биринчи бўлимда келтирилди. Ундаги мавжуд камчиликларни бартараф этиш мақсадида қуйидаги ўзгаришлар амалга оширилди:

- шифрлаш алгоритми, хэшлаш алгоритмларини бардошлисига алмаштирилди;
- y_i ҳисоблашда кирувчи параметрларни тўғридан-тўғри емас балки хэшлаб киритилди;
- X_{i+1} ни W_i га боғлиқлигини таъминланди, яъни $X_{i+1} = X_i + hash(output_i + W_i)$.
- Кирувчи параметрларни ҳосил қилишда тасодифий параметрлардан фойдаланилди.

Ушбу юқорида белгиланган ўзгаришлар қилиниб, бардошли шифрлаш алгоритми сифатида ГОСТ 28147-89 дан фойдаланилди.

```
private string gost_shifrlash(string kalit, string text)
        {
            byte[] plain_text = new Byte[512];
            System.Text.ASCIIEncoding enc = new
System.Text.ASCIIEncoding();
            byte[] strbytes = enc.GetBytes(text);
            Array.Copy(strbytes, 0, plain_text, 0, strbytes.Length);
            byte[] key = new Byte[32];
            Random rand = new Random();
            for (int i = 0; i < key.Length; i++)
            {
                int t = rand.Next(0,16);
                key[i] = Convert.ToByte(kalit[t]);
            }
            byte[] iv = new Byte[8] { 0, 1, 2, 3, 4, 5, 6, 7 };
            GOSTsymmManaged gost = new GOSTsymmManaged();
            gost.LoadTestSBoxes(2);
            gost.Key = key;
            gost.IV = iv;
            gost.Mode = CipherMode.CBC;
            GOSTsymmTransform ct_e =
(GOSTsymmTransform)gost.CreateEncryptor();
            ct_e.UseExpandedSBoxes = false;
            MemoryStream ms1 = new MemoryStream();
            CryptoStream cs1 = new CryptoStream(ms1, ct_e,
CryptoStreamMode.Write);
            cs1.Write(plain_text, 0, plain_text.Length);
            cs1.Close();
            byte[] cipher_text = ms1.ToArray();
            string natija = PrintByteArray(cipher_text);
            return sha256_xeshing(natija);
          }
```

Шифрлашда фойдаланилган ҳар бир калит манбадан йиғилган тасодифий параметрлар асосида ҳосил қилинади.

3.4. Параметрли алгебра асосланган калит генератори алгоритмининг дастурий таъминоти

Ушбу генератор комбинациялашган генератор саналиб, дастлабки калитлар O`z Dst 1105:2009 шифрлаш стандартида фойдаланилган сеанс калитини шакллантириш функциясидан фойдаланилди. Ушбу функция байтлар ўрнини алмаштиришга ва параметрли кўпайтириш, қўшиш, даражага ошириш амалларидан иборат бўлиб, калит шаклланиш босқичида натижа сифатида иккита 256 ўлчамли массив ҳосил қилинади. Ушбу босқичлар қуйидагича [8,10]:

1. Кирувчи икки параметр (калит) асосида k ва k_f лар шакллантириш.

2. Ушбу икки калит асосида $k_{se}=k+k'*(1+k_f*k)$ ҳисобланди ва чапдан *672* бит қолдиринсин, бунда k'- k_f нинг ўнгдан *192* битли қисми.

3. k_{se} да ўнгдан *256+64* битли қисм ажратиб олинсин, чапдан *256* битли қисмдан байтли элементлардан таркиб топган чизиқли массив $K_{st}=[0,1,2,3,\dots 31]$, қолган *64* битли қисмдан – байт сатҳида элементлардан таркиб топган чизиқли массив $B= [0,1,2,3,4,5,6,7]$ шакллантирилсин.

4. Чизиқли массив B элементларидан $B_1=[0,1,2,3]$ ва $B_2=[4,5,6,7]$ массивлар жуфти ажратиб олиниб, улардан қуйида келтирилган қоидалар асосида (d_1, R_1, L_1) ва (d_2, R_2, L_2) параметрлар учликлари шакллантирилсин:

4.1. $j=0$ учун агар $b[j] <3$ бўлса, $d_1=3$ қабул қилинсин, акс ҳолда $d_1= b[0]$ қабул қилинсин; а $j=4$ учун агар $b[j] <3$ бўлса, $d_2=3$ қабул қилинсин, акс ҳолда $d_2= b[4]$ қабул қилинсин.

4.2. $j=1$ учун агар, $b[j]=0$ бўлса, $R_1=1$ қабул қилинсин, акс ҳолда $R_1= b[1]$ қабул қилинсин; $j=5$ учун агар, $b[j]=0$ бўлса, $R_2=1$ қабул қилинсин, акс ҳолда $R_2= b[5]$ қабул қилинсин.

4.3. $j=2$ учун агар $b[j]=0$ бўлса, $L_1=1$ қабул қилинсин, акс ҳолда $L_1= b[2]$ қабул қилинсин; $j=6$ учун агар $b[j]=0$ бўлса, $L_2=1$ қабул қилинсин, акс ҳолда $L_2= b[6]$ қабул қилинсин.

4.4. $d_s\ (mod\ 2)=0$ учун агар $d_s\ (mod\ 4)=0$ бўлса, $d_s= d_s\ -1$ қабул қилинсин, акс ҳолда $d_s = d_s\ +1$ қабул қилинсин, бу ерда $s\in\{1,2\}$.

4.5. $d_s\ (mod\ 2)=1$ учун агар, $d_s\ -1\ (mod\ 4)=0$ бўлса, у ҳолда $d_s = d_s-2$ қабул қилинсин.

4.6. $j=3$ учун агар $b[j] =0$ бўлса, $b[j] =1$ қабул қилинсин; $j=7$ учун агар $b[j] =0$ бўлса, $b[j] =1$ қабул қилинсин.

5. Байт сатҳида, алмаштириб RC4 ва ISAAC+ генераторларига кирувчи параметр ҳосил қилиш учун, чизиқли массивлар жуфтлиги $(B_{1A}\ [256],\ B_{2A}\ [256])$ шакллантирилсин. Мазкур массивларни шакллантириш моҳияти қуйидаги кўрсатмаларда ўз аксини топган:

$s\in\{1,2\}$ учун ҳар бир адрес $i\in\{0,1,2,\dots255\}$ га мос $((i+L_s)\ mod256)+1$ қийматни d_s даражага параметр R_s билан 257 модули бўйича оширилсин, натижа 256 модули бўйича тақдим этилсин ва ҳар қадамда жорий натижа бир қадам аввалги натижа ҳамда i билан таққослансин. Агар таққосланган қийматлар тенг бўлса, ёки жорий натижа бир қадам аввалги натижага яқин ($|b_{sA}\ [i-1]-b_{sA}\ [i]\ | \geq 8$) бўлса, у ҳолда $s=1$ учун (i-$b[3]$) ёки $s=2$ учун (i-$b[7]$) га тенг адресдаги элемент билан жой алмаштирилсин.

Ҳисоблашлар алгоритми қуйидаги амалларни ўз ичига олади:

1. $i=0\ \div255$ учун $b_{sA}[i]\equiv(((i+L)\ mod256)+1\ )^{|\ ds}\ (mod\ 257)(mod\ 256)$ ҳисоблансин.

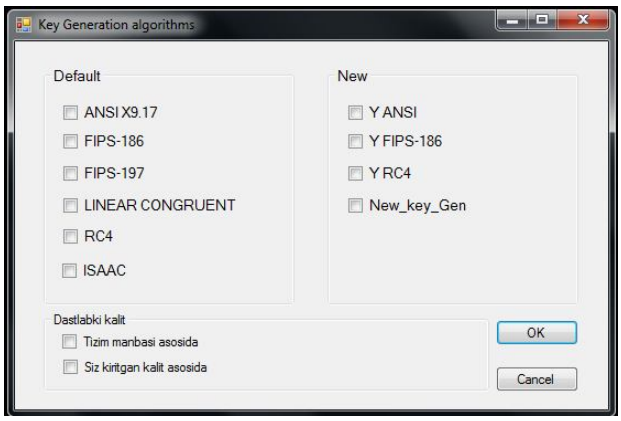
$i=1$ дан бошлаб ҳар қадамда i-$b_{sA}\ [i]\neq0$ ва $|b_{sA}\ [i-1]-b_{sA}\ [i]\ | \geq 8$ шартлари текширилсин ва агар иккала шарт қаноалантирилса, унда натижа қабул қилинсин, акс ҳолда $b_{sA}\ [i]$ $s=1$ учун (i-$b[3]$) $(mod\ 256)$ ёки $s=2$ учун (i-$b[7]$) $(mod\ 256)$га тенг манзилдаги элемент билан жойлари

алмаштирилсин ва *s=1* учун *b[3]= b[3]-5 (mod 256)* ёки *s=2* учун *b[7]= b[7]-5 (mod 256)* қабул қилинсин.

2. $b_{sA}[i,\ i=0,1,2,...255]$ элементларидан байт сатҳида алмаштириб шифрматн ҳосил қилиш, яъни *sh* режимида фойдаланиш учун чизиқли массивлар жуфтлиги $(B_{1A}[256], B_{2A}[256])$ шакллантирилсин.

Ушбу олинган массивлар, байтлар ўрнини алмаштиришга асосланган RC4 ва ISAAC+ калит генераторларига кириш параметри (массиви) сифатида киритилади ва ҳар бир генератордан олинган чиқиш қийматлари Макларен-Марсальи усули бўйича алмаштирилади ва натижавий 256 ўлчамли массив ҳосил қилинади.

Юқорида келтирилган генераторлар ва биринчи бўлимда келтирилган бардошли калит генераторлар “Key Generation algorithms” номли дастурда мужассамлашган ва ушбу дастурнинг умумий кўриниши қуйидагича:



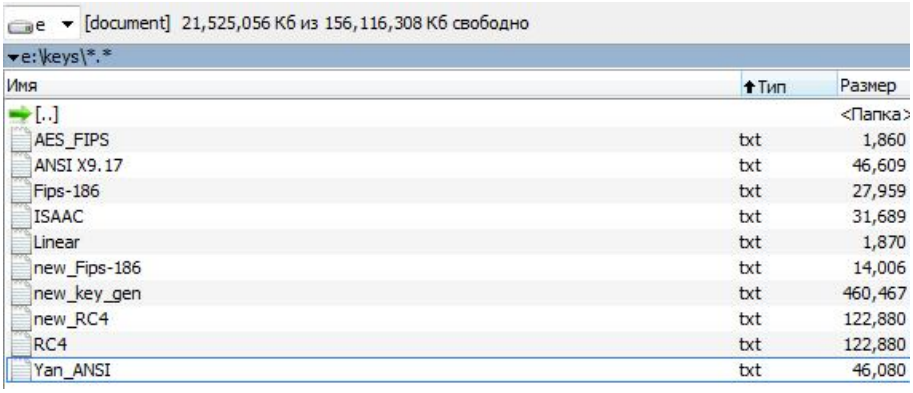
11-расм. Дастурнинг умумий кўриниши

Дастур ойнасининг чап томонида (default қисмида) кўрсатилган генераторлар ўнг қисмида эса юқорида келтирилган алгоритмларни оддий белгилаш орқали амалга оширишингиз мумкин. Дастурнинг қуйи қисмида дастлабки калитларни киритиш тартиби келтирилган, бунда сиз тизим манбасидан ёки фойдаланувчи киритган маълумот орқали ҳосил қилишингиз мумкин. Ҳосил қилинган дастур натижаси текст файлда

иккилик кўринишида сақланади. Қуйида тизим ресурслари асосида ҳосил қилиш кўрсатилган.

12-расм. Дастурдан фойдаланиш

Яратилган натижалар e:\keys\ манзилада иккилик кўринишда сақланади.



13-расм. Файлларнинг сақланиши ва манзили

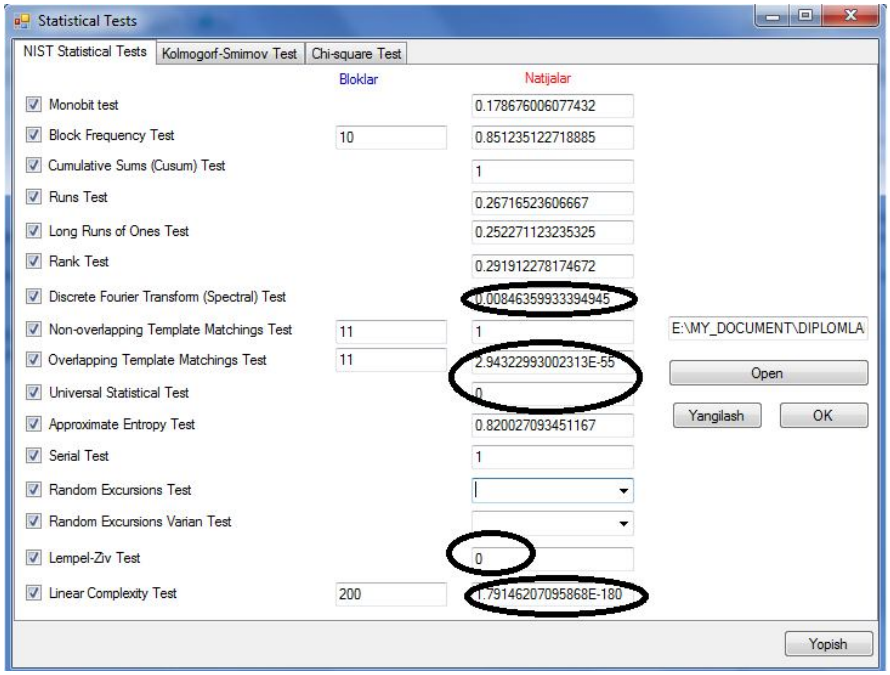
Ушбу файлларнинг ўлчами ҳар хиллиги улардаги калитлар сонига боғлиқ ва улардаги калит генератор алгоритмларига боғлиқ.

3.5. Яратилган генераторларнинг самарадорлигини баҳолаш

Ушбу бўлим барча олдинги бўлимларнинг натижаси сифатида қаралиб, бунда тасодифий калит генераторларидан олинган натижалар

сонларни тасодифийликка текширувчи тестлар ёрдамида тасодифийликка текширилади.

Тасодифийликка текширувчи тестлар мажмуи дастурининг умумий кўриниши қуйидагича:



14-расм. Дастурнинг умумий кўриниши

Ушбу дастур учта тасодифийликка текширувчи тестлар ойнасидан(иккинчи бўлимда айтиб ўтилган) иборат бўлиб, текширилиши керак бўлган файл танланади ва теслаш тури танланиб кутилган натижалар олинади.

Ушбу тестлашни ўтказишимиз учун калит генераторлари дастури орқали биз дастлаб тасодифий кетма-кетликларни ҳосил қилиб оламиз ва керакли манзилга сақлаймиз. Қуйидаги жадвалларда ПТСКК генераторлари ишлаб чиқарган натижаларни тасодифийликка текшириш натижалари келтирилган.

Ушбу тестлаш икки ўлчамда (1860 ва бир милионга яқин) амалга оширилади.

8-жадвал

1680 бит ўлчамдаги ПТСКК тестлаш натижаси

Тест/ генератор	AES_FIPS	ANSI X9.17	Fips-186	ISAAC	Linear	new_Fips-186	new_key_gen	new_RC4	RC4	Yan_ANSI
Monobit test	-	-	-	+	+	-	+	+	+	-
Block Frequency Test(10)	+	-	-	+	+	-	+	+	+	+
Cumulative Sums (Cusum) Test	+	-	+	+	+	+	+	+	+	+
Runs Test	-	-	+	+	+	+	+	+	+	-
Long Runs of Ones Test	-	-	-	+	+	-	+	+	+	-
Rank Test	+	+	+	+	+	+	+	+	+	+
Discrete Fourier Transform (Spectral) Test	+	+	-	-	-	+	+	+	+	+
Non-overlapping Template Matchings Test (11)	+	+	+	+	+	+	+	+	+	+
Overlapping Template Matchings Test (11)	-	-	-	-	-	-	-	-	-	-
Universal Statistical Test	-	-	-	-	-	-	-	-	-	-
Approximate Entropy Test	-	-	-	+	+	-	+	+	+	-
Serial Test	+	+	+	+	+	+	+	+	+	+
Random Excursions Test	+	+	+	+	+	+	+	+	+	+
Random Excursions Varian Test	+	+	+	+	+	+	+	+	+	+
Lempel-Ziv	-	-	-	-	-	-	-	-	-	-

Test										
Linear Complexity Test(50)	-	-	-	-	-	-	-	-	-	-
Хи-квадрат	-	-	-	+	+	-	+	+	+	-
Кол-Смирнов (0.01)	-	-	-	-	-	-	-	-	-	-
Жами	8	6	7	12	12	8	13	13	13	8

Кичик узунликда олинган натижалар ўрта узунликда қараганда юқорироқ эканлиги кўрилди. Ушбу тестлашда ҳам ўзгартириш асосида яратилган генератор натижалари юқорироқ еканлиги натижалар орқали исботланди. Бу ўринда шуни айтиб ўтиш керакки, тестлаш натижалари тестлаш сонига ва кирувчи параметрга боғликдир.

9-жадвал

1 мил. бит ўлчамдаги ПТСКК тестлаш натижаси

Тест/ генератор	**AES_FIPS**	**ANSI X9.17**	**Fips-186**	**ISAAC**	**Linear**	**new_Fips-186**	**new_key_gen**	**new_RC4**	**RC4**	**Yan_ANSI**
Monobit test	-	-	-	-	-	-	-	+	+	-
Block Frequency Test (100)	-	-	-	-	+	-	-	+	+	-
Cumulative Sums (Cusum) Test	+	-	+	+	+	+	+	+	+	+
Runs Test	+	-	-	-	+	-	+	+	+	-
Long Runs of Ones Test	-	-	-	+	-	-	-	+	+	-
Rank Test	-	-	-	-	-	-	-	-	-	-
Discrete Fourier Transform (Spectral) Test										
Non-overlapping Template Matchings Test (01100111010001)	-	-	-	-	-	-	-	-	-	-

Overlapping Template Matchings Test (01100111010001)	+	+	+	+	+	+	+	+	+	+
Universal Statistical Test	-	-	-	+	+	-	+	+	+	-
Approximate Entropy Test	-	-	-	-	-	-	-	+	+	-
Serial Test	+	+	+	+	+	+	+	+	+	+
Random Excursions Test	+	+	+	+	+	+	+	+	+	+
Random Excursions Varian Test	+	-	+	+	+	+	+	+	+	+
Lempel-Ziv Test	+	-	+	+	+	+	+	+	+	-
Linear Complexity Test(5000)	-	-	-	-	-	-	-	-	-	-
Хи-квадрат	+	-	-	+	+	-	+	+	+	-
Кол-Смирнов (0.01)	-	-	-	-	-	-	-	-	-	-
Жами	8	3	6	9	10	6	9	13	13	5

Юқоридаги жадвалдаги натижалардан келиб чиқиб, шуни айтиш мумкинки, ўзгартирилган ва янги ҳосил қилинган ПТСКК генераторлари бошқаларига қараганда бардошли саналиб, уларни симметрик шифрлаш тизимларида калитлар ҳосил қилишда фойдаланиш мумкин.

III боб бўйича хулоса

Ушбу магистрлик диссертациясининг якуний учинчи бўлимида мавжуд калит генераторлари асосида янги ўзгартирилган калит генераторлари ишлаб чиқилди.

Ушбу вазифани бажаришда биринчи бўлимдаги хулосалар асосида, мавжуд криптобардошли ҳисобланган калит генераторлари асос қилиниб, уларни заиф томонлари топилиб, уларни бартараф этиш орқали амалга оширилди.

Маълумки калит генераторлари учун кирувчи калит тасодифийлиги муҳим аҳамиятга ега. Шуни ҳисобга олган ҳолда тизим манбаси асосида тасодифий калитни ҳосил қилиш амалга оширилди.

Ҳозирда криптографияда янги санадган параметрли алгебра асосида дастлабки калитларни шакллантириш ва ушбу калитларни криптобардошли саналган RC4 ва ISAAC+ калит генераторларига кириш параметри қилиб бериш орқали натижавий комбинациялашган псевдотасодифий кетма-кетлик ҳосил қилувчи генератор яратилди.

Ҳар бир ҳосил қилинган кетма-кетликлар тасодифийликка текшириш тестлари орқали тестланди. Ушбу тестлаш натижаларига асосан янги яратилган калит генераторлари дастлабкиларига кўра криптобардошли тасодифий кетма-кетликларни ҳосил қилиши аниқланди.

ХУЛОСА

Криптографик алгоритмларни қўллаш асосида ахборот-коммуникация тармоғида алмашинувчи рақамли маълумотларни муҳофазасини ташкил қилиш бугунги кунда ахборот хавфсизлиги масалаларини ҳал қилишнинг асосий услубларидан биридир. Маълумотларни криптобардошли шифрлаш алгоритмлари орқали шифрлашда, тасодифий бўлган калитлар кетма-кетлигидан фойдаланиш самаралидир. Ушбу магистрлик диссертациясида олинган натижалар қуйидагилардан иборатдир:

1. Симметрик криптотизимлар учун ПТСКК яратилиш йўналишлари туркумлари, уларда қўлланилган математик амаллар ва уларга хавфсизлик юзасидан қўйилган талаблар асосида ПТСКК генераторлари туркумланди ва таҳлил қилинди.
2. Бир томонлама функциялардан, байтлар ўрнини алмаштиришга асосланган ПТСКК генераторлари фойдаланишга қулайлиги ва бардошлиги исботланиб, уларни дастурий модули ишлаб чиқилиб, фойдаланишга тавсия этилди.
3. Криптобардошли деб тавсия ътилган бир томонлама функциялар ва байтлар ўрнини алмаштиришга асосланган генераторлар таҳлил этилиб, заиф томонлари кўрсатилди. Ушбу камчиликларни бартараф этиш йўллари таклиф этилиб, улар асосида янги дастурий модуллар ишлаб чиқилди.
4. Криптографияда янги саналган, параметрли алгебра мураккаблигига асосланган ва комбинациялашган генератор асосида янги бардошли ПТСКК ишлаб чиқарувчи генератор ишлаб чиқилди.
5. ПТСКК тасодифийликка текшириш йўллари ўрганилиб, тестлашда қуйидаги тестлардан фойдаланиш мумкинлиги таклиф этилди:
   - Хи-квадрат усули;

- Колмогорф-Смирнов усули;
- NIST 800-22 тўпламига кирган тестлаш усуллари.

6. Яратилган ПТСКК ҳосил қилувчи генератор тасодифийликка текширувчи тестлар ёрдамида тестланиб, бардошлилиги аниқланда ва улар фойдаланиш учун таклиф этилди.

ФОЙДАЛАНИЛГАН АДАБИЁТЛАР РЎЙХАТИ

1. “Ахборотлаштириш тўғрисида”ги Ўзбекистон Республикаси қонуни, 11-декабр 2003-йил.
2. “Ўзбекистон Республикасида ахборотни криптографик муҳофаза қилишни ташкил этиш чора-тадбирлари” тўғрисида Ўзбекистон Республикаси президентининг қарори.
3. “Миллий Ахборот-коммуникация тизимларининг компютер хавфсизлигини таъминлаш борасидаги қўшимча чора-тадбирлар тўғрисида”ги Ўзбекистон Республикаси Президентининг қарори, 5 - сентабр 2005 – йил.
4. “Ахборотнинг криптографик ҳимоя воситаларини лойиҳалаштириш, тайёрлаш, ишлаб чиқариш, реализация қилиш, таъмирлаш ва улардан фойдаланиш фаолиятини лицензиялаш тўғрисидаги низомни тасдиқлаш ҳақида”ги Ўзбекистон Республикаси Вазирлар маҳкамасининг қарори, 21 - ноябр 2007 – йил.
5. Акбаров Д. Е. “Ахборот хавфсизлигини таъминлашнинг криптографик усуллари ва уларнинг қўлланилиши” – Тошкент, 2008 – 394 бет.
6. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. –М.: издательство ТРИУМФ, 2003 - 816 стр.
7. Практическая криптография. Нильс Фергюсон, Брюс Шнайер. 2005.
8. Хасанов Х.П. Такомиллашган диаматрицалар алгебралари ва параметрли алгебра асосида криптотизимлар яратиш усуллари ва алгоритмлари. –Тошкент, 2008. - 208 бет.
9. Узлуксиз шифрлашнинг криптобардошли алгоритмлари ва уларнинг самарадорлигини баҳолаш. Мусаев А.И. Номзодлик диссертацияси.

10. O‘z DSt 1105:2009. Государственный Стандарт Узбекистана. Информационная технология. Криптографическая защита информации. Алгоритм шифрования данных. Ташкент. Узбекское агентство стандартизации, метрологии и сертификации. 2009.
11. Практическая криптография. Нильс Фергюсон, Брюс Шнайер. 2005.
12. A Comparison of Four Pseudo Random Number Generators Implemented in Ada. William N. Graham. Applied Research Laboratories. The University of Texas at Austin. bill@titan.tsd.arlut.utexas.edu.
13. Pseudorandom Number Generators for Cryptographic Applications. Diplomarbeit zur Erlangung des Magistergrades an der Naturwissenschaftlichen Fakultat der Paris-Lodron-Universitat Salzburg. Andrea Rock. Salzburg, Marz 2005.
14. Testing Random-Number Generators. Raj Jain www.rajjain.com. ©2010.
15. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST Special Publication 800-22 (with revisions dated May 15, 2001).
16. On the pseudo-random generator ISAAC. Jean-Philippe Aumasson. FHNW, 5210 Windisch, Switzerland.
17. Principles of Construction and Usage of Pseudorandom Generators. Alexander Vakhitov. June 13, 2005.
18. Cryptographic PseudoRandom Numbers. Brian Gernhardt. Rochester Institute of Technology. Cryptography II, Spring 2010.
19. Authentication by using pseudo-random numbers. Arziyeva J.T., Xudoyqulov Z.T., Mardiyev U.R. International Scientific and Practical Conference “INNOVATION-2012”.
20. Симметрик криптотизимларда калит генерациялаш алгоритмлари тадқиқи. Қурбонов Х.А., Худойқулов З.Т., Фатиллаев Б.Қ. Республиканиский семинар: «Информационная безопастность в

сфере связи и информатизации. Проблемы и пути их решения». Ташкент 30 октября 2012 г.

21. http://en.wikipedia.org/wiki/Linear_congruential_generator
22. http://en.wikipedia.org/wiki/RC4
23. http://en.wikipedia.org/wiki/ISAAC_(cipher)
24. http://random.com.hr/products/random/Diehard.html
25. http://en.wikipedia.org/wiki/Kolmogorov-Smirnov

Статистик тестлар

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Extreme.Mathematics;

namespace Statistical_Tests
{
   class Other_ST_test
   {
      private void Form1_Load(object sender, EventArgs e)
      {

      }
      public double Monobit(string key)
      {
         string kalit = key;
         int sum = 0;
         for (int i = 0; i < kalit.Length; i++)
         {
            if (kalit[i].ToString() == "1") { sum += 1; } else { sum -= 1; }
         }
         double Sobs = Math.Abs(sum) /
Math.Sqrt(Convert.ToDouble(kalit.Length));
         return SpecialFunction.erfc(Sobs / Math.Sqrt(2));
      } // tayyor
      public double Frequency(string key, int block_length)
      {

         int N = key.Length / block_length;
         string kalit = key.Substring(0,N*block_length);
         double[] pi = new double[N];
         for (int i = 0; i < N; i++)
         {
            double surat = 0;
            string str = kalit.Substring(i*block_length, block_length);
            for (int j = 0; j < str.Length; j++)
            {
               surat += Convert.ToDouble(str[j].ToString());
            }
            // System.Windows.Forms.MessageBox.Show(surat.ToString());
```

```
            pi[i] = surat / (double)block_length;
        }
        double sum = 0;
        for (int k = 0; k < N; k++)
        {
            sum += Math.Pow((pi[k] - 0.5), 2);
        }
        double x2 = (double)4 * (double)block_length * sum;
        return SpecialFunction.igamc((double)N / (double)2, x2/(double)2);
    } //tayyor
    public double Runs(string key) //tayyor
    {
        double sum = 0;
        for (int i = 0; i < key.Length; i++)
        {
            if (key[i].ToString() == "1")
            {
                sum += 1;
            }
            else { sum += 0; }
        }
        double vobs = 1;
        double pi = sum / key.Length;
        for (int j = 0; j < key.Length - 1; j++)
        {
            if (key[j] == key[j + 1])
            {
                vobs += 0;
            }
            else { vobs += 1; }
        }
        return SpecialFunction.erfc(Math.Abs(vobs - 2 * key.Length * pi * (1 -
pi)) / (2 * Math.Sqrt(2 * key.Length) * pi * (1 - pi)));
    }
    public double Longs(string key)
    {
        double xobs = 0;
        double result = 0;
        if (key.Length < 6272)
        {
            int N = key.Length / 8;
            double[] vi = { 0, 0, 0, 0 };
            double[] pi = { 0.2148, 0.3672, 0.2305, 0.1875 };
```

```
            double K = 3;
            for (int i = 0; i < N; i++)
            {
               string key1 = key.Substring(0, 8);
               if (soni_longs(key1) <= 1) { vi[0] += 1; } else if (soni_longs(key1)
== 2) { vi[1] += 1; } else if (soni_longs(key1) == 3) { vi[2] += 1; } else vi[3] +=
1;
               key = key.Remove(0, 8).ToString();
            }
            for (int i = 0; i <= K; i++)
            {
               xobs += Math.Pow((vi[i] -(double) N * pi[i]), 2) / ((double)N *
pi[i]);
            }
            result = SpecialFunction.igamc(K / 2, xobs / 2);
         }
         if ((key.Length >= 6272) && (key.Length < 750000))
         {
            //double M=128;
            int N = key.Length / 128;
            double[] vi = new double[6];
            double[] pi = { 0.1174, 0.2430, 0.2493, 0.1752, 0.1027, 0.1124 };
            double K = 5;
            for (int i = 0; i < N; i++)
            {
               string key1 = key.Substring(0, 128);
               if (soni_longs(key1) <= 4) { vi[0] += 1; }
               else if (soni_longs(key1) == 5) { vi[1] += 1; }
               else if (soni_longs(key1) == 6) { vi[2] += 1; }
               else if (soni_longs(key1) == 7) { vi[3] += 1; }
               else if (soni_longs(key1) == 8) { vi[4] += 1; }
               else if (soni_longs(key1) >= 9) { vi[5] += 1; }
               key = key.Remove(0, 128).ToString();
            }
            for (int i = 0; i <= K; i++)
            {
               xobs += Math.Pow((vi[i] - (double)N * pi[i]), 2) / ((double)N *
pi[i]);
            }
            result = SpecialFunction.igamc(K / 2, xobs / 2);
         }
         else if ((key.Length >= 750000))
         {
```

```
            //double M=10000;
            int N = key.Length / 10000;
            double[] vi = new double[7];
            double[] pi = { 0.0882, 0.2092, 0.2483, 0.1933, 0.1208, 0.0675,
0.0727 };
            double K = 6;
            for (int i = 0; i < N; i++)
            {
               string key1 = key.Substring(0, 10000);
               if (soni_longs(key1) <= 10) { vi[0] += 1; }
               else if (soni_longs(key1) == 11) { vi[1] += 1; }
               else if (soni_longs(key1) == 12) { vi[2] += 1; }
               else if (soni_longs(key1) == 13) { vi[3] += 1; }
               else if (soni_longs(key1) == 14) { vi[4] += 1; }
               else if (soni_longs(key1) == 15) { vi[5] += 1; }
               else if (soni_longs(key1) >= 16) { vi[6] += 1; }
               key = key.Remove(0, 10000).ToString();
            }
            for (int i = 0; i <= K; i++)
            {
               xobs += Math.Pow((vi[i] - (double)N * pi[i]), 2) / ((double)N *
pi[i]);
            }
            result = SpecialFunction.igamc(K / 2, xobs / 2);
         }
         return result; ;
      }//tayyor
      public double Binary_matrix(string key)
      {
         int N = key.Length / (32 * 32);
         string str = key.Substring(0, N * 1024);
         string suz;
         int[] rank = new int[N];
         double[] iii = new double[32];
         for (int k = 0; k < N; k++)
         {
            suz = str.Substring(0, 1024);
            Matrix mat = Matrix.Create(32, 32);
            for (int i = 0; i < 32; i++)
            {
               for (int j = 0; j < 32; j++)
               {
                  mat[i, j] = Convert.ToInt16(suz[0]);
```

```
                suz = suz.Remove(0, 1);
            }
          }
          str = str.Remove(0, 1024).ToString();
          rank[k] += mat.Rank();
        }
        int max = rank.Max();
        int max_son = 0;
        int max_son_1 = 0;
        for (int i = 0; i < N; i++)
        {
          if (rank[i] == max) { max_son++; } else if (rank[i] == max - 1) {
max_son_1++; }
        }
        double xobs = (Math.Pow((max_son - 0.2888 * N), 2) / (0.2888 * N)) +
(Math.Pow((max_son_1 - 0.5776 * N), 2) / (0.5776 * N)) + (Math.Pow((N -
max_son - max_son_1 - 0.1336 * N), 2) / (0.1336 * N));
        double result = SpecialFunction.igamc(1, xobs / 2);
        return Math.Pow(Math.E,(-1)*xobs/2);
      }//tayyor
      public double transform_test(string key)
      {
        int n = key.Length;
        double[] data = new double[n];
        double N0 = 0.95 * n / 2;
        double[] dft1;
        double sum = 0;
        for (int i = 0; i < n / 2; i++)
        {
          data[i] = 2 * Convert.ToDouble(key[i].ToString()) - 1;
        }
        dft1 = dft(data);
        for (int i = 0; i < dft1.Length; i++)
        {
          sum += dft1[i];
        }
        Random rand = new Random();
        int random = rand.Next(Convert.ToInt16(N0 - sum),
Convert.ToInt16(N0));
        double d = (N0 - random) / Math.Sqrt(n * 0.95 * 0.05 / 2);
        return SpecialFunction.erfc(Math.Abs(d) /Math.Sqrt(2));
       // return random;
      }//tayyor
```

```
        public double matching_test(string key, string b)
        {
            int m = b.Length; ;
            string B = b;
            int M = 131072;
            int N = key.Length / M;
            string[] array = new string[N];
            int[] son = new int[N];
            for (int i = 0; i < N; i++)
            {
                string str = key.Substring(i*M, M);
                array[i] = str;
            }
            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < M/m; j++)
                {
                    if (array[i].Substring(j*m, m).ToString() == B)
                    {
                        son[i] += 1;
                    }
                    else { son[i] += 0; }
                }
            }
            double myu = (M - m + 1) / Math.Pow(2, (double)m);
            double x = 0;
            double sigma = M * ((1 / Math.Pow(2, (double)m)) - ((2 * (double)m - 1)
/ Math.Pow(2, (double)m * 2)));
            for (int i = 0; i < N; i++)
            {
                x += Math.Pow(((double)son[i] - myu), 2);
            }
            double x2 = x / sigma;
            return SpecialFunction.igamc((double)N / 2, x2 / 2);
        }//tayyor
        public double tem_natching_test(string key, string b)
        {
            int m = b.Length; ;
            string B = b;
            int M = 1032;
            int K = 5;
            int N = key.Length / M;
            string[] array = new string[N];
```

```
            int[] son = new int[N];

            for (int i = 0; i < N; i++)
            {
               string str = key.Substring(i*M,M);
               array[i] = str;
            }
            for (int i = 0; i < array.Length; i++)
            {
               for (int j = 0; j < M/m; j++)
               {
                  if (array[i].Substring(j*m, m).ToString() == B)
                  {
                     son[i] += 1;
                  }
                  else { son[i] += 0; }
               }
            }
            double myu = (M - m + 1) / Math.Pow(2, (double)m);
            double nyu = myu / 2;
            double x = 0;
            double sigma = M * ((1 / Math.Pow(2, (double)m)) - ((2 * (double)m - 1)
/ Math.Pow(2, (double)m * 2)));
            for (int i = 0; i < N; i++)
            {
               x += Math.Pow(((double)son[i] - myu), 2);
            }
            double x2 = x / sigma;
            return SpecialFunction.igamc((double)N / 2, x2 / 2);
         }//tayyor
         public double universal_test(string key)
         {
            int L, Q, K;
            double EV = 0;
            double V = 0;
            double n = (double)key.Length;
            if (n < 387840) { L = 5; Q = 320; }
            else if ((n >= 387840) && (n < 904960)) { L = 6; Q = 640; EV =
5.2177052; V = 2.954; }
            else if ((n >= 904960) && (n < 2068480)) { L = 7; Q = 1280; EV =
6.1962507; V = 3.125; }
            else if ((n >= 2068480) && (n < 4654080)) { L = 8; Q = 2560; EV =
7.1836656; V = 3.238; }
```

```
        else if ((n >= 4654080) && (n < 1342400)) { L = 9; Q = 5120; EV =
8.1764248; V = 3.311; }
        else if ((n >= 1342400) && (n < 22753280)) { L = 10; Q = 10240; EV =
9.1723248; V = 3.356; }
        else if ((n >= 22753280) && (n < 49643520)) { L = 11; Q = 20480; EV
= 10.170032; V = 3.384; }
        else if ((n >= 49643520) && (n < 107560960)) { L = 12; Q = 40960; EV
= 11.168765; V = 3.401; }
        else if ((n >= 107560960) && (n < 231669760)) { L = 13; Q = 81920;
EV = 12.168070; V = 3.410; }
        else if ((n >= 231669760) && (n < 496435200)) { L = 14; Q = 163840;
EV = 13.167693; V = 3.416; }
        else if ((n >= 496435200) && (n < 1059061760)) { L = 15; Q = 327680;
EV = 14.167488; V = 3.416; }
        else { L = 16; Q = 655360; EV = 15.167379; V = 3.421; }
        K = key.Length / L - Q;
        string init_seg = key.Substring(0, Q * L);
        string test_seg = key.Remove(0, Q * L);
        int[] son_init = new int[Q];
        list(L);
        for (int i = 0; i < Q; i++)
        {
           int kalit = dictionary.Where(kvp => kvp.Value == init_seg.Substring(i
* L, L).ToString()).Select(kvp => kvp.Key).FirstOrDefault();
           if (dictionary.ContainsValue(init_seg.Substring(i * L, L).ToString()))
{ son_init[kalit] = i + 1; }
        }
        double sum=0.0; int sanoq = Q+1;
        for (int j = 0; j < K; j++)
        {
           int kalit = dictionary.Where(kvp => kvp.Value == test_seg.Substring(j
* L, L).ToString()).Select(kvp => kvp.Key).FirstOrDefault();
           if (dictionary.ContainsValue(test_seg.Substring(j * L, L).ToString()))
{ sum = sum + Math.Log((double)(sanoq - son_init[kalit]), 2); son_init[kalit] =
sanoq; sanoq = sanoq + 1; }
        }
        double c = 0.7 - (0.8 / (double)L) + ((double)4 + ((double)32 /
(double)L)) * ((Math.Pow((double)K, ((double)-3 / (double)L))) / (double)15);
        double sigma = c * (double)Math.Sqrt(V / (double)K);
        double p = SpecialFunction.erfc(Math.Abs((sum / ((double)K) - EV) /
(Math.Sqrt((double)2) * sigma)));
        return p;
      }//tayyor
```

```
private Dictionary<int, string> dictionary = new Dictionary<int, string>();
private void list(int l)
{
   for (int i = 0; i < (int)(Math.Pow(2,l)); i++)
   {
      dictionary.Add(i,FromDecimalToBinary(i, l));
   }
}
private string FromDecimalToBinary(int binary,int l)
{
   if (binary < 0)
   {
      Console.WriteLine("It requires a integer greater than 0.");
      return null;
   }
   string binarystring = "";
   int factor = (int)Math.Pow(2,(double)(l-1));
   for (int i = 0; i < l; i++)
   {
      if (binary >= factor)
      {
         binary -= factor;
         binarystring += "1";
      }
      else
      {
         binarystring += "0";
      }
      factor /= 2;
   }

   return binarystring;
   //if (binary < 0)
   //{
   //   Console.WriteLine("It requires a integer greater than 0.");
   //   return null;
   //}
   //string binarystring = Convert.ToString(binary,2);
   //while (binarystring.Length < l)
   //{
   //   binarystring = "0" + binarystring;
   //}
   //return binarystring;
```

```
}
public double LVZ_test(string key)
{
   List<string> compressed = qisish(key);
   string str = string.Empty;
   for (int i = 0; i < compressed.Count; i++)
   {
      str += compressed[i].ToString() + ",";
   }
   int w = compressed.Count;
   double myu = 69586.25;
   double sigma = Math.Sqrt(2* 70.448718);
   double p = 0.5 * SpecialFunction.erfc((myu - (double)w) / sigma);
   return p;
}//tayyor
public static List<string> qisish(string qisilmagan)
{
   Dictionary<string, int> dictionary = new Dictionary<string, int>();
   string w = string.Empty;
   List<string> compressed = new List<string>();
   for (int i = 0; i < qisilmagan.Length; i++)
   {
      string wc = w + qisilmagan[i];
      if (dictionary.ContainsKey(wc))
      {
         w = wc;
      }
      else
      {
         compressed.Add(wc.ToString());
         dictionary.Add(wc, dictionary.Count);
         w = string.Empty;
      }
   }
   if (!string.IsNullOrEmpty(w))
      compressed.Add(dictionary[w].ToString());
   return compressed;
}//q
public double soni_longs(string key)
{
   string a = key;
   int[] b = new int[key.Length];
   int n = 0;
```

```
    int soni = 0;
    int max = 0;
    for (int i = 0; i < key.Length; i++)
    {
        if (a[i].ToString() == "1")
            n++;
        else
        {
            b[soni] = n;
            soni++;
            n = 0;
        }
        if (i == key.Length - 1)
            b[soni] = n;
    }
    for (int i = 0; i < key.Length; i++)
    {
        if (max < b[i])
            max = b[i];
    }
    return max;
}//q
private double[] dft(double[] data)
{
    int n = data.Length;
    int m = n;// I use m = n / 2d;
    double[] real = new double[n];
    double[] imag = new double[n];
    double[] result = new double[m];
    double pi_div = 2.0 * Math.PI / n;
    for (int w = 0; w < m; w++)
    {
        double a = w * pi_div;
        for (int t = 0; t < n; t++)
        {
            real[w] += data[t] * Math.Cos(a * t);
            imag[w] += data[t] * Math.Sin(a * t);
        }
        result[w] = Math.Sqrt(real[w] * real[w] + imag[w] * imag[w]) / n;
    }
    return result;
}//q
public double linear_test(string key, int m)
```

```
        {
            int M = m;
            int n = key.Length;
            int N = n / M;
            double[] vi = new double[7]{0,0,0,0,0,0,0};
            double Ti;
            double myu = (double)M/2 +Convert.ToDouble(9+Math.Pow((-
1),M))/(double)36 + ((double)(M) /(double)3 + (double)(2) /(double)9) /
(Math.Pow((double)2,(double)M));
            for (int i = 0; i < N; i++)
            {
                string str = key.Substring(i*M, M);
                double Li = soni_longs(str);
                //System.Windows.Forms.MessageBox.Show(Li.ToString());
                Ti = (Li - myu) + Convert.ToDouble(2)/(double)9;
                if (Ti <= -2.5) { vi[0] += 1; }
                else if ((-2.5 < Ti) && (Ti <= -1.5)) { vi[1] += 1; }
                else if ((-.5 < Ti) && (Ti <= -0.5)) { vi[2] += 1; }
                else if ((-0.5 < Ti) && (Ti <= 0.5)) { vi[3] += 1; }
                else if ((0.5 < Ti) && (Ti <= 1.5)) { vi[4] += 1; }
                else if ((1.5 < Ti) && (Ti <= 2.5)) { vi[5] += 1; }
                else { vi[6] += 1; }
            }
            double[] pi = { 0.01047, 0.03125, 0.125, 0.5, 0.25, 0.0625, 0.02078 };
            double xobs = 0;
            for (int i = 0; i < 7; i++)
            {
                xobs += (Math.Pow((Convert.ToDouble(vi[i]) - Convert.ToDouble(N)
* pi[i]), Convert.ToDouble(2))) / (Convert.ToDouble(N) * pi[i]);

            }
           return SpecialFunction.igamc(3, xobs / 2);
        }//tayyor
        public double serial_test(string key)
        {
            int n = key.Length;
            int m = 2;
            int o = 0;
            int ii = 0;
            int[] m2 = new int[4];
            double sum = 0;
            string str2 = key + "00".ToString();
            for (int i = 0; i < str2.Length - 1; i++)
```

```
    {
        if (str2.Substring(i, 2) == "00".ToString()) { m2[0] += 1; }
        else if (str2.Substring(i, 2) == "01".ToString()) { m2[1] += 1; }
        else if (str2.Substring(i, 2) == "10".ToString()) { m2[2] += 1; }
        else if (str2.Substring(i, 2) == "11".ToString()) { m2[3] += 1; }
    }
    for (int k = 0; k < n; k++)
    {
        if (key[k].ToString() == "0") { o += 1; }
        else ii += 1;
    }
    for (int j = 0; j < m2.Length; j++)
    {
        double bir = m2[j] * m2[j];
        sum = sum + bir;
    }
    double w22 = (Math.Pow(2, 2) * (sum) / n) - n;
    double w12 = (1 * (o * o + ii * ii) / n) - n;
    double w02 = 0;
    double wm2 = w22 * w22 - w12 * w12;
    double wm22 = w22 * w22 - 2 * w12 * w12 + w02;
    double p1 = SpecialFunction.igamc(1, wm2 / 2);
    double p2 = SpecialFunction.igamc(1 / 2, wm22 / 2);
    return p2;
}// tayyor
public double entropy_test(string key)
{
    int m = 3;
    int n = key.Length;
    string str1 = key + key.Substring(0, 2);
    int[] soni = new int[8];
    for (int i = 0; i < n; i++)
    {
        if (str1.Substring(i, 3) == "000") { soni[0] += 1; }
        else if (str1.Substring(i, 3) == "001") { soni[1] += 1; }
        else if (str1.Substring(i, 3) == "010") { soni[2] += 1; }
        else if (str1.Substring(i, 3) == "011") { soni[3] += 1; }
        else if (str1.Substring(i, 3) == "100") { soni[4] += 1; }
        else if (str1.Substring(i, 3) == "101") { soni[5] += 1; }
        else if (str1.Substring(i, 3) == "110") { soni[6] += 1; }
        else if (str1.Substring(i, 3) == "111") { soni[7] += 1; }
    }
    double fm = 0;
```

```
        for (int j = 0; j < 8; j++)
        {
            if (soni[j] != 0)
            {
                fm += soni[j] * Math.Log(Convert.ToDouble(soni[j]) /
Convert.ToDouble(n)) / Convert.ToDouble(n);
            }
        }
        string str2 = key + key.Substring(0, 3);
        int[] soni2 = new int[16];
        for (int k = 0; k < n; k++)
        {
            if (str2.Substring(k, 4) == "0000") { soni2[0] += 1; }
            else if (str2.Substring(k, 4) == "0001") { soni2[1] += 1; }
            else if (str2.Substring(k, 4) == "0010") { soni2[2] += 1; }
            else if (str2.Substring(k, 4) == "0011") { soni2[3] += 1; }
            else if (str2.Substring(k, 4) == "0100") { soni2[4] += 1; }
            else if (str2.Substring(k, 4) == "0101") { soni2[5] += 1; }
            else if (str2.Substring(k, 4) == "0110") { soni2[6] += 1; }
            else if (str2.Substring(k, 4) == "0111") { soni2[7] += 1; }
            else if (str2.Substring(k, 4) == "1000") { soni2[8] += 1; }
            else if (str2.Substring(k, 4) == "1001") { soni2[9] += 1; }
            else if (str2.Substring(k, 4) == "1010") { soni2[10] += 1; }
            else if (str2.Substring(k, 4) == "1011") { soni2[11] += 1; }
            else if (str2.Substring(k, 4) == "1100") { soni2[12] += 1; }
            else if (str2.Substring(k, 4) == "1101") { soni2[13] += 1; }
            else if (str2.Substring(k, 4) == "1110") { soni2[14] += 1; }
            else if (str2.Substring(k, 4) == "1111") { soni2[15] += 1; }
        }
        double fm2 = 0;
        for (int g = 0; g < 16; g++)
        {
            if (soni2[g] == 0) { fm2 += 0; }
            else
            {
                fm2 += soni2[g] * Math.Log(Convert.ToDouble(soni2[g]) /
Convert.ToDouble(n)) / Convert.ToDouble(n);
            }
        }
        double qoldiq = fm - fm2;
        double xm = 2 * n * (Math.Log(2) - qoldiq);
        //return fm;
        return SpecialFunction.igamc(4, xm / 2);
```

```
}//tayyor
public double cumulative_test(string key)
{
   int n = key.Length;
   double[] x = new double[n];
   double[] sum = new double[n];
   for (int i = 0; i < n; i++)
   {
      if (key[i].ToString() == "0") { x[i] += -1; } else { x[i] += 1; }
   }
   double qush = 0;
   for (int j = 0; j < n; j++)
   {
      qush += x[j];
      sum[j] = qush;
   }
   double max = sum.Max();
   double index_max = Array.IndexOf(sum, max);
   //double[] sum_1 = new double[n];
   //double qush_1 = 0;
   //for (int k = n - 1; k <= 0; k--)
   //{
   //    qush_1 += x[k];
   //    sum_1[k] = qush_1;
   //}
   //double max_1 = sum_1.Max();
   //double index_max_1=Array.IndexOf(sum_1,max_1);
   int gg = Convert.ToInt16((max - n) / (4 * max));
   int ff = Convert.ToInt16((n - max) / (4 * max));
   int hh = Convert.ToInt16(((-1 * n) / max) / 4);
   double qoldiq1 = 0;
   for (int h = ff; h <= gg; h++)
   {
      qoldiq1 += (CND(((4 * h + 1) * max) / (Math.Sqrt(n))) - CND(((4 * h
- 1) * max) / (Math.Sqrt(n))));
   }
   double qoldiq2 = 0;
   for (int t = hh; t <= gg; t++)
   {
      qoldiq2 += (CND(((4 * t + 3) * max) / (Math.Sqrt(n))) - CND(((4 * t +
1) * max) / (Math.Sqrt(n))));
   }
   double p = 1 - qoldiq1 + qoldiq2;
```

```
            return p;
        }//tayyor
        public double CND(double d)
        {
            const double A1 = 0.31938153;
            const double A2 = -0.356563782;
            const double A3 = 1.781477937;
            const double A4 = -1.821255978;
            const double A5 = 1.330274429;
            const double RSQRT2PI = 0.39894228040143267793994605993438;

            double
            K = 1.0 / (1.0 + 0.2316419 * Math.Abs(d));

            double
            cnd = RSQRT2PI * Math.Exp(-0.5 * d * d) * (K * (A1 + K * (A2 + K *
(A3 + K * (A4 + K * A5)))));

            if (d > 0)
                cnd = 1.0 - cnd;

            return cnd;
        }//q
        public int[,] massiv;
        public double[] excursion_test(string key)
        {
            int n = key.Length;
            int[] x = new int[n];
            int[] sum = new int[n + 2];
            for (int i = 0; i < n; i++)
            {
                if (key[i].ToString() == "0") { x[i] = -1; } else { x[i] = 1; }
            }
            int qush = 0;
            for (int j = 0; j < n; j++)
            {
                qush += x[j];
                sum[j + 1] = qush;
            }
            sum.SetValue(0, 0);
            sum.SetValue(0, n + 1);
            List<int> list = new List<int>();
            for (int k = 0; k < n + 2; k++)
```

```
        {
            if (sum[k] == 0)
            {
                list.Add(k);
            }
        }
        int[] array = list.ToArray();
        List<int[]> splitted = new List<int[]>();//This list will contain all the
splitted arrays.
        for (int i = 1; i < array.Length; i++)
        {
            int uzun = array[i] - array[i - 1] + 1;
            int[] qism = new int[uzun];
            Array.Copy(sum, array[i - 1], qism, 0, uzun);
            splitted.Add(qism);
        }
        massiv = new int[8, 6];
        for (int i = 0; i < 8; i++)
        {
            for (int j = 0; j < 6; j++)
            {
                massiv[i, j] = 0;
            }
        }
        for (int i = 0; i < array.Length-1; i++)
        {
            creatmassiv(splitted[i], splitted[i].Length);
        }
        double[,] px = new double[8, 6]
        {
            {0.8750, 0.0156, 0.0137, 0.0120, 0.0105, 0.0733},
            {0.8333, 0.0278, 0.0231, 0.0193, 0.0161, 0.0804},
            {0.7500, 0.0625, 0.0469, 0.0352, 0.0264, 0.0791},
            {0.5000, 0.2500, 0.1250, 0.0625, 0.0312, 0.0312},
            {0.5000, 0.2500, 0.1250, 0.0625, 0.0312, 0.0312},
            {0.7500, 0.0625, 0.0469, 0.0352, 0.0264, 0.0791},
            {0.8333, 0.0278, 0.0231, 0.0193, 0.0161, 0.0804},
            {0.8750, 0.0156, 0.0137, 0.0120, 0.0105, 0.0733}
            //{0.9000, 0.0100, 0.0090, 0.0081, 0.0073, 0.0656},
            //{0.9167, 0.0069, 0.0064, 0.0058, 0.0053, 0.0588},
            //{0.9286, 0.0051, 0.0047, 0.0044, 0.0041, 0.0531},
        };
        int katta_j = array.Length - 1;
```

```
            double[] p_qiymat=new double[8];
            for (int i = 0; i < 8; i++)
            {
                double qiymat = 0;
                for (int j = 0; j < 6; j++)
                {
                    qiymat += Math.Pow(((double)massiv[i, j] - (double)katta_j * px[i,
j]), 2) / ((double)katta_j * px[i, j]);

                }
                double xkav = qiymat;
                p_qiymat[i] = SpecialFunction.igamc(2.5,xkav/2);
            }
            return p_qiymat;
        }//tayyor
        public void creatmassiv(int [] a, int b)
        {
            int mtort = 0;
            int much = 0;
            int mikki = 0;
            int mbir = 0;
            int bir = 0;
            int ikki = 0;
            int uch = 0;
            int tort = 0;
            for (int i = 0; i < b; i++)
            {
                switch (a[i])
                {
                    case -4:
                        mtort++;
                        break;
                    case -3:
                        much++;
                        break;
                    case -2:
                        mikki++;
                        break;
                    case -1:
                        mbir++;
                        break;
                    case 1:
                        bir++;
```

```
            break;
        case 2:
            ikki++;
            break;
        case 3:
            uch++;
            break;
        case 4:
            tort++;
            break;
        default: break;
      }
   }
   if (mtort > 5) { mtort = 5; }
   if (much > 5) { much = 5; }
   if (mikki > 5) { mikki = 5; }
   if (mbir > 5) { mbir = 5; }
   if (bir > 5) { bir = 5; }
   if (ikki > 5) { ikki = 5; }
   if (uch >5) { uch = 5; }
   if (tort > 5) { tort = 5; }
   massiv[0, mtort] = massiv[0, mtort] + 1;
   massiv[1, much] = massiv[1, much] + 1;
   massiv[2, mikki] = massiv[2, mikki] + 1;
   massiv[3, mbir] = massiv[3, mbir] + 1;
   massiv[4, bir] = massiv[4, bir] + 1;
   massiv[5, ikki] = massiv[5, ikki] + 1;
   massiv[6, uch] = massiv[6, uch] + 1;
   massiv[7, tort] = massiv[7, tort] + 1;
}//adding
public double[] random_excursion_test(string key)
{
   int n = key.Length;
   double[] x = new double[n];
   double[] sum = new double[n + 2];
   sum.SetValue(0, 0);
   for (int i = 0; i < n; i++)
   {
      if (key[i].ToString() == "0") { x[i] += -1; } else { x[i] += 1; }
   }
   double qush = 0;
   for (int j = 0; j < n; j++)
   {
```

```
            qush += x[j];
            sum[j + 1] = qush;
        }
        sum.SetValue(0, n + 1);
        int[] test = new int[18];
        for (int i = 0; i < sum.Length; i++)
        {
            if (sum[i] == (-9)) { test[0] += 1; }
            else if (sum[i] == (-8)) { test[1] += 1; }
            else if (sum[i] == (-7)) { test[2] += 1; }
            else if (sum[i] == (-6)) { test[3] += 1; }
            else if (sum[i] == (-5)) { test[4] += 1; }
            else if (sum[i] == (-4)) { test[5] += 1; }
            else if (sum[i] == (-3)) { test[6] += 1; }
            else if (sum[i] == (-2)) { test[7] += 1; }
            else if (sum[i] == (-1)) { test[8] += 1; }
            else if (sum[i] == (1)) { test[9] += 1; }
            else if (sum[i] == (2)) { test[10] += 1; }
            else if (sum[i] == (3)) { test[11] += 1; }
            else if (sum[i] == (4)) { test[12] += 1; }
            else if (sum[i] == (5)) { test[13] += 1; }
            else if (sum[i] == (6)) { test[14] += 1; }
            else if (sum[i] == (7)) { test[15] += 1; }
            else if (sum[i] == (8)) { test[16] += 1; }
            else if (sum[i] == (9)) { test[17] += 1; }
        }
        int son = -1;
        int[] ss = { -9, -8, -7, -6, -5, -4, -3, -2, -1, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        for (int j = 0; j < sum.Length; j++)
        {
            if (sum[j] == 0) { son += 1; }
        }
        double[] p = new double[18];
        for (int k = 0; k < test.Length; k++)
        {
            p[k] += SpecialFunction.erfc((Math.Abs(test[k] - son)) / Math.Sqrt(2
* son * (4 * Math.Abs(ss[k]) - 2)));
        }
        return p;
    }//tayyor
  }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Statistical_Tests
{
   class Chi_square
   {
      public double sum;
      public string non_key(string str)
      {
         string satr = str;
         int son = satr.Length / 256;
         for (int i = 0; i < son; i++)
         {
            string qator = satr.Substring(0, 256);
            int bir = 0;
            int nol = 0;
            for (int k = 0; k < 256; k++)
            {
               if (qator[k].ToString() == "0") { nol++; } else { bir++; }
            }
            satr = satr.Remove(0, 256);
            hisobla(bir, nol);
         }
         return check(sum / (double)son).ToString() + " " + (sum /
(double)son).ToString();
      }
      private void hisobla(int bir, int nol)
      {
         sum = sum + (Math.Pow((bir - 128), 2) + Math.Pow((nol - 128), 2)) /
128;
      }
      public string with_key(string str, string non_sec_key)
      {
         string nat_key = string.Empty;
         for (int i = 0; i < str.Length; i++)
         {
            nat_key +=
Convert.ToChar(Convert.ToInt16(str[i].ToString())^Convert.ToInt16(non_sec_
key[i].ToString()));
         }
```

```
            return non_key(nat_key);
        }
        private bool check(double son)
        {
            if ((son > 0.1015) && (son < 1.323)) { return true; } else return false;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Statistical_Tests
{
    class KS_test
    {
        public string KS_tes(string str, double alfa)
        {
            int son = str.Length / 8;
            string qator = str.Substring(0, 8 * son);
            double[] xi = new double[son];
            for (int i = 0; i < son; i++)
            {
                xi[i] += Convert.ToDouble(Convert.ToInt32(qator.Substring(i*8, 8),
2)) / (Convert.ToDouble(256));
            }
            Array.Sort(xi);
            double[] massiv = new double[son];
            for (int j = 0; j < son; j++)
            {
                massiv[j] += ((double)j / (double)son) - xi[j];
            }
            double[] massiv2 = new double[son];
            for (int j = 0; j < son; j++)
            {
                massiv2[j] += xi[j] - ((double)(j - 1) / (double)son);
            }
            Array.Sort(massiv);
            double K_max = Math.Sqrt(son) * massiv.Max();
            double K_min = Math.Sqrt(son) * massiv2.Max();
            if (D(alfa, son) == (double)4) { return "20&35"; }
```

```
        if ((D(alfa, son) > K_max) && (D(alfa, son) > K_min)) { return "Pass ";
}
        else return "Failed";
      }
      private double D(double alfa, int n)
      {
        double[,] massiv = new double[20, 5]
        {
        {0.9,0.925,0.95,0.975,0.995},
        {0.684,0.726,0.776,0.842,0.929},
        {0.565,0.597,0.642,0.708,0.828},
        {0.494,0.575,0.564,0.624,0.733},
        {0.446,0.424,0.510,0.454,0.669},
        {0.410,0.436,0.470,0.521,0.618},
        {0.381,0.405,0.438,0.486,0.577},
        {0.358,0.381,0.411,0.457,0.543},
        {0.339,0.360,0.388,0.432,0.514},
        {0.322,0.342,0.368,0.410,0.490},
        {0.307,0.326,0.452,0.391,0.468},
        {0.295,0.313,0.338,0.375,0.405},
        {0.284,0.302,0.325,0.361,0.433},
        {0.274,0.292,0.314,0.349,0.418},
        {0.266,0.293,0.304,0.338,0.404},
        {0.258,0.274,0.295,0.328,0.392},
        {0.250,0.266,0.286,0.318,0.381},
        {0.244,0.259,0.278,0.309,0.371},
        {0.237,0.252,0.272,0.301,0.363},
        {0.231,0.246,0.264,0.294,0.356}};
        if (n <= 20)
        {
          if (alfa == 0.2) { return massiv[n, 0]; }
          if (alfa == 0.15) { return massiv[n, 1]; }
          if (alfa == 0.1) { return massiv[n, 2]; }
          if (alfa == 0.05) { return massiv[n, 3]; }
          if (alfa == 0.01) { return massiv[n, 4]; }
        }
        if (n > 35)
        {
          if (alfa == 0.2) { return 1.07 / Math.Sqrt(n); }
          if (alfa == 0.15) { return 1.14 / Math.Sqrt(n); }
          if (alfa == 0.1) { return 1.22 / Math.Sqrt(n); }
          if (alfa == 0.05) { return 1.36 / Math.Sqrt(n); }
          if (alfa == 0.01) { return 1.63 / Math.Sqrt(n); }
```

```
            }
            return 4;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using System.ComponentModel;
using System.Threading;
using System.Windows.Forms;
using System.Diagnostics;
using System.Security.Cryptography;
namespace RC4
{
    class New_key_Generator
    {
        public string sha256_xeshing(string str)
        {
            byte[] bytes = Encoding.Unicode.GetBytes(str);
            SHA256Managed hashstring = new SHA256Managed();
            byte[] hash = hashstring.ComputeHash(bytes);
            string hashString = string.Empty;
            foreach (byte x in hash)
            {
                hashString += x.ToString("X");
            }
            return hashString;
        }
    }
}
```