ՀՀ ԳԻՏՈՒԹՅՈՒՆՆԵՐԻ ԱԶԳԱՅԻՆ ԱԿԱԴԵՄԻԱ
ԻՆՖՈՐՄԱՏԻԿԱՅԻ ԵՎ ԱՎՏՈՄԱՏԱՑՄԱՆ ՊՐՈԲԼԵՄՆԵՐԻ
ԻՆՍՏԻՏՈՒՏ

Կյուրեղյան Քնարիկ Մարտինի

SAFER ԸՆՏԱՆԻՔԻ 256 ԲԻԹԱՅԻՆ ԾԱԾԿԱԳՐԱԿԱՆ ՀԱՄԱԿԱՐԳԻ ՄՇԱԿՈՒՄ ԵՎ ԻՐԱԿԱՆԱՑՈՒՄ

ԱՏԵՆԱԽՈՍՈՒԹՅՈՒՆ

Ե.13.05 – «Մաթեմատիկական մոդելավորում, թվային մեթոդներ և ծրագրերի համալիրներ» մասնագիտությամբ տեխնիկական գիտությունների թեկնածուի գիտական աստիճանի համար

Գիտական ղեկավար՝ Ֆ.մ.գ.թ. Մելսիկ Կյուրեղյան

Երևան 2017

Բովանդակություն

Ներածություն

Թեմայի արդիականությունը: Մենք ապրում ենք տեղեկատվական դարաշրջանում, որտեղ էլեկտրոնային ծառայություններից կախվածությունը օրեցօր դառնում է ավելի ակնառու, իսկ գաղտնի ինֆորմացիայի անվտանգության ապահովման խնդիրը դառնում է առավել անհրաժեշտ: Հարկավոր է անհընդհատ կատարելագործել ինֆորմացիոն անվտանգության համակարգերի հիմքում ընկած ծածկագրական համակարգերի կրիպտոկայունությունը և արագագործությունը, հայտնաբերել և արագ վերացնել խոցելի կողմերը, և մշակել առավել անվտանգ, արագագործ, փոքր ծավալի հիշողություն պահանջող նոր ծածկագրական համակարգեր:

Ծածկագրաբանությունը, կախված տվյալների ծածկագրման (encryption) և վերծանման (decryption) համար օգտագործվող բանալիներից, լինում է սիմետրիկ և ասիմետրիկ: Սիմետրիկ ծածկագրաբանության ծածկագրական համակարգերը (Symmetric Cryptosystems) կամ որ նույնն է գաղտնի բանալիով ծածկագրական համակարգերը (Private-Key Cryptosystems) տվյալների ծածկագրման և վերծանման համար օգտագործում են միևնույն (սիմետրիկ կամ գաղտնի) բանալին: Ասիմետրիկ ծածկագրաբանության ծածկագրական համակարգերը (Asymmetric Cryptosystems), որոնք ավելի հայտնի են որպես բաց բանալիով ծածկագրական համակարգեր (Public-Key Cryptosystems) անվանումով, օգտագործում են երկու իրարից տարբեր բանալիներ: Տվյալների ծածկագրման համար օգտագործվող բանալին դրվում է բացեիբաց և անվանում են բաց կամ հասարակական բանալի (public key), իսկ վերծանման համար օգտագործվող բանալին պահվում է գաղտնի և կոչվում է գաղտնի բանալի (private key): Գաղտնի բանալիով ծածկագրական համակարգերը ի տարբերություն բաց բանալիով ծածկագրական համակարգերի ավելի արագագործ են և պահանջում են ավելի փոքր ծավալի հիշողություն [33]: Առաջին դեպքում գաղտնի բանալու փոխանցման խնդիրը լուծվում է մասնավորապես բաց բանալիով ծածկագրաբանության միջոցով (Diffie-Helman key exchange protocol [2], RSA encryption algorithm [28], ElGamal [3], etc.):

Բլոկային ծածկագրական համակարգերը (block ciphers) սիմետրիկ բանալիով ալգորիթմներ են, որոնք ինֆորմացիան բաժանում են ֆիքսված փոքր երկարության

բայթերի հաջորդականությունների, որոնց անվանում են բլոկներ կամ բաց տեքստեր (plaintext), և յուրաքանչյուր բլոկ ծածկագրում են առանձին: Մասնավորապես, իտերատիվ (iterative) բլոկային ծածկագրական համակարգում բլոկի վրա մի քանի անգամ կիրառվում է միևնույն ձևափոխությունը՝ գաղտնի բանալուց գեներացված ենթաբանալիների մասնակցությամբ: Իտերացիան (ձևափոխությունը) անվանում են ռաունդ կամ ցիկլ:

Շենոնը ծածկագրաբանության տեսական հիմունքների մասին իր գիտական հոդվածներից մեկում [“Communication theory of secrecy systems,” Bell Systems Technical Journal28 (1949), 656 – 715] ներկայացրել է բլոկային ծածկագրական համակարգերի մշակման երկու կարևորագույն սկզբունք. դիֆուզիա (diffusion) և կոնֆուզիա (confusion). Դիֆուզիան (երբ չնչին փոփոխությունը ծածկագրվող բաց տեքստում բերում է զգալի փոփոխությունների ծածկագրված տեքստում (ciphertext)) նպաստում է, որ համակարգը ունենա կրիպտոկայունության բարձր մակարդակ և հետևաբար լինի արագագործ: Կոնֆուզիան (երբ բանալու և ծածկագրի միջև կապը հնարավորինս բարդ է) ապահովում է ծածկագրից բանալու վերծանման անհնարինությունը:

Սիմետրիկ ծածկագրաբանության բլոկային ծածկագրման ալգորիթմները օգտագործում են գաղտնիություն (confidentiality) ապահովելու համար, կիրառում են տվյալների ամբողջականության պահպանման մեխանիզմներում (data integrity mechanisms), ինքնության ճանաչման համակարգերում (message authentication techniques, entity authentication protocols) և սիմետրիկ բանալիով թվային ստորագրության սխեմաներում ((symmetric-key) digital signature schemes):

Գոյություն ունեն ծածկագրական հարձակումների (cryptographic attack) բազմաթիվ տեսակներ, որոնց էֆեկտիվությունը (այսինքն նրանց կիրառության դեպքում համակարգը ջարդելու, գաղտնի բանալին առանց բոլոր հնարավոր բանալիների փորձարկմամբ վերականգնելու հավանականությունը համեմատաբար ավելի մեծ է), հետևաբար՝ ընտրությունը և կիրառությունը կախված է ուսումնասիրվող համակարգի տեսակից:

Բլոկային ծածկագրական համակարգերի համար հայտնի հարձակումներից արդյունավետ են դիֆերենցիալ վերլուծությունը (differential cryptanalysis) [1],[8],[9],[20], [25] և գծային վերլուծությունը (linear cryptanalysis) [4],[6]: Գծային վերլուծությամբ հարձակումը էֆեկտիվ է հատկապես այն դեպքում, երբ համակարգի գործողությունը ըստ մոդուլ 2-ի գումարն է [5]: Դիֆերենցիալ և գծային վերլուծությունների հիման վրա անընդհատ մշակվում են նոր հաձակումներ [41-43], և կրպտոկայուն բլոկային ծածկագրական համակարգ մշակելու համար հարկավոր է հետազոտել բոլոր այդ հարձակումների նկատմամբ համակարգի խոցելի կողմերը և ի սկզբանե ապահովել նրա կրիպտոկայունությունը գոյություն ունեցող էֆեկտիվ հարձակումների նկատմամբ:

Սիմետրիկ ծածկագրաբանության SAFER ընտանիքը, որի անվանումը ընտրվել է պրոֆեսոր Ջեյմս Մեսիի կողմից և “Secure And Fast Encryption Routine” (Անվտանգ և արագ ծածկագրման ընթացակարգ) արտահայտության բառերի հապավումն է, իտերատիվ բլոկային ծածկագրական համակարգերի ընտանիք է: SAFER ընտանիքի տարբեր բլոկային ծածկագրական համակարգեր ստեղծվել են տարբեր ժամանակահատվածում և հետևաբար ունեն բլոկի տարբեր երկարություններ: SAFER ընտանիքի վերջին տարիներին մշակված 128 բիթ բլոկի երկարության SAFER+ բլոկային ծածկագրական համակարգը ստեղծվել է ՀՀ ԳԱԱ ակադեմիկոս Գուրգեն Խաչատրյանի և ֆ.մ.գ.թ. Մելսիկ Կյուրեղյանի հետ համատեղ՝ NIST (National Institute of Standards and Technology, US) կողմից հայտարարված AES (Advanced Encryption Standard) մրցույթին ներկայացնելու նպատակով [21],[31]:

SAFER ընտանիքի SAFER+ ծածկագրական համակարգը չի դադարում լինել ոլորտի ուշադրության կենտրոնում, մասնավորապես վերջին տասնամյակում հրատարկվել են SAFER ընտանիքի և SAFER+ համակարգի անվտանգությանը վերաբերվող մի շարք հոդվածներ, այնումենայնիվ դրանցից և ոչ մեկում SAFER+ համակարգի խոցելիություն չի հայտնաբերվել [39-41]: Բացի այդ SAFER+ ալգորիթմի հիման վրա մշակվել է թափանցիկ ծածկագրման (White-Box encryption) ալգորիթմ [44], որը ծածկագրումը կատարում է գաղտնի բանալու հիման վրա ստեղծված հատուկ աղյուսակների (look-up tabels) միջոցով: Թափանցիկ ծածկագրման ալգորիթմները կիրառվում են որպես բաց բանալիով ծածկագրական համակարգերի արագագործ այլընտրանք [44],[45]:

Գոյություն ունեցող բլոկային ծածկագրական համակարգերի բլոկի առավելագույն երկարությունը 128 բիթ է, և չնայած որ նրանք ունեն նաև 256 բիթ երկարության օգտագործողի կողմից ընտրված բանալու դեպքը, նրանք իրականում չեն ապահովում 256 բիթ անվտանգություն 128 բիթ երկարության բլոկի համար՝ բախում հարձակման նկատմամբ (Բախում հարձակում (collision attack). գտնել երկու իրարից տարբեր մուտքեր $m_1 \neq m_2$, այնպիսին, որ $\mathrm{hash}(m_1) = \mathrm{hash}(m_2)$) [17]: 256 բիթային ծածկագրման ալգորիթմը ապահովում է 256 բիթ անվտանգություն, սակայն կարևոր է, որ այն ունենա նաև համեմատաբար մեծ արագագործություն:

Ատենախոսության նպատակն է.

- Հետազոտել սիմետրիկ ծածկագրաբանության բլոկային ծածկագրական համակարգերի SAFER ընտանիքը և մշակել
    - 128 բիթ բլոկի երկարության և 128, 192, 256 բիթ երկարության բանալիներով SAFER+ համակարգի այնպիսի ձևափոխություններ, որոնք չեն վնասի համակարգի կրիպտոկայունությունը և կնպաստեն համակարգի արագագործությանը (ընդհուպ մինչև ռաունդների կրճատման միջոցով):
    - 256 բիթային ծածկագրական համակարգ, հաշվի առնելով դիֆերենցիալ վերլուծության նկատմամբ SAFER+ և ձևափոխած SAFER+ համակարգերի կրիպտոկայունության մակարդակը, որից կախված է համակարգի արագագործությունը:
- SAFER ընտանիքի ծածկագրական համակարգերում դիֆուզիան ապահովում է հակադարձելի գծային շերտը, որը կազմված է պսևդո-Հադամարի ձևափոխությունից և բայթային տեղադրությունից: Վերջինիս ճիշտ ընտրության դեպքում դիֆերենցիալ վերլուծության նկատմամբ կարելի է ապահովել կրիպտոկայունություն հնարավորինս փոքր թվով ռաունդներից հետո: Դիֆերենցիալ վերլուծութան համար ստեղծված ծրագրային փաթեթների միջոցով ուսումնասիրել, բլոկի բայթերի բոլոր հնարավոր տեղադրություններից որոնց դեպքում մշակված համակարգերում կարելի է ապահովել օպտիմալ դիֆուզիա, այսինքն կրիպտոկայունություն հնարավոր փոքր թվով ռաունդներից հետո:

Հետազոտման օբյեկտը սիմետրիկ ծածկագրաբանության բլոկային ծածկագրական համակարգերի SAFER ընտանիքն է և բլոկային ծածկագրական համակարգերի նկատմամբ էֆեկտիվ հայտնի հարձակումները:

Հետազոտման մեթոդները: Աշխատանքում օգտագործված են սիմետրիկ ծածկագրաբանության մեթոդներ:

Արդյունքների գիտական նորույթը.

- SAFER ընտանիքի 128 բիթային SAFER+ բլոկային ծածկագրական համակարգի հիման վրա մշակված 128 բիթային ծածկագրական համակարգը (Ձևափոխած SAFER+), որը 32 բիթ պրոցեսորի վրա ապահովում է մոտավորապես 1,7 անգամ ավելի մեծ արագագործություն, քան SAFER+ համակարգը:
- SAFER ընտանիքի 256 բիթային ծածկագրական համակարգը (SAFER-256), որը կրիպտոկայունության բարձր մակարդակի (այսինքն ռաունդների փոքր թվի) շնորհիվ 256 բիթ երկարության բլոկը ծածկագրում է 1,5 անգամ ավելի արագ, քան՝ 128 բիթային SAFER+ համակարգը: Նշենք, որ գոյություն ունեցող բլոկային ծածկագրական համակարգերի ծածկագրման բլոկի առավելագույն երկարությունը 128 բիթ է:

Ստացված արդյունքների կիրառական նշանակությունը: Աշխատանքում ստացված արդյունքները կարելի է կիրառել ծածկագրաբանության մեջ:

Պաշտպանության են ներկայացվում հետևյալ դրույթները.

- SAFER ընտանիքի 128 բիթային SAFER+ բլոկային ծածկագրական համակարգի հիման վրա մշակված, համարժեք կրիպտոկայունությամբ (SAFER ընտանիքի բլոկային ծածկագրական համակարգերի նկատմամբ արդյունավետ հարձակումների դիֆերենցիալ և գծային վերլուծությունների նկատմամբ), սակայն 32 բիթ պրոցեսորի վրա SAFER+ համակարգից 1,7 անգամ ավելի արագագործ 128 բիթ երկարությամբ բլոկի ծածկագրման նոր ալգորիթմ. ձևափոխած SAFER+:

- SAFER ընտանիքի բլոկային ծածկագրական համակարգերի հիման վրա մշակված՝ 256 բիթ երկարության բանալու կիրառմամբ 256 բիթ երկարությամբ բլոկի ծածկագրման ալգորիթմ: Նոր SAFER-256 համակարգը մշակված է մասնավորապես ձևափոխած SAFER+ ալգորիթմի հիման վրա և դիֆերենցիալ վերլուծության նկատմամբ կայուն է նվազագույնը 5 ռաունդի դեպքում, իսկ գծային վերլուծության նկատմամբ նվազագույնը 3 ռաունդի դեպքում: 256 բիթ երկարության բանալիով 256 բիթ երկարության ինֆորմացիան SAFER-256 համակարգը ծածկագրում է 1,5 անգամ ավելի արագ քան SAFER+ համակարգը:

Աշխատանքի արդյունքների հավաստիությունը հիմնավորվում է մշակված ծրագրային համակարգերի կիրառմամբ ստացված մի շարք փորձնական արդյունքներով:

Աշխատանքի արդյունքների ներդրվել են Երևանի կապի միջոցների գիտահետազոտական ինստիտուտում, այն է ատենախոսության շրջանակներում մշակված 128 և 256 բիթային բլոկային ծածկագրական համակարգերը օգտագործում են Ավտոմատ Հեռախոսային Համակարգի մեջ:

Հրատարակությունները: Ատենախոսության թեմայով հրատարակվել է վեց գիտական աշխատություն:

Ատենախոսության կառուցվածքը և ծավալը: Աշխատանքը բաղկացած է բովանդակությունից, ներածությունից, երեք գլուխներից, եզրակացությունից, օգտագործված գրականության ցանկից, որը պարունակում է 45 անվանում, 16 նկարից և 7 աղյուսակից: Աշխատանքի ծավալը 116 էջ է, իսկ մնացած 32 էջերում ներկայացված են 3 հավելվածները:

Ստացված արդյունքների ապրոբացիան: Ատենախոսության արդյունքները զեկուցվել են ՀՀ ԳԱԱ ԻԱՊԻ-ի ընդհանուր սեմինարներում ու կոդավորման և ազդանշանների մշակման լաբորատորիայի մասնագիտական սեմինարներում, հայկական մաթեմատիկական միության 2014թ. տարեկան նստաշրջանում (AMUAS-2014), վեբ-անվտանգությանը նվիրված կոնֆերանսում (ARMSec-2016):

ԳԼՈՒԽ 1

SAFER ԸՆՏԱՆԻՔԻ ԲԼՈԿԱՅԻՆ ԾԱԾԿԱԳՐԱԿԱՆ ՆՈՐ ՀԱՄԱԿԱՐԳԵՐ

1.1. Բլոկային ծածկագրական համակարգերի SAFER ընտանիքը

Բլոկային ծածկագրական համակարգերի SAFER ընտանիքը բաղկացած է SAFER K-64, SAFER K-128, SAFER SK-64, SAFER SK-128, SAFER SK-40, SAFER+ և SAFER++ համակարգերից [24]: Այս ընտանիքի բոլոր “SAFER K” և “SAFER SK” համակարգերում բլոկի երկարությունը 64 բիթ է, իսկ օգտագործողի կողմից ընտրված բանալու (user-selected key) երկարությունը՝ 40, 64 կամ 128 բիթ և բանալու երկարության հիման վրա էլ ընտրվել են համապատասխան համակարգերի անվանումները: Հետագայում մշակված SAFER+ և SAFER++ համակարգերում բլոկի երկարությունը 128 բիթ է, իսկ բանալու երկարությունը 128 կամ 256 բիթ, SAFER+ համակարգում առկա է նաև 192 բիթ երկարության բանալու դեպքը: “SAFER K” և “SAFER SK” համակարգերը ստեղծվել են պրոֆեսոր Ջեյմս Մեսիի կողմից, Cylink ընկերության պատվերով, առանց հեղինակային իրավունքի պաշտպանման: Առաջին ծածկագրական SAFER K-64 համակարգը առաջին անգամ ներկայացվել է Անգլիայի Քեմբրիջի համալսարանում՝ ծածկագրման արագ ալգորիթմներին նվիրված սեմինարին [19]:

SAFER+ և SAFER++ բլոկային ծածկագրական համակարգերը պրոֆեսոր Ջեյմս Մեսիի և ՀՀ ԳԱԱ ԻԱՊԻ կոդավորման և ազդանշանների մշակման լաբորատորիայի աշխատակիցներ ՀՀ ԳԱԱ ակադեմիկոս Գուրգեն Խաչատրյանի ու ֆ.մ.գ.թ. Մելսիկ Կյուրեղյանի համագործակցության արդյունքներն են:

SAFER ընտանիքի նախորդ ծածկագրական համակարգերում դիֆուզիան (երբ չնչին փոփոխությունները ռաունդի մուտքում բերում են զգալի փոփոխությունների ռաունդի ելքում, ինչը թույլ է տալիս նվազեցնել ռաունդների քանակը, մեծացնելով

ծածկագրման արագագործությունը՝ միաժամանակ ապահովելով համակարգի կայունությունը դիֆերենցիալ վերլուծության նկատմամբ) ապահովում էր “Hadamard Shuffle” ձևափոխությունը [20], որն իրենից ներկայացնում է 8 բայթերի տեղադրություն: SAFER+ համակարգի կրիպտոկայունությունը դիֆերենցիալ վերլուծության նկատմամբ ուսումնասիրելու ժամանակ Գ. Խաչատրյանը և Մ. Կյուրեղյանը նկատել են, որ 16 բայթերի տեղադրության ճիշտ ընտրության դեպքում կարելի է ապահովել անհամեմատ ավելի լավ դիֆուզիա [21]: Այդ իսկ պատճառով Մեսին համակարգում օգտագործվող տեղադրությունը, որն ապահովում է հնարավոր լավագույն դիֆուզիան, անվանել է "Armenian Shuffle": SAFER+ համակարգը հանդիսանում է SAFER ընտանիքի էապես կատարելագործված տարբերակը: 1998թ. SAFER+ ալգորիթմը ներկայացվել է NIST (National Institute of Standards and Technology, US) կողմից կազմակերպված AES (Advanced Encryption Standard) մրցույթին [21], որտեղ հայտնվել է նախընտրական փուլում՝ 14 այլ ծածկագրման ալգորիթմների հետ միասին: Հետագայում համակարգը օգտագործվել է BLUTOOTH ստանդարտի ինքնության ճանաչման "message authentication codes called E1" և բանալու գեներացման "key derivation called E21 և E22" սխեմաներում [34],[37],[38]: Ավելի ուշ մշակված SAFER++ համակարգը [23] ևս կրիպտոկայուն է և ի տարբերություն SAFER+ համակարգի ավելի արագ, շնորհիվ պսևդո-Հադամարի 4-PHT ձևափոխության (նախորդ բոլոր ալգորիթմներում օգտագործվել է պսևդո–Հադամարի 2-PHT ձևափոխությունը): SAFER++ համակարգը մասնակցել է NESSIE (New European Schemes for Signature) մրցույթին [23] և ճանաչվել է 21-րդ դարի ազատ օգտագործման լավագույն բլոկային ծածկագրական համակարգերից մեկը:

Այս գլուխը նվիրված է հայտնի SAFER+ բլոկային ծածկագրական համակարգի հիման վրա մշակված, համարժեք կրիպտոկայունությամբ և ավելի մեծ արագագործությամբ 128 բիթային ծածկագրական համակարգի և SAFER ընտանիքի բլոկային ծածկագրական համակարգերի տրամաբանական շարունակություն հանդիսացող 256 բիթային ծածկագրման ալգորիթմի նկարագրությանը: Ներկայացված են գաղտնի բանալուց ենթաբանալիների գեներացման ալգորիթմները:

1.2. SAFER+ համակարգի նկարագրությունը

SAFER+ ալգորիթմի մուտքը և ելքը իրենցից ներկայացնում են 128 բիթերի հաջորդականություններ, այսինքն SAFER+ բլոկային ծածկագրական համակարգը ծածկագրում է 128 բիթ երկարության բաց տեքստ: Համակարգի գաղտնի բանալին, որն ընտրվում է օգտագործողի կողմից 128, 192 կամ 256 բիթ երկարության բիթերի հաջորդականություն է [21]:

SAFER+ ալգորիթմը բաղկացած է միևնույն կառուցվածքի r անգամ կիրառվող ֆունկցիայից, որն անվանում են ռաունդ և, որը բաղկացած է հինգ տարբեր հակադարձելի ձևափոխություններից.

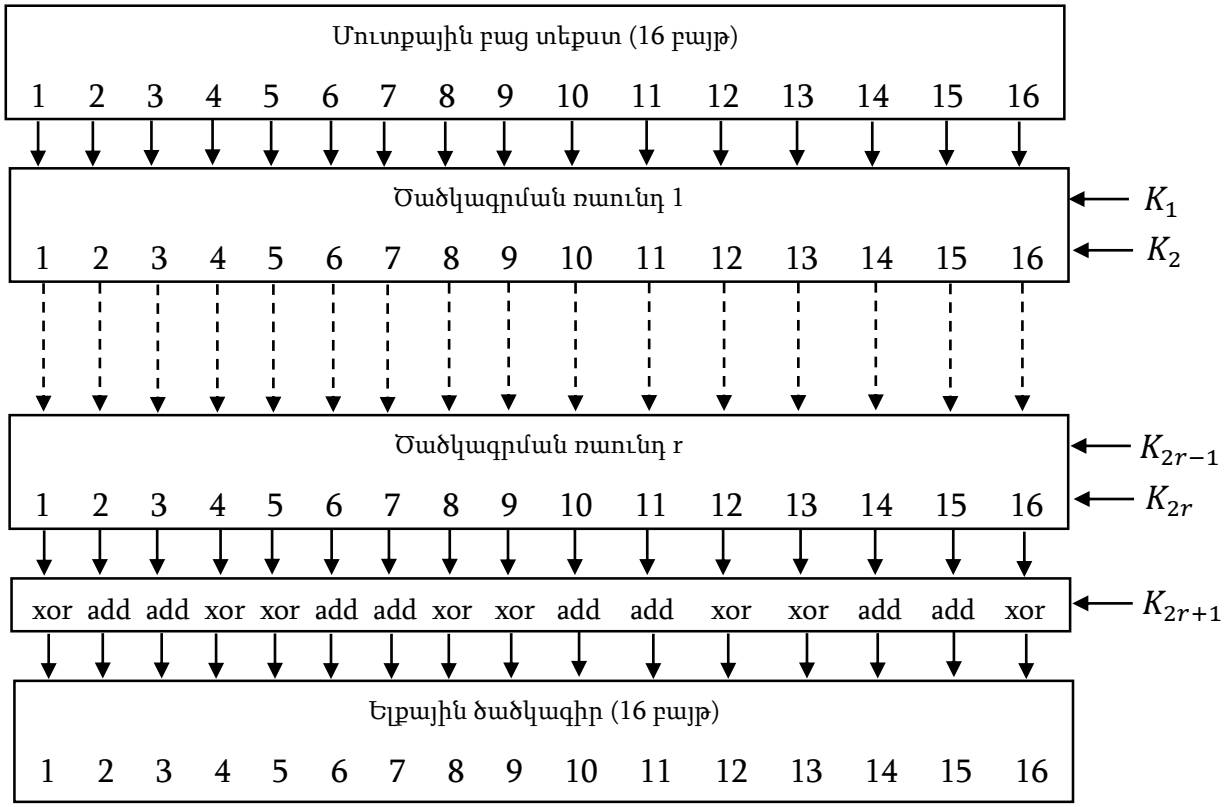
- $\mathrm{xor/add}$. առաջին ենթաբանալու և բլոկի բայթերի «գումարում»,
- $\exp/log$. ոչ գծային ֆունկցիաների կիրառում,
- $\mathrm{add/xor}$. երկրորդ ենթաբանալու և բլոկի բայթերի «գումարում»,
- $\mathrm{2\text{-}PHT}$. պսևդո-Հադամարի ձևափոխություն,
- $\mathrm{Armenian\ Shuffle}$. բլոկի բայթերի ֆիքսած տեղադրություն:

Ձևափոխություններից յուրաքանչյուրի մանրամասն նկարագրությունը բերված է 1.2.1 պարագրաֆում: Ռաունդների r քանակը կախված է բանալու երկարությունից և որոշվում է դիֆերենցիալ ու գծային վերլուծությունների նկատմամբ համակարգի կրիպտոկայունության մակարդակից, այսինքն քանի ռաունդ է հարկավոր, որպեսզի համակարգը լինի կրիպտոկայուն նշված երկու էֆեկտիվ հարձակումների նկատմամբ:

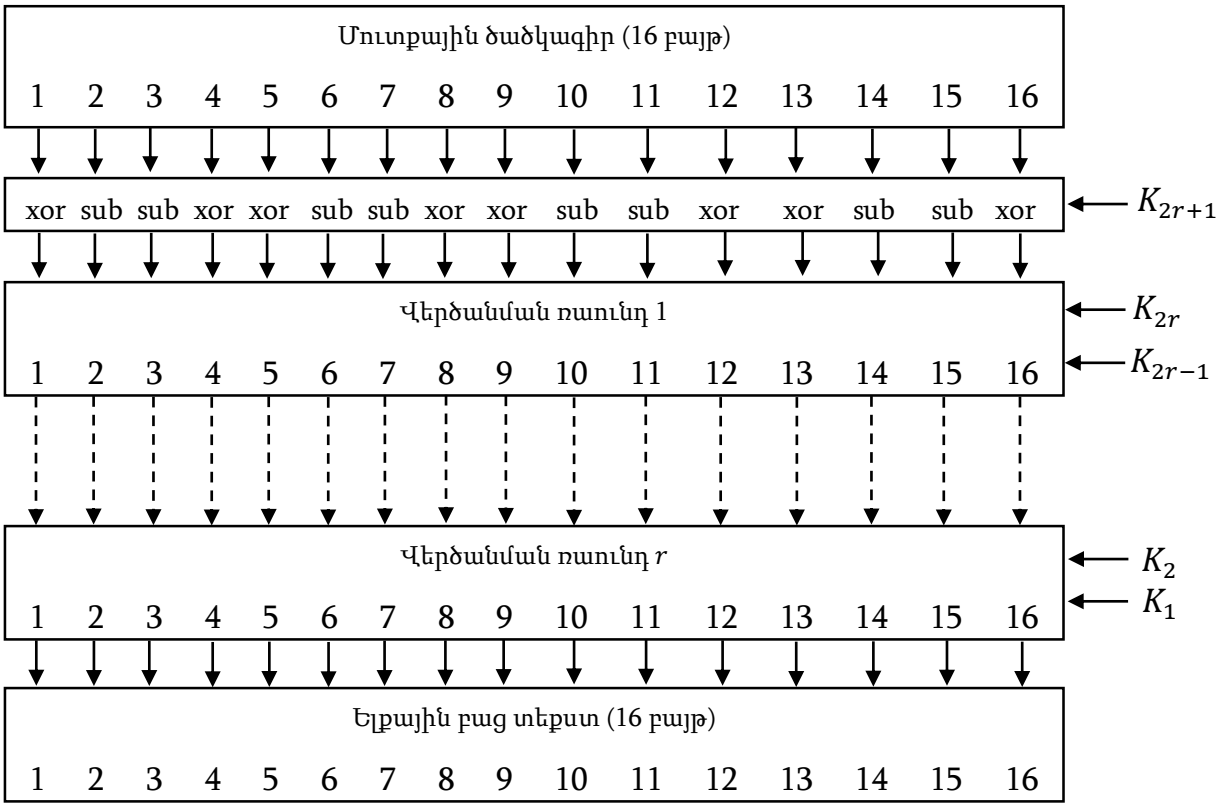
Նկար 1-ում պատկերված է SAFER+ բլոկային ծածկագրական համակարգի ծածկագրման ընդհանուր կառուցվածքը և ինչպես երևում է նկարից, ծածկագրման մուտքը 16 բայթ երկարության բաց տեքստն է (128 բիթերի բայթային ներկայացումը), որը անցնում է ծածկագրման r ռաունդների միջով. բաց տեքստի նկատմամբ կիրառվում է ռաունդ ֆունկցիան r անգամ: Ծածկագրման յուրաքանչյուր ռաունդում օգտագործվում են 16 բայթ երկարության երկու ենթաբանալի:

Համակարգի $(K_1, K_2, \dots, K_{2r+1})$ ենթաբանալիները որոշվում (գեներացվում) են K գաղտնի բանալու միջոցով, համաձայն SAFER+ համակարգի բանալիների գեներացման ալգորիթմի: 1.2.5 և 1.2.6 պարագրաֆներում ներկայացված են ենթաբանալիների գեներացման ալգորիթմները և սխեմաները բանալու համապատասխանաբար 128 և 256 բիթ երկարությունների դեպքում: Ենթաբանալիներից վերջինը՝ K_{2r+1}-ը, համարվում է լրացուցիչ բանալի, քանի որ «գումարվում» է ծածկագրման r ռաունդների ելքային 16 բայթ բլոկին: K_{2r+1} ենթաբանալու այս եղանակով «գումարմանը» անվանում են SAFER+ համակարգի ծածկագրման ելքային ձևափոխություն, քանի որ այս ձևափոխության արդյունքը 16 բայթ երկարության ծածկագիրն է (ծածկագրված տեքստը): Ինչպես երևում է նկար 1-ից, K_{2r+1} բանալին «գումարվում» է հետևյալ եղանակով. 1, 4, 5, 8, 9, 12, 13 և 16 բայթերը գումարվում են բիթ առ բիթ ըստ մոդուլ 2-ի (“exclusive-or” կամ "xor"), իսկ 2, 3, 6, 7, 10, 11, 14 և 15 բայթերը ըստ մոդուլ 256-ի:

Ծածկագրից բաց տեքստի վերծանման նպատակով՝ ծածկագրի նկատմամբ կիրառվում են ծածկագրման ալգորիթմի ձևափոխությունների հակադարձ ձևափոխությունները հակառակ հերթականությամբ (տե՛ս Նկար 2): Վերծանման մուտքը իրենից ներկայացնում է 16 բայթ երկարության ծածկագիրը: Վերծանումը սկսվում է մուտքային ձևափոխությունից, որը ծածկագրման ելքային ձևափոխության հակադարձ ձևափոխությունն է: Մուտքային ձևափոխության ժամանակ ենթաբանալիներից վերջին K_{2r+1} ենթաբանալին «հանվում» է ծածկագրված տեքստից հետևյալ եղանակով. ենթաբանալու 1, 4, 5, 8, 9, 12, 13 և 16 բայթերը գումարվում են մուտքային ծածկագրի համապատասխան բայթերին բիթ առ բիթ ըստ մոդուլ 2-ի, քանի որ գումարումը և հանումը ըստ մոդուլ 2-ի համընկնում են, իսկ ենթաբանալու 2, 3, 6, 7, 10, 11, 14 և 15 բայթերը հանվում են ծածկագրի համապատասխան բայթերից ըստ մոդուլ 256-ի: Արդյունքում ստացվում է 16 բայթ երկարության բլոկ, որը համընկնում է ծածկագրման r ռաունդների ելքային 16 բայթ երկարության բլոկի հետ՝ նախքան ելքային ձևափոխության կիրառումը: Այնուհետև ստացված բլոկը անցնում է վերծանման r ռաունդների միջով: Վերծանման i-րդ ռաունդը բաղկացած է ծածկագրման $(r - i + 1)$-րդ ռաունդի ձևափոխությունների հակադարձ ձևափոխություններից. $i = 1, 2, \dots, r$:



Նկար 1: SAFER+ ծածկագրման ալգորիթմի ընդհանուր կառուցվածքը:



Նկար 2: SAFER+ վերծանման ալգորիթմի ընդհանուր կառուցվածքը:

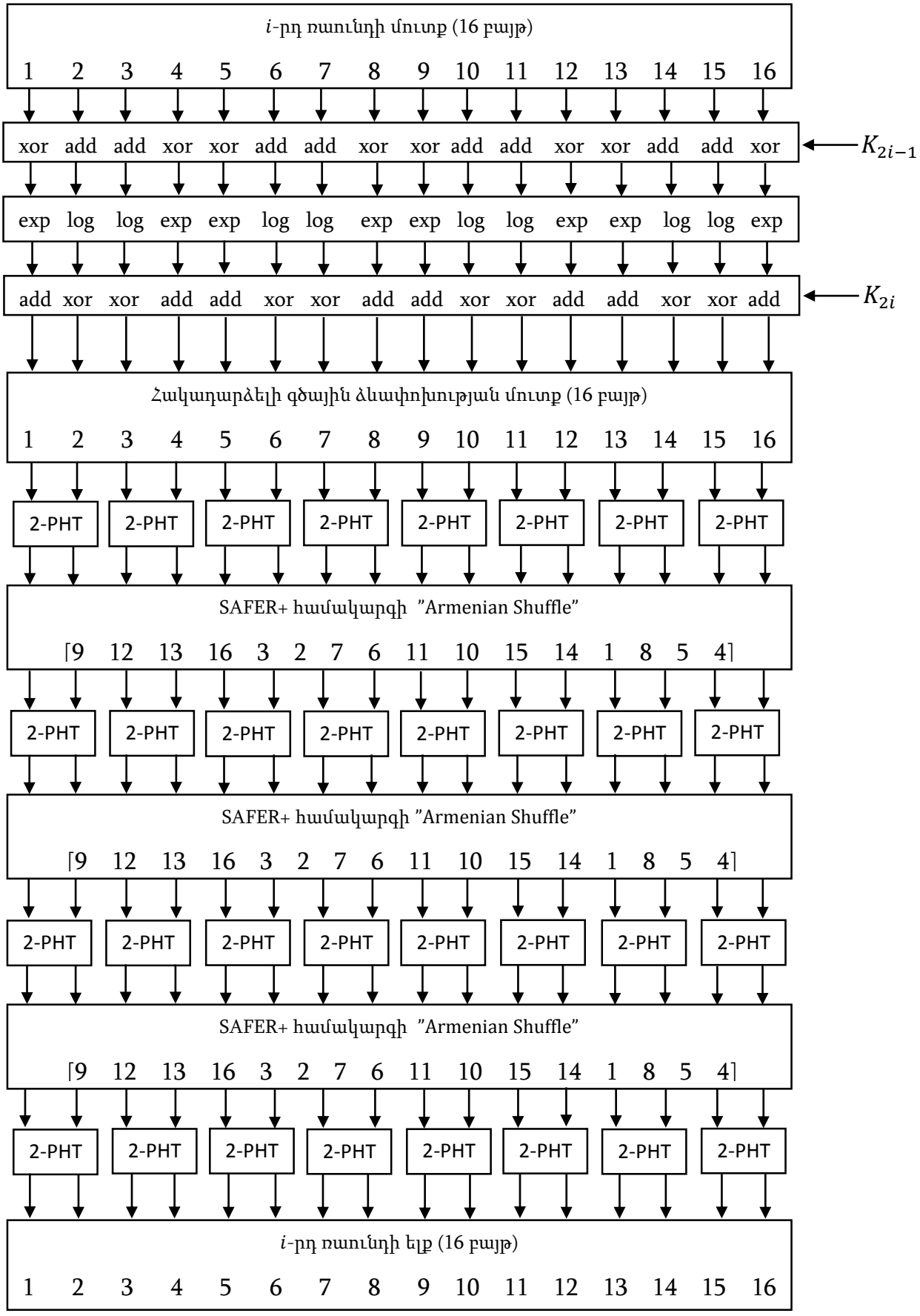
Հարկ է նշել, որ վերծանման i-րդ ռաունդի և ծածկագրման $(r-i+1)$-րդ ռաունդի ենթաբանալիները նույնն են, սակայն կիրառվում են հակառակ հերթականությամբ: Վերծանման r ռաունդների ելքում ստացվում է բաց տեքստը:

Ռաունդների r քանակը որոշվում է դիֆերենցիալ վերլուծության միջոցով (տե՛ս Գլուխ 2).

- $r = 6$, եթե բանալու երկարությունը 128 բիթ է,
- $r = 8$, եթե բանալու երկարությունը 192 բիթ է,
- $r = 9$, եթե բանալու երկարությունը 256 բիթ է:

1.2.1. SAFER+ համակարգի ծածկագրման ռաունդի նկարագրությունը

SAFER+ համակարգի ծածկագրման ռաունդի կառուցվածքային նկարագրությունը բերված է նկար 3-ում: i-րդ ռաունդի առաջին գործողությունը տվյալ ռաունդի 16 բայթ մուտքի և ռաունդի առաջին K_{2i-1} ենթաբանալու «գումարումն» է հետևյալ եղանակով. 1, 4, 5, 8, 9, 12, 13 և 16 բայթերը գումարվում են բիթ առ բիթ ըստ մոդուլ 2-ի, իսկ 2, 3, 6, 7, 10, 11, 14 և 15 բայթերը ըստ մոդուլ 256-ի: Հաջորդ ձևափոխության ժամանակ 16 բայթ երկարության բլոկը անցնում է ռաունդի ոչ գծային շերտով. $j = 1, 4, 5, 8, 9, 12, 13, 16$ բայթերի x արժեքների վրա կիրառվում է $45^x mod\ 257$ ցուցչային ֆունկցիան (պայմանով, որ երբ $x = 128$, ապա $45^{128} mod\ 257 = 256$ հավասար է 0-ի), իսկ $j = 2, 3, 6, 7, 10, 11, 14, 15$ բայթերի x արժեքների վրա $\log_{45} x$ լոգարիթմական ֆունկցիան (պայմանով, որ երբ $x = 0$, ապա $\log_{45} 0$ հավասար է 128-ի): Ոչ գծային շերտի ելքին «գումարվում» է ռաունդի երկրորդ K_{2i} ենթաբանալին հետևյալ եղանակով. 1, 4, 5, 8, 9, 12, 13 և 16 բայթերը գումարվում են ըստ մոդուլ 256-ի, իսկ 2, 3, 6, 7, 10, 11, 14 և 15 բայթերը՝ բիթ առ բիթ ըստ մոդուլ 2-ի: Արդյունքում ստացված 16 բայթերի վրա կիրառվում է ծածկագրման հակադարձելի գծային ձևափոխությունը (տե՛ս Նկար 3):

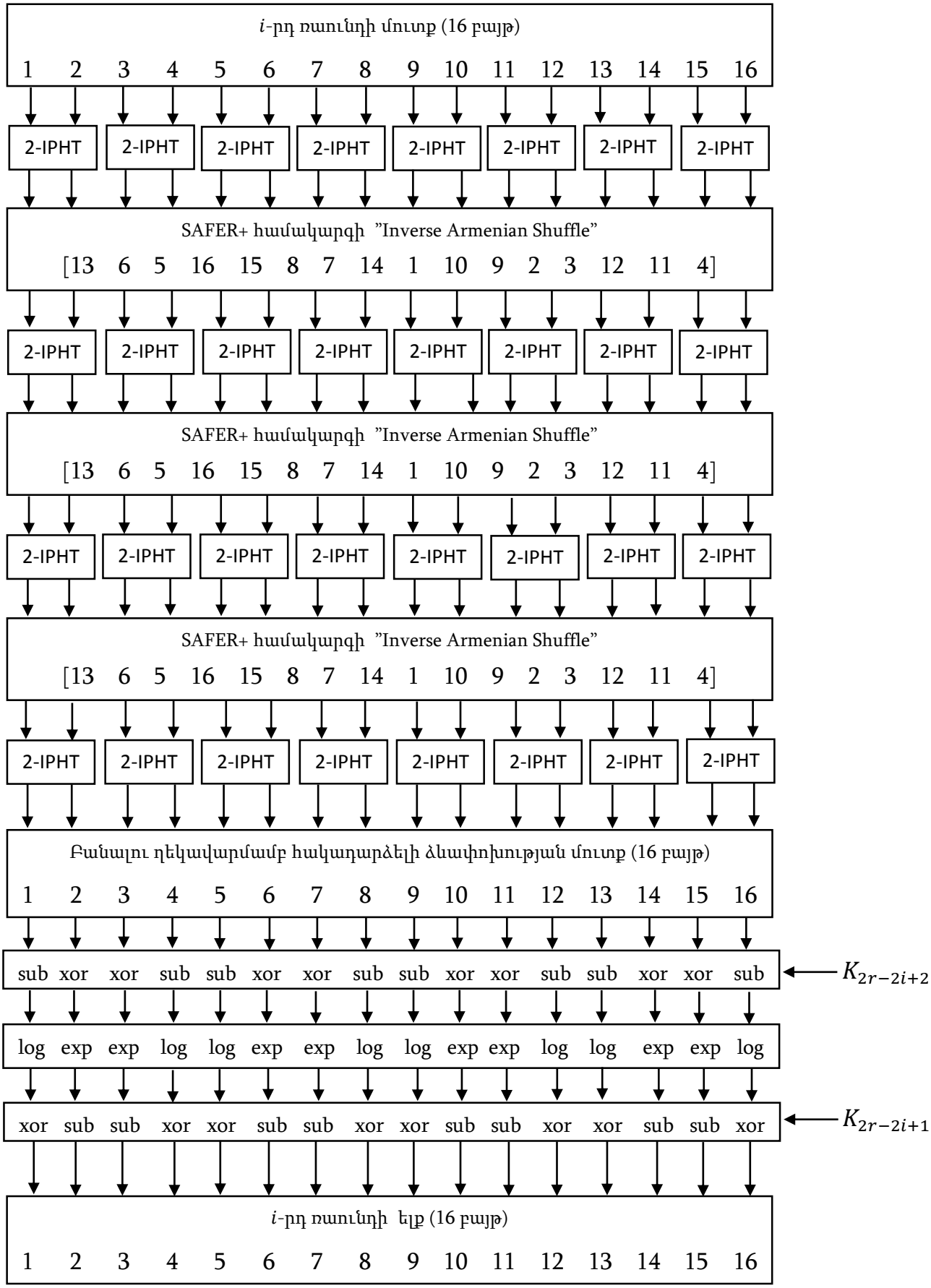


Նկար 3: SAFER+ համակարգի ծածկագրման i-րդ ռաունդի կառուցվածքը:

Ծածկագրման ռաունդի հակադարձելի գծային ձևափոխության գործողություններից մեկը 16 բայթերի [9, 12, 13, 16, 3, 2, 7, 6, 11, 10, 15, 14, 1, 8, 5, 4] տեղադրությունն է, որին անվանում են "Armenian Shuffle" և որի առաջին ելքային բայթը հանդիսանում է իններորդ մուտքային բայթը, երկրորդ ելքային բայթը՝ տասներկուերորդ մուտքային բայթը, երրորդ ելքայի բայթը՝ տասներեքերորդ մուտքային բայթը, իսկ վերջին տասնվեցերորդ ելքային բայթը հանդիսանում է չորրորդ մուտքային բայթը: Հաջորդ գործողությունը Մեսսին անվանել է պսևդո-Հադամարի ձևափոխություն (Pseudo-Hadamard Transform), որը կատարվում է $H = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$ մատրիցի միջոցով. 2-PHT: 2-PHT ձևափոխությունը երկու հաջորդական բայթից բաղկացած (a_1, a_2) մուտքը փոխակերպում է երկու բայթից կազմված (A_1, A_2) ելքին հետևյալ կերպ՝ $(A_1, A_2) = (a_1, a_2) \cdot H = (2a_1 + a_2, a_1 + a_2)$, որտեղ թվաբանությունը ըստ մոդուլ 256-ի է (16 բայթի համար 8 հատ 2-PHT ձևափոխություն): Այսպիսով, ինչպես երևում է նկար 3-ից, ռաունդի հակադարձելի գծային ձևափոխությունը բաղկացած է 4 անգամ կիրառվող 8 հատ 2-PHT ձևափոխությունից և 3 անգամ կիրառվող բայթերի Armenian Shuffle տեղադրությունից, որոնք կիրառվում են մեկընդմեջ:

1.2.2. SAFER+ համակարգի վերծանման ռաունդի նկարագրությունը

Նկար 4-ում սխեմատիկորեն պատկերված է SAFER+ համակարգի վերծանման ռաունդի կառուցվածքը: Վերծանման ռաունդի ձևափոխությունները ծածկագրման ռաունդի ձևափոխությունների հակադարձ ձևափոխություններն են, որոնք կիրառվում են հակառակ հերթականությամբ: Հետևաբար տրամաբանական է, որ վերծանման ռաունդում իրականացվող առաջին գործողությունը 2-PHT ձևափոխության հակադարձ 2-IPHT ձևափոխությունն է (Inverse Pseudo-Hadamard Transform), որը կիրառվում է ռաունդի 16 բայթ երկարության մուտքի երկու իրար հաջորդող բայթերի նկատմամբ (16 բայթի համար 8 հատ 2-IPHT ձևափոխություն):



Նկար 4: SAFER+ համակարգի վերծանման i-րդ ռաունդի կառուցվածքը:

2-IPHT գծային ձևափոխությունը իրականացվում է 2-PHT ձևափոխության H մատրիցի $H^{-1} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}$ հակադարձ մատրիցի միջոցով: 2-IPHT ձևափոխությունը երկու բայթից բաղկացած (A_1, A_2) մուտքը փոխակերպում է երկու բայթից բաղկացած (a_1, a_2) ելքին. $(a_1, a_2) = (A_1, A_2) \cdot H^{-1} = (A_1 - A_2, -A_1 + 2A_2)$, որտեղ թվաբանությունը ըստ մոդուլ 256-ի է: Հաջորդ գործողությունը բայթերի Armenian Shuffle տեղադրության հակադարձ տեղադրությունն է՝ [13, 6, 5, 16, 15, 8, 7, 14, 1, 10, 9, 2, 3, 12, 11, 4]: Այս երկու գործողությունները նկարագրված հաջորդականությամբ կիրառվում են երեք անգամ և վերջում մեկ անգամ ևս կիրառվում է 8 հատ 2-IPHT գծային ձևափոխությունը (տե՛ս Նկար 4): Վերծանման i-րդ ռաունդի գծային ձևափոխության ելքային 16 բայթ արդյունքից, որը համընկնում է ծածկագրման $(r - i + 1)$-րդ ռաունդի գծային ձևափոխության մուտքի հետ, «հանվում» է ծածկագրման ռաունդում կիրառված երկրորդ $K_{2r-2i+2}$ ենթաբանալին հետևյալ եղանակով. ենթաբանալու 1, 4, 5, 8, 9, 12, 13, 16 բայթերը հանվում են 16 բայթ բլոկի համապատասխան բայթերից ըստ մոդուլ 256-ի, իսկ 16 բայթ մուտքի և ենթաբանալու 2, 3, 6, 7, 10, 11, 14, 15 բայթերը գումարվում են բիթ առ բիթ ըստ մոդուլ 2-ի: Ստացված 16 բայթ երկարության արդյունքը անցնում է վերծանման ոչ գծային շերտով. $j = 1, 4, 5, 8, 9, 12, 13, 16$ բայթերի x արժեքների վրա կիրառվում է $\log_{45} x$ լոգարիթմական ֆունկցիան (ընդ որում, եթե $x = 0$, $\log_{45} 0$ հավասար է 128-ի), իսկ $j = 2, 3, 6, 7, 10, 11, 14, 15$ բայթերի x արժեքների վրա՝ $45^x \bmod 257$ ցուցչային ֆունկցիան (ընդ որում, եթե $x = 128$, ապա $45^{128} \bmod 257 = 256$ հավասար է 0-ի): Ռաունդի վերջում ծածկագրման $(r - i + 1)$-րդ ռաունդի առաջին $K_{2r-2i+1}$ բանալին «հանվում» է 16 բայթ երկարության բլոկից. ենթաբանալու 1, 4, 5, 8, 9, 12, 13, 16 բայթերը բիթ առ բիթ ըստ մոդուլ 2-ի գումարվում են մուտքային բլոկի համապատասխան բայթերին, իսկ 2, 3, 6, 7, 10, 11, 14, 15 բայթերը ըստ մոդուլ 256-ի հանվում են համապատասխան մուտքային բայթերից՝ ձևավորելով վերծանման տվյալ ռաունդի 16 բայթ երկարության ելքը:

1.2.3. SAFER+ համակարգի ենթաբանալինները

r ռաունդից կազմված SAFER+ համակարգում օգտագործվում են $2r+1$ հատ 16 բայթ երկարության ենթաբանալի, քանի որ յուրաքանչյուր ռաունդում 16 բայթ երկարության բլոկին «գումարվում» է երկու ենթաբանալի, իսկ ելքային ձևափոխության ժամանակ ռաունդների ելքին ավելացվում է մեկ ենթաբանալի: Այդ ենթաբանալիները գեներացվում են օգտագործողի կողմից ընտրված 128, 192 կամ 256 բիթ երկարության գաղտնի բանալուց՝ SAFER+ համակարգի ենթաբանալիների գեներացման համապատասխան ալգորիթմի միջոցով [21]: Ենթաբանալիների գեներացման ալգորիթմների մանրամասն նկարագրություններն ու սխեմաները 128 և 256 բիթ բանալիների դեպքում բերված են 1.2.5 և 1.2.6 պարագրաֆներում:

Ինչպես արդեն նշվել է նախորդ պարագրաֆներում ռաունդների r քանակը, հետևաբար նաև ենթաբանալիների քանակը կախված է օգտագործողի կողմից ընտրված բանալու երկարությունից.

- եթե բանալու երկարությունը 128 բիթ է, ապա $r=6$,
- եթե բանալու երկարությունը 192 բիթ է, ապա $r=8$,
- եթե բանալու երկարությունը 256 բիթ է, ապա $r=9$:

Եվս մեկ անգամ նշենք, որ ծածկագրումը և վերծանումը կատարվում են միևնույն գաղտնի բանալու միջոցով: Բաց տեքստը ծածկագրելու և այդ ծածկագրից բաց տեքստը վերականգնելու համար օգտագործվում է գաղտնի բանալուց գեներացված ենթաբանալիների միևնույն բազմությունը:

Ծածկագրումը և վերծանումը 128 բիթային բանալու դեպքում կատարվում է 13 ենթաբանալիների միջոցով. $K_1, K_2, \dots, K_{13}$, այսինքն անհրաժեշտ է գեներացնել 13 ենթաբանալի, 192 բիթային բանալու դեպքում անհրաժեշտ է գեներացնել 17 ենթաբանալի. $K_1, K_2, \dots, K_{17}$, իսկ 256 բիթ երկարության բանալու դեպքում՝ 19 ենթաբանալի. $K_1, K_2, \dots, K_{19}$:

1.2.4. Շեղումնային բառերը և նրանց որոշումը

SAFER+ համակարգի ենթաբանալիների գեներացման ալգորիթմում օգտագործվում են *շեղումնային բառեր* (bias words)՝ ենթաբանալիների կամայական բաշխումն ապահովելու համար [21]: Շեղումնայի բառերի քանակը հավասար է ենթաբանալիների քանակին, այսինքն 13՝ 128-բիթ բանալու դեպքում, 17՝ 192-բիթ բանալու դեպքում և 19՝ 256-բիթ բանալու դեպքում:

i-րդ շեղումնային բառը նշանակենք B_i-ով, իսկ $B_{i,j}$-ով՝ B_i շեղումնային բառի j-րդ բայթը, որտեղ $i = 1, 2, \dots, 2r + 1$, $j = 1, 2, \dots, 16$: B_1 շեղումնային բառը կարելի է համարել «պարապ» այն իմաստով, որ ենթաբանալիների գեներացման ալգորիթմում գործնականորեն չի օգտագործվում, այն ներմուծվում է ալգորիթմի ծրագրային իրականացման համար: $B_2, B_3, \dots, B_{13}$ շեղումնայի բառերի շեղումնային բայթերը որոշվում են հետևյալ եղանակով.

$$B_{i,j} = 45^{(45^{17i+j} mod 257)} mod 257, \text{ որտեղ } i = 2, 3, \dots, 13 \text{ և } j = 1, 2, \dots, 16,$$

ընդ որում, եթե $B_{i,j}$-ն հավասար է 256-ի, ապա այն ընդունում է 0 արժեք: 192 բիթ երկարությամբ բանալու դեպքում անհրաժեշտ է 4 շեղումնային բառ ևս, իսկ 256 բիթ եկարությամբ բանալու դեպքում 2 շեղումնային բառ ևս: Իսկ այդ 6 շեղումնային բառերի շեղումնային բայթերը որոշվում են այսպես.

$$B_{i,j} = 45^{45^{17i+j}} mod 257, \text{ որտեղ } i = 14, 15, \dots, 19 \text{ և } j = 1, 2, \dots, 16,$$

ընդ որում $B_{i,j}$-ն ընդունում է 0 արժեք, եթե վերջինս հավասար է 256-ի: Աղյուսակ 1-ում ներկայացված է SAFER+ համակարգի ենթաբանալիների գեներացման 19 շեղումնային բառերը: Աղյուսակի առաջին տողը իրենից ներկայացնում է B_2 շեղումնային բառը, երկրորդ տողը՝ B_3 շեղումնային բառը, իսկ վերջին տողը B_{19} շեղումնային բառն է:

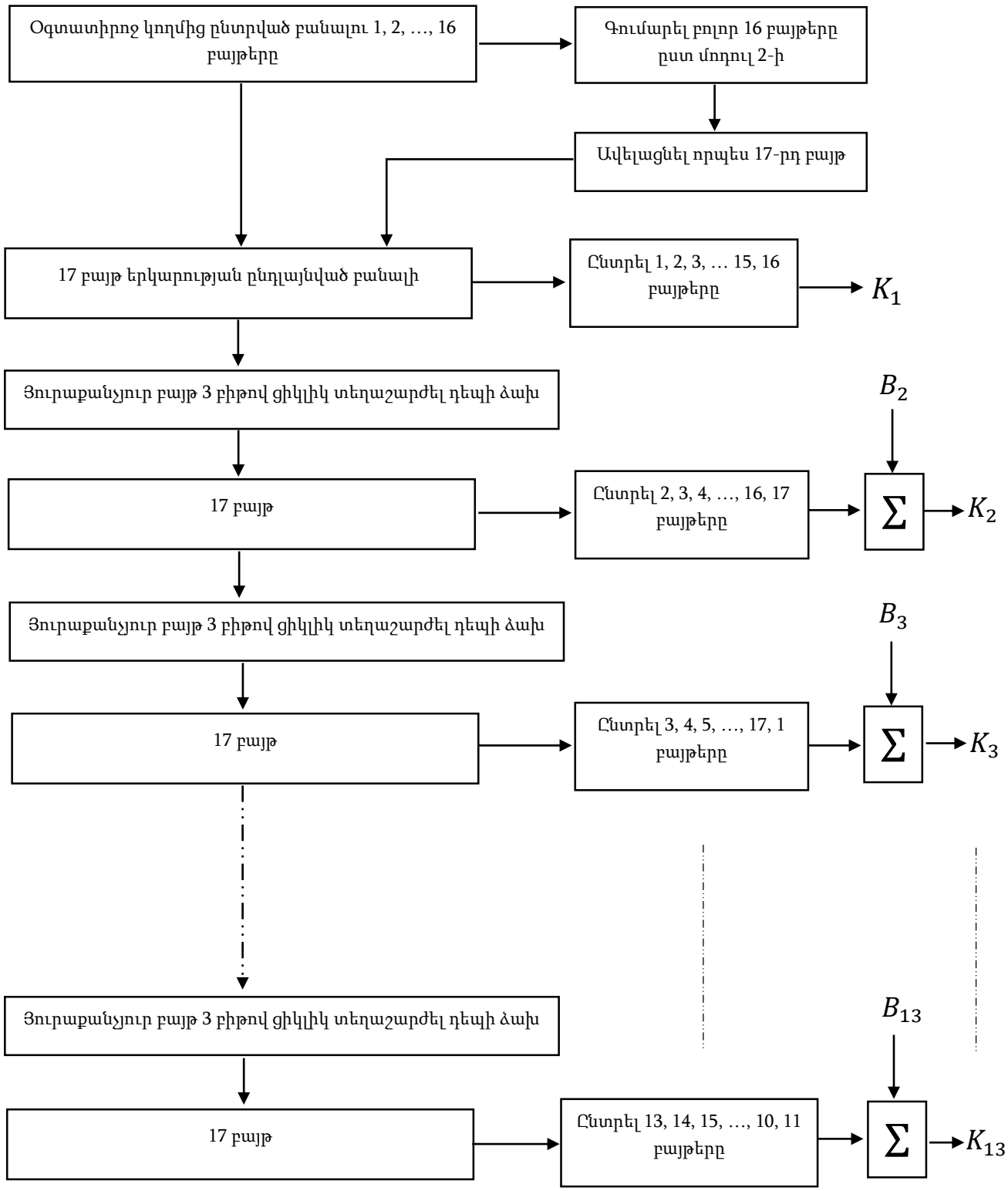
Հաջորդ երկու պարագրաֆներում [21] աշխատանքում բերված SAFER+ համակարգի ենթաբանալիների գեներացման ալգորիթմների նկարագրություն է և սխեմաները՝ օգտագործողի կողմից ընտրված 128 և 256 բիթ երկարության բանալիների դեպքում:

Աղյուսակ 1: SAFER+ համակարգի ենթաբանալիների գեներացման $B_2, B_3, \ldots, B_{19}$ շեղումնային բառերը:

$B_2 =$	70	151	177	186	163	183	16	10	197	55	179	201	90	40	172	100
$B_3 =$	236	171	170	198	103	149	88	13	248	154	246	110	102	220	5	61
$B_4 =$	138	195	216	137	106	233	54	73	67	191	235	212	150	155	104	160
$B_5 =$	93	87	146	31	213	113	92	187	34	193	190	123	188	153	99	148
$B_6 =$	42	97	184	52	50	25	253	251	23	64	230	81	29	65	68	143
$B_7 =$	221	4	128	222	231	49	214	127	7	162	247	57	218	111	35	202
$B_8 =$	58	208	28	209	48	62	18	161	205	15	224	168	175	130	89	44
$B_9 =$	125	173	178	239	194	135	206	117	6	19	2	144	79	46	114	51
$B_{10} =$	192	141	207	169	129	226	196	39	47	108	122	159	82	225	21	56
$B_{11} =$	252	32	66	199	8	228	9	85	94	140	20	118	96	255	223	215
$B_{12} =$	250	11	33	0	26	249	166	185	232	158	98	76	217	145	80	210
$B_{13} =$	24	180	7	132	234	91	164	200	14	203	72	105	75	78	156	53
$B_{14} =$	69	77	84	229	37	60	12	74	139	63	204	169	219	107	174	244
$B_{15} =$	45	243	124	109	157	181	38	116	242	147	83	176	240	17	237	131
$B_{16} =$	182	3	22	115	59	30	142	112	189	134	27	71	126	36	86	241
$B_{17} =$	136	70	151	177	186	163	183	16	10	197	55	179	201	90	40	172
$B_{18} =$	220	134	119	215	166	17	251	244	186	146	145	100	131	241	51	239
$B_{19} =$	44	181	178	43	136	209	153	203	140	132	29	20	129	151	113	202

1.2.5. Ենթաբանալիների գեներացման մեթոդը 128-բիթ բանալու դեպքում

128 բիթ երկարության գաղտնի բանալու դեպքում 6 ռաունդների և ելքային ձևափոխության համար անհրաժեշտ 13 ենթաբանալիները գեներացվում են հետևյալ եղանակով: Օգտագործողի կողմից ընտրված 16 բայթ (128 բիթ) երկարության բանալին ընդունվում է որպես առաջին ենթաբանալի՝ K_1: Ստեղծվում է 17-բայթ բանալու ռեգիստր, որի առաջին 16 բայթերը օգտագործողի կողմից ընտրված բանալու 16 բայթերն են, իսկ վերջին 17-րդ բայթը այդ 16 բայթերի բիթ առ բիթ ըստ մոդուլ 2-ի գումարը (տե՛ս Նկար 5): Բանալու ռեգիստրի յուրաքանչյուր բայթ 3 բիթով ցիկլիկ տեղաշարժվում է դեպի ձախ և աղյուսակ 1-ի առաջին տողի 16 բայթ երկարության B_2 շեղումնային բառի բայթերը գումարվում են ռեգիստրի 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17 բայթերին ըստ մոդուլ 256-ի՝ ձևավորելով երկրորդ K_2 ենթաբանալին: Այնուհետև բանալու ռեգիստրի բայթերը նորից 3 բիթով ցիկլիկ տեղաշարժվում են դեպի ձախ և K_3 ենթաբանալին ձևավորելու նպատակով աղյուսակի 1-ի B_3 շեղումնային բառի 16 բայթերը ըստ մոդուլ 256-ի գումարվում են ռեգիստրի 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 1 բայթերին: Բանալու ռեգիստրի բայթերի 3 բիթով դեպի ձախ ցիկլիկ տեղաշարժի, բանալու ռեգիստրի բայթերից ընտրված 16 բայթերի և 16 բայթ երկարության շեղումնային բառի ըստ մոդուլ 256-ի գումարումը շարունակվում է մինչև ստացվում են բոլոր ենթաբանալիները: Վերջին K_{13} ենթաբանալին ձևավորվում է B_{13} շեղումնային բառի և բանալու ռեգիստրի 13, 14, 15, 16, 17, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 բայթերի ըստ մոդուլ 256-ի գումարման արդյունքում (K_{12} ենթաբանալին որոշելուց հետո բանալու ռեգիստրի բայթերը 3 բիթով դեպի ձախ ցիկլիկ տեղաշարժ կատարելուց հետո): Այս բոլոր քայլերը սխեմատիկորեն պատկերված են նկար 5-ում:

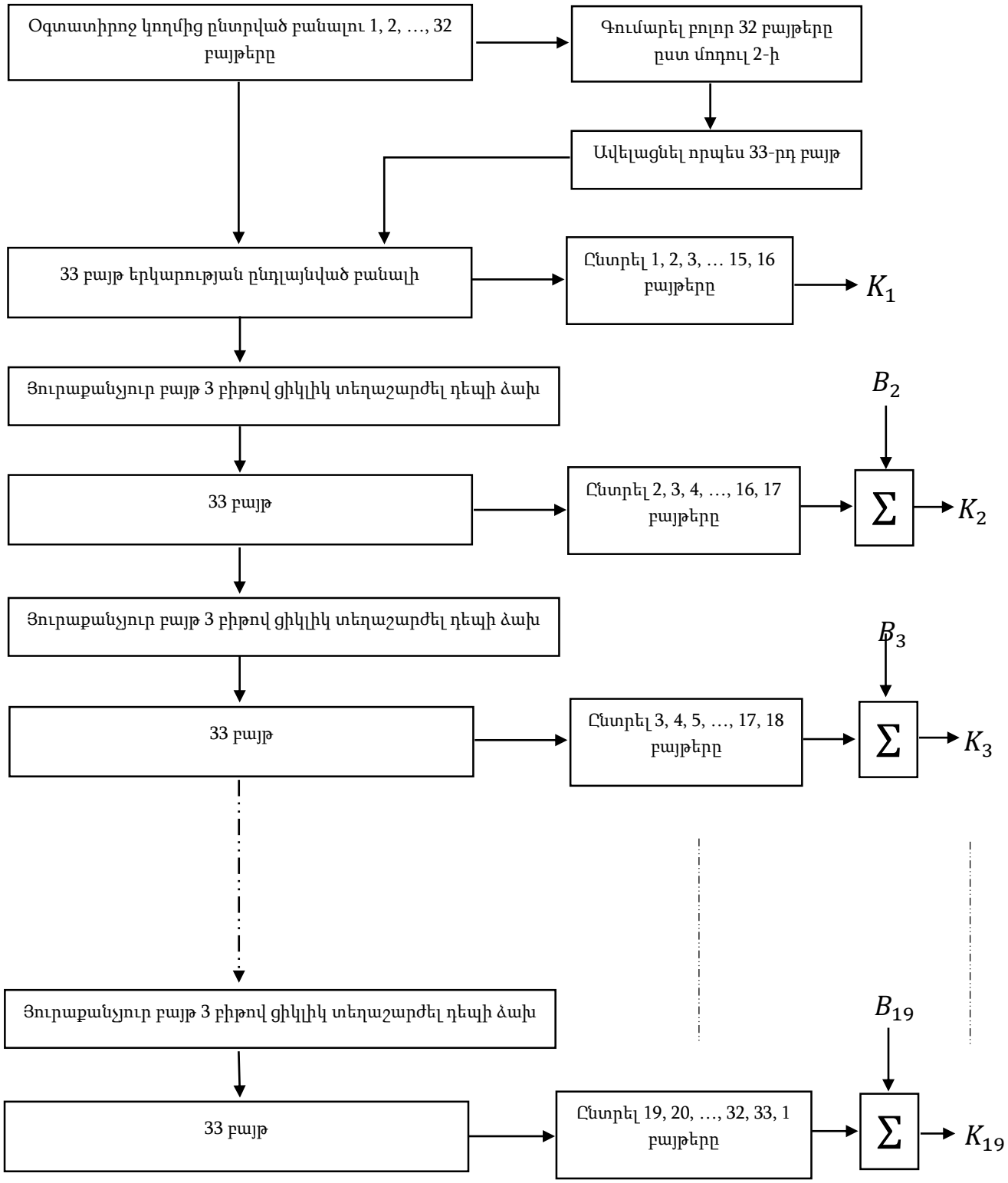


Նկար 5: SAFER+ համակարգի ենթաբանալիների գեներացման սխեման 128 բիթ երկարության բանալու դեպքում:

1.2.6. Ենթաբանալիների գեներացման մեթոդը 256-բիթ բանալու դեպքում

256 բիթ երկարության գաղտնի բանալու դեպքում անհրաժեշտ ենթաբանալիների թիվը 19 է, որոնցից 18-ը համակարգ են ներմուծվում 9 անգամ կիրառվող ռաունդ ֆունցիայի միջոցով, իսկ մեկը ելքային ձևափոխության ժամանակ: Նկար 6-ում պատկերված է ենթաբանալիների գեներացման ալգորիթմը սխեմատիկորեն:

Օգտագործողի կողմից ընտրված 32 բայթ (256 բիթ) երկարության բանալու առաջին 16 բայթերը դիտարկվում են որպես առաջին ենթաբանալի՝ K_1: Ստեղծվում է 33-բայթ բանալու ռեգիստր, որի առաջին 32 բայթերը օգտագործողի կողմից ընտրված բանալու 32 բայթերն են, իսկ վերջին 33-րդ բայթը այդ 32 բայթերի բիթ առ բիթ ըստ մոդուլ 2-ի գումարն է (տե՛ս Նկար 6): Բանալու ռեգիստրի յուրաքանչյուր բայթ 3 բիթով ցիկլիկ տեղաշարժվում է դեպի ձախ և աղյուսակ 1-ի առաջին տողի 16 բայթ երկարության B_2 շեղումնային բառի բայթերը գումարվում են ռեգիստրի 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17 բայթերին ըստ մոդուլ 256-ի՝ ձևավորելով երկրորդ K_2 ենթաբանալին: Այնուհետև բանալու ռեգիստրի բայթերը նորից 3 բիթով ցիլիկ տեղաշարժվում են դեպի ձախ և K_3 ենթաբանալին ձևավորելու նպատակով աղյուսակի 1-ի 2-րդ տողի B_3 շեղումնային բառի 16 բայթերը ըստ մոդուլ 256-ի գումարվում են ռեգիստրի 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18 բայթերին: Բանալու ռեգիստրի բայթերի 3 բիթ տեղաշարժի, բանալու ռեգիստրից (սահմանված կանոնակարգով) վերցված 16 բայթերի և շեղումնային բառի բայթերի ըստ մոդուլ 256-ի գումարումը շարունակվում է մինչև ստացվում են 19 ենթաբանալիները: Վերջին K_{19} ենթաբանալին ձևավորվում է B_{19} շեղումնային բառի 16 բայթերի և ռեգիստրի 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 1 բայթերի ըստ մոդուլ 256-ի գումարման արդյունքում (K_{12} ենթաբանալին որոշելուց հետո բանալու ռեգիստրի բայթերը 3 բիթով դեպի ձախ ցիկլիկ տեղաշարժ կատարելուց հետո):



Նկար 6: SAFER+ համակարգի ենթաբանալիների գեներացման սխեման 256 բիթ երկարության բանալու դեպքում:

1.3. SAFER+ համակարգի ձևափոխությունները

Ատենախոսության նպատակներից մեկը սիմետրիկ ծածկագրաբանության SAFER ընտանիքի բլոկային ծածկագրական համակարգերի հիման վրա ավելի արագագործ 128 բիթային ծածկագրական համակարգի մշակումն էր: Մասնավորապես SAFER+ համակարգի համար առանձնացնել ձևափոխությունների հաջորդականություն, որը կբարելավվի համակարգի արագագործությունը՝ թեկուզ ռաունդների քանակի կրճատման միջոցով: Դիֆերենցիալ վերլուծությամբ (տե՛ս Գլուխ 2) կատարված հետազոտությունները ցույց տվեցին, որ կառուցվածքային ձևափոխությունների և ոչ մի հաջորդականության միջոցով հնարավոր չի կրճատել SAFER+ համակարգի ռաունդների քանակը, որը կհանգեցներ համակարգի արագագործության բարելավմանը:

Ժամանակակից պրոցեսորների առանձնահատկությունները հնարավորություն են տալիս ծածկագրումը կատարել ավելի արագ, եթե ծածկագրական համակարգի ստրուկտուրան թույլ է տալիս օգտվել այդ առանձնահատկություններից: Մասնավորապես 32-բիթ ARM (Advanced RISC Machine) պրոցեսորները հնարավորություն են տալիս միաժամանակ xor անել 4 հաջորդական բայթեր (32 հաջորդական բիթեր), այսինքն՝ մեծացնել համակարգի արագագործությունը գործողությունների կրճատման հաշվին: Ինչպես երևում է նկար 1-ից և 3-ից SAFER+ համակարգի բիթային xor գործողությունը նախատեսված է 1 կամ 2 հաջորդական բայթերի համար: Բանալիների ներմուծման շերտերում երկու գործողությունների կիրառման հաջորդականությունների փոփոխությունից հետո SAFER ընտանիքի համակարգերի տրամաբանական հիմնավորումը պահպանելու համար կատարվել են ձևափոխություններ ցուցչային/լոգարիթմական ֆունկցիաների կիրառման ոչ գծային շերտում (տե՛ս Պարագրաֆ 1.5), իսկ համակարգի կրիպտոկայունությունը պահպանելու նպատակով (այսինքն՝ համակարգը ձևափոխություններից հետո ունենա գոնե համարժեք կրիպտոկայունություն) դիֆերենցիալ վերլուծությամբ մշակվել են ձևափոխություններ համակարգի հակադարձելի գծային ձևափոխության շերտում:

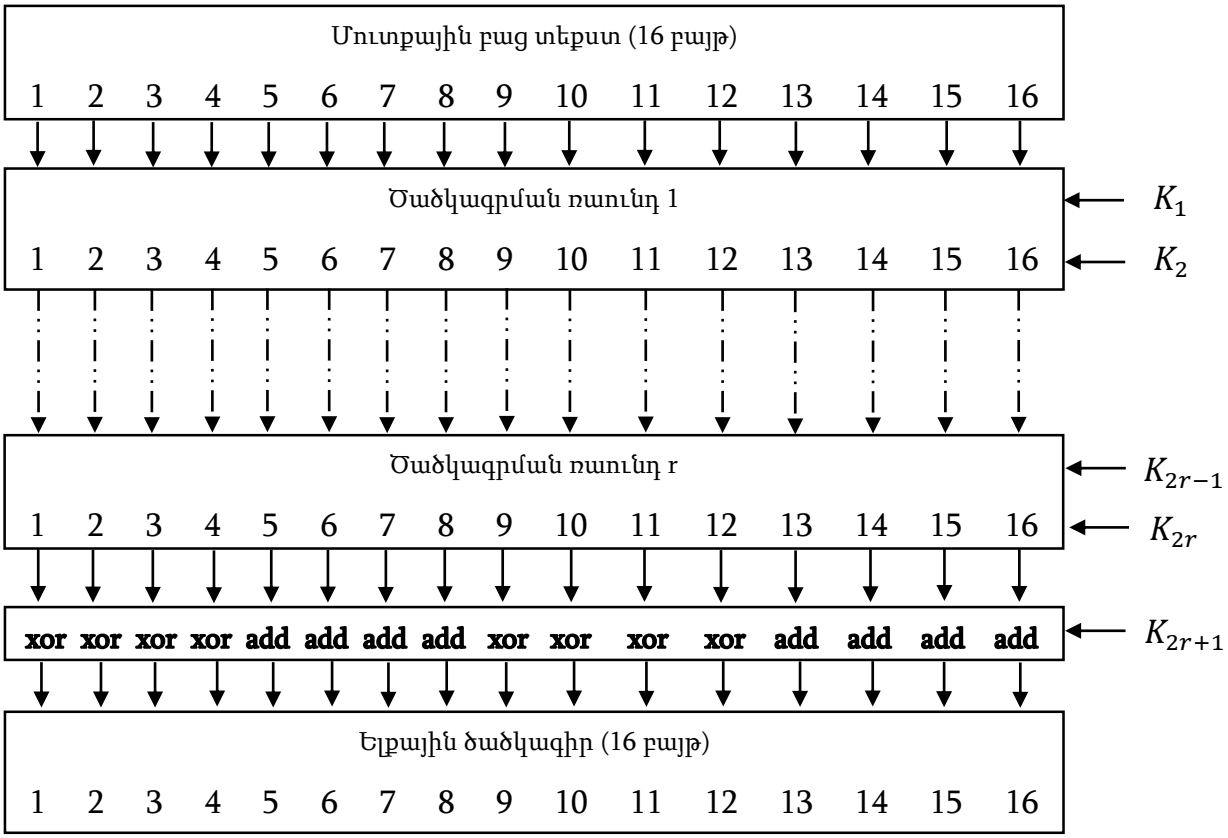
Չնայած որ SAFER+ համակարգը ձևափոխություններից հետո դիֆերենցիալ վերլուծության նկատմամբ կայուն են նորից նվազագույնը 5 ռաունդի դեպքում,

ձևափոխած SAFER+ համակարգի դիֆերենցիալ շղթաների անցումային հավանականությունները շատ ավելի փոքր են 2^{-128}-ից (դիֆերենցիալ շղթաների մասին մանրամասն ներկայացված է Գլուխ 2-ում): Դիֆերենցիալ վերլուծության նկատմամբ ձևափոխած SAFER+ համակարգի այս առավելությունից ելնելով աշխատանքում ներկայացված 256 բիթային համակարգի մշակման հիմք է ընդունվել ոչ թե SAFER+ ալգորիթմի, այլ SAFER+ համակարգի արագացված ալգորիթմի (ձևափոխած SAFER+ համակարգի) առանձնահատկությունները: Իսկ դիֆերենցիալ վերլուծության միջոցով կատարված հետազոտությունները ցույց են տվել, որ կատարված ընտրությունը ճիշտ է, քանի որ SAFER+ ալգորիթմի հիման վրա մշակված 256 բիթային համակարգը դիֆերենցիալ վերլուծության նկատմամբ կրիպտոկայուն է նվազագույնը 6 ռաունդի դեպքում, իսկ ձևափոխած SAFER+ ալգորիթմի հիման վրա մշակված 256 բիթային համակարգը՝ նվազագույնը 5 ռաունդի դեպքում:

1.3.1. Ձևափոխած SAFER+ համակարգը

Այս պարագրաֆում ներկայացված են SAFER+ բլոկային ծածկագրական համակարգի ձևափոխությունները, որոնք նպաստում են համակարգի արագագործության բարելավմանը՝ չդարձնելով համակարգի կայունությունը խոցելի էֆեկտիվ հարձակումների՝ դիֆերենցիալ և գծային վերլուծությունների նկատմամբ: Նկար 7-ում և նկար 8-ում սխեմատիկորեն պատկերված են ձևափոխած SAFER+ համակարգի համապատասխանաբար ծածկագրման ընդհանուր և ծածկագրման i-րդ ռաունդի կառուցվածքները, որտեղ ձևափոխությունները ընդգծված են մուգ գույնով: Ձևափոխությունները կատարվել են SAFER+ համակարգի ոչ գծային և գծային մասերում (տե՛ս Նկար 3, 8).

- Առաջարկվել է $\exp$ ֆունկցիան օգտագործել ոչ թե 1, 4, 5, 8, 9, 12, 13 և 16 բայթերի համար այլ 1, 2, 3, 4, 9, 10, 11 և 12 բայթերի համար, իսկ $\log$ ֆունկցիան 2, 3, 6, 7, 10, 11, 14 և 15 բայթերի փոխարեն 5, 6, 7, 8, 13, 14, 15 և 16 բայթերի համար:

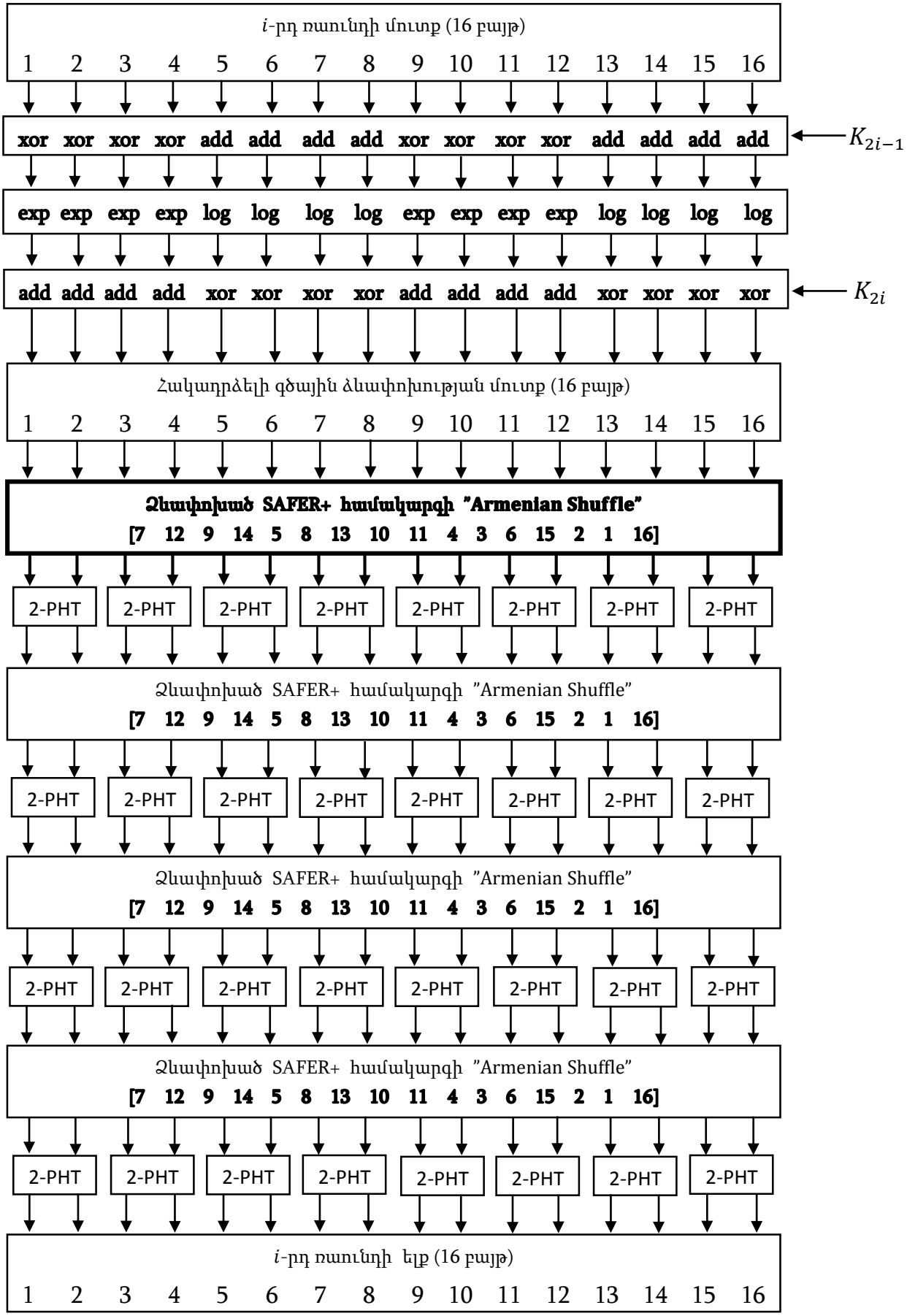


Նկար 7: SAFER+ համակարգի ծածկագրման ընդհանուր կառուցվածքը՝ ձևափոխություններից հետո: (Ձևափոխած SAFER+)

- Առաջարկվել է SAFER+ համակարգում օգտագործվող բայթերի [9, 12, 13, 16, 3, 2, 7, 6, 11, 10, 15, 14, 1, 8, 5, 4] տեղադրության փոխարեն օգտագործել բայթերի [7, 12, 9, 14, 5, 8, 13, 10, 11, 4, 3, 6, 15, 2, 1, 16] տեղադրությունը: [7, 12, 9, 14, 5, 8, 13, 10, 11, 4, 3, 6, 15, 2, 1, 16] տեղադրությունը (որը նորից անվանել ենք ***Armenian Shuffle***) ընտրվել է էֆեկտիվ հարձակումների նկատմամբ կրիպտոկայունության մակարդակից ելնելով, այսինքն այն ապահովում է հնարավոր լավագույն դիֆուզիան (նրա դեպքում համակարգը կայուն է նվազագույն 5 ռաունդի դեպքում):
- Առաջարկվել է ալգորիթմի գծային մասը, որը սկսվում է 2-PHT ձևափոխությամբ, սկսել բայթերի ընտրված տեղադրությամբ:

Ձևափոխած SAFER+ բլոկային ծածկագրական համակարգի վերծանման ընդհանուր պրոցեսը և վերծանման ռաունդի քայլերը ձևավորվում են նույն սկզբունքով, ինչպես որ SAFER+ համակարգում էր:

Ի տարբերություն SAFER+ համակարգի ստրուկտուրայի ձևափոխած SAFER+ համակարգի ստրուկտուրան հնարավորություն կտա օգտվել 32 բիթ պրոցեսորների առանձնահատկություններից և 32 հաջորդական բիթերի նկատմամբ xor գործողությունը կիրառել մեկ քայլով: Դեռ ավելին, ARM պրոցեսորների NEON տեխնոլոգիան կազմված է 32×64 բիթ և 16×128 բիթ ռեգիստրներից, որոնց միջոցով իրականացվում է վեկտորական գործողություններ [35], այսինքն 32-բիթ ARM պրոցեսորը հնարավորություն կտա կատարել գործողություններ ոչ թե առանձին բայթերի, այլ բայթային վեկտորների (այսինքն վեկտորների, որոնց կոմպոնենտները բայթեր են) հետ: Այսպիսով բանալիների ներմուծման շերտերում 16 գործողության (տե՛ս Նկար 1 և 3) փոխարեն կատարվում է ընդամենը 4 գործողություն (տե՛ս Նկար 7 և 8): NEON ստրուկտուրայում վեկտորի բազմապատկումը մատրիցով [36] կատարվում է արագ (16 չափանի վեկտորի և 16×16 չափանի մատրիցի դեպքում կատարվում է 32 գործողություն), այդ իսկ պատճառով պրոցեսորի վրա իրագործման ժամանակ հակադարձելի գծային ձևափոխության փոխարեն կարելի է ձևափոխության մուտքային վեկտորը բազմապատկել հակադարձելի գծային ձևափոխության մատրիցով (տե՛ս Պարագրաֆ 1.5.6): Հետևաբար 32 բիթ պրոցեսորի վրա SAFER+ համակարգի յուրաքանչյուր ռաունդում գործողությունների քանակը 16+16+32 է, իսկ ձևափոխած SAFER+ համակարգի յուրաքանչյուր ռաունդում՝ 4+4+32, այսինքն առաջարկված ձևափոխությունների շնորհիվ SAFER+ համակարգի գործողությունների քանակը մոտավորապես 37,5%-ով կկրճատվի: Հաշվի առնելով նաև ելքային ձևափոխությունը (տե՛ս Նկար 1 և 7) կունենաք, որ ձևափոխած SAFER+ համակարգը հայտնի SAFER+ համակարգից մոտավորապես 1,7 անգամ արագագործ է 32 բիթ պրոցեսորի վրա. $\frac{r(16+16+32)+16}{r(4+4+32)+4} \approx 1{,}7$, որտեղ r -ը ռաունդների քանակն է և կախված գաղտնի բանալու երկարությունից ընդունում է 6, 8 կամ 9 արժեքները (տե՛ս Պարագրաֆ 1.2.3):



Նկար 8: Չևափոխած SAFER+ համակարգի ծածկագրման i-րդ ռաունդի կառուցվածքը:

1.3.2. Ձևափոխած SAFER+ համակարգի ծրագրային իրականացումը և ծրագրային փաթեթի համառոտ նկարագիրը

Ձևափոխած SAFER+ համակարգը 128 բիթ երկարության մուտքային և ելքային բլոկներով սիմետրիկ ծածկագրման համակարգ է: Ծածկագրումը իրականացնում է 128, 192 կամ 256 բիթ երկարությամբ գաղտնի բանալուց գեներացված 128 բիթ երկարությամբ ենթաբանալիների մասնակցությամբ: Ձևափոխած SAFER+ ալգորիթմի ծրագրային իրականացումը կատարվել է C++ լեզվով Visual Studio 2012-ի միջոցով Windows 7 օպերացիոն համակարգի վրա: Նախորդ պարագրաֆում նկարագրված ծածկագրման, վերծանման ու ենթաբանալիների գեներացման ալգորիթմների հիման վրա ստղծվել են հետևյալ ֆունկցիաները. `GenerateKeySchedule(unsigned char key[KEY_COUNT][BLOCK_SIZE], unsigned char* Key)`, `EncryptBlock(unsigned char input[BLOCK_SIZE])`, `DecryptBlock(unsigned char input[BLOCK_SIZE])`: Ծրագրի փոփոխականները դա մուտքային 16 երկարությամբ բայթային վեկտորն է (`input[BLOCK_SIZE]`), 16, 24 կամ 32 բայթ երկարությամբ բանալին է (`Key[KEY_SIZE]`) և բանալիի երկարությունը (`KEY_SIZE`), որից կախված փոխվում է ռաունդների քանակը (`ROUND_COUNT`), հետևաբար նաև գեներացվող բանալիների քանակը (`KEY_COUNT=2*ROUND_COUNT+1`) (տե՛ս Պարագրաֆ 1.2.3): Ծրագրի բոլոր զանգվածների կոմպոնենտները $\mathbb{Z}_{256}$ օղակի էլեմենտներ են: Բանալիի և բաց տեքստի փոփոխությունը կատարվում է ծրագրի ներսից: Ծրագրի exe ֆայլը արտաբերում է 16 երկարության հետևյալ բայթային վեկտորները. ենթաբանալիները (subkeys), բաց տեքստը (plaintext) և նրա համապատասխան ծածկագիրը (cipertext), ծածկագրից բաց տեքստի վերծանման արդյունքը (recoverd palintext):

Ձևափոխած SAFER+ համակարգի ծրագրային իրականացման օրինակ.

128-բիթ գաղտնի բանալի (user-selected key)

218 133 66 5 111 247 106 7 193 31 221 19 95 202 65 49

Մուտքային բաց տեքստ (plaintext)

125 4 175 19 93 204 255 77 56 43 131 7 40 241 219 193

Ծածկագիր (ciphertext)

216 181 239 224 163 137 181 35 106 10 155 197 120 11 81 243

$K_1, K_2, \ldots, K_{13}$ *ենթաբանալիները (round subkeys)*

218 133 66 5 111 247 106 7 193 31 221 19 95 202 65 49
114 169 217 53 98 10 72 24 189 37 75 195 176 50 53 36
124 236 133 195 1 86 200 212 111 94 205 32 182 40 11 243
148 161 199 93 120 108 116 4 105 125 128 86 248 203 29 171
83 214 56 143 241 98 57 236 23 109 210 142 61 70 187 184
37 150 59 20 193 7 134 170 124 224 126 93 138 3 101 17
134 32 135 90 94 125 83 170 6 102 87 164 240 120 55 135
26 8 255 140 146 41 107 201 243 18 59 88 247 34 70 42
62 204 143 2 33 81 15 166 30 237 135 210 84 157 105 157
184 123 103 163 215 236 77 231 5 152 140 199 205 160 104 112
115 228 25 121 88 48 15 11 191 28 85 81 93 153 160 71
32 201 182 130 124 41 91 196 108 168 64 59 173 159 211 16
13 96 27 151 107 8 252 236 94 193 199 15 187 106 141 18

192-բիթ գաղտնի բանալի (user-selected key)

133 21 117 231 93 113 231 157 109 167 90 182 154 168 154 56 134 17 39 44
151 148 219 113

Մուտքային բաց տեքստ (plaintext)

235 56 44 72 176 61 48 227 3 68 227 133 39 101 239 253

Ծածկագիր (ciphertext)

129 253 47 77 16 33 157 91 19 188 57 70 154 167 8 146

$K_1, K_2, \ldots, K_{17}$ *ենթաբանալիները (round subkeys)*

133 21 117 231 93 113 231 157 109 167 90 182 154 168 154 56
238 66 240 164 46 246 252 117 2 9 104 157 159 252 109 152
73 164 1 34 96 252 179 246 142 71 156 152 12 234 166 129
89 125 186 88 165 195 133 253 176 244 60 9 6 168 138 238
50 110 16 248 171 235 1 38 203 75 103 254 36 170 213 86
226 84 134 234 5 70 88 72 107 141 2 148 165 212 90 90
124 122 53 124 80 11 64 33 107 130 17 125 118 31 129 28
237 125 16 28 6 145 39 244 212 223 2 140 52 116 235 167
234 84 12 165 92 47 104 173 140 36 41 188 230 194 77 164
253 95 132 125 198 182 133 91 183 165 219 91 246 191 160 246

146 205 232 241 174 242 170 153 39 151 249 155 86 91 212 56
103 64 114 53 138 6 200 7 64 205 139 3 187 64 91 252
193 62 176 7 82 108 22 138 135 20 5 128 200 166 237 140
153 154 112 40 173 207 34 21 213 44 132 146 157 245 104 231
151 211 150 177 57 101 132 198 97 88 178 198 68 230 140 248
189 211 56 87 192 16 32 235 235 128 203 233 44 32 1 31
14 87 190 221 81 55 146 129 225 74 76 40 176 183 153 147

256-բիթ գաղտնի բանալի (user-selected key)

107 90 214 181 173 9 249 243 82 219 109 187 45 139 24 73 114 22 198 205 11
5 77 143 51 251 62 18 225 79 87 122

Մուտքային բաց տեքստ (plaintext)

111 5 89 215 57 239 41 141 49 96 3 151 250 116 82 56

Ծածկագիր (ciphertext)

233 158 196 112 107 213 130 173 147 158 217 193 224 238 9 49

$K_1, K_2, \ldots, K_{19}$ *ենթաբանալիները (round subkeys)*

107 90 214 181 173 9 249 243 82 219 109 187 45 139 24 73
24 77 94 39 235 134 175 156 163 162 144 50 182 232 246 247
161 24 21 8 229 145 236 3 83 136 65 80 108 46 161 194
245 30 234 124 81 141 237 35 186 25 2 4 40 127 148 45
55 231 49 94 250 46 50 118 244 121 63 15 227 250 207 112
174 93 177 93 31 207 218 145 220 76 138 138 40 164 42 20
196 211 201 77 156 31 138 173 97 199 192 145 245 166 79 222
184 26 151 126 167 227 131 164 246 93 162 128 104 227 249 213
207 136 31 170 239 18 230 190 120 41 200 93 90 51 191 194
158 248 172 18 221 162 14 186 223 162 232 247 122 75 145 209
87 14 141 169 14 54 165 218 15 255 214 183 179 226 171 213
113 101 56 48 172 221 210 70 131 180 108 230 248 247 71 78
234 108 136 24 17 188 16 164 190 27 28 97 126 13 127 86
10 89 248 30 48 159 242 207 13 229 147 64 216 138 183 228
141 24 69 197 184 236 82 136 39 209 31 159 232 89 116 192
223 81 216 75 244 127 46 25 174 236 154 14 192 96 63 219
250 92 93 126 197 168 4 159 61 192 117 197 170 169 127 38
140 188 229 47 206 123 119 141 153 131 33 115 253 171 6 115
221 40 116 108 219 180 101 201 27 8 149 231 86 53 149 164

1.4. 256-բիթային SAFER-256 ծածկագրական համակարգի նկարագրությունը

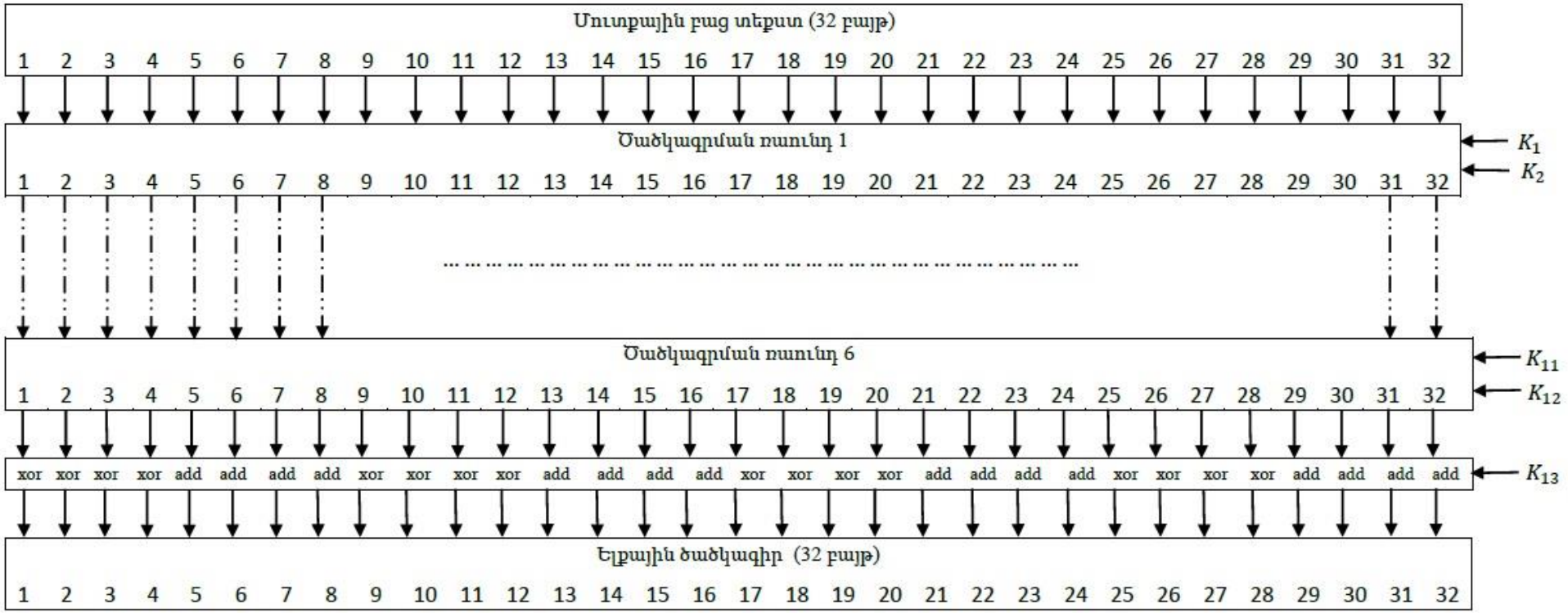
SAFER ընտանիքը կազմված է 64 և 128 բիթ երկարության բլոկի ծածկագրման ալգորիթմներից, որոնք մշակվել են տարբեր ժամանակահատվածներում: 256 բիթ ծածկագրական համակարգ մշակելու գաղափարը, որին նպաստեց ձևափոխած SAFER+ համակարգի դիֆերենցիալ շղթաների անցումային հավանականությունների (տե՛ս գլուխ 2) էականորեն փոքր լինելը, հետապնդում է երկու նպատակ: Առաջինը այն է, որ բոլոր գոյություն ունեցող և հայտնի ծածկագրական համակարգերը բլոկի մինչև 128 բիթ երկարությամբ բլոկային ծածկագրական համակարգեր են, և չնայած որ այդ համակարգերը ունեն նաև 256 բիթ երկարությամբ օգտագործողի կողմից ընտրված բանալու դեպքը, նրանք իրականում չեն ապահովում 256 բիթ անվտանգություն 128 բիթ երկարությամբ բլոկի համար՝ բախում հարձակման (collision attack) նկատմամբ [17]: (Բախում հարձակում. գտնել երկու իրարից տարբեր $m_1 \neq m_2$ մուտքեր, այնպիսին որ $\mathrm{hash}(m_1) = \mathrm{hash}(m_2)$): Իսկ երկրորդ նպատակն է ստեղծել ծածկագրման արագ և արդյունավետ ալգորիթմ: 256 բիթ բլոկի և բանալու երկարության նոր ծածկագրական համակարգը, որն անվանել ենք SAFER-256, կարելի է համարել SAFER ընտանիքի բլոկային ծածկագրական համակարգերի տրամաբանական շարունակությունը: 256 բիթ երկարության բլոկը SAFER-256 համակարգը ծածկագրում է 1,5 անգամ ավելի արագ, քան հայտնի SAFER+ համակարգը, քանի որ 256 բիթ բլոկը ծածկագրելու համար 9 ռաունդից կամված SAFER+ համակարգը պետք է կիրառել 2 անգամ, իսկ 6 ռաունդից կազմված SAFER-256 համակարգը՝ միայն մեկ անգամ:

SAFER-256 համակարգի ծածկագրման ընդհանուր կառուցվածքը բերված է նկար 9-ում: Ծածկագրման մուտքը 32 բայթ երկարության բաց տեքստն է, որը անցնում է ծածկագրման 6 ռաունդներով: Ծածկագրման յուրաքանչյուր ռաունդում օգտագործվում է 32 բայթ երկարության երկու ենթաբանալի, և մեկ ենթաբանալի ելքային ձևափոխության ժամանակ ավելացվում է ռաունդների 32 բայթ երկարության ելքին: Համակարգի $K_1, K_2, \dots, K_{13}$ ենթաբանալիները գեներացվում են 256 բիթ երկարության

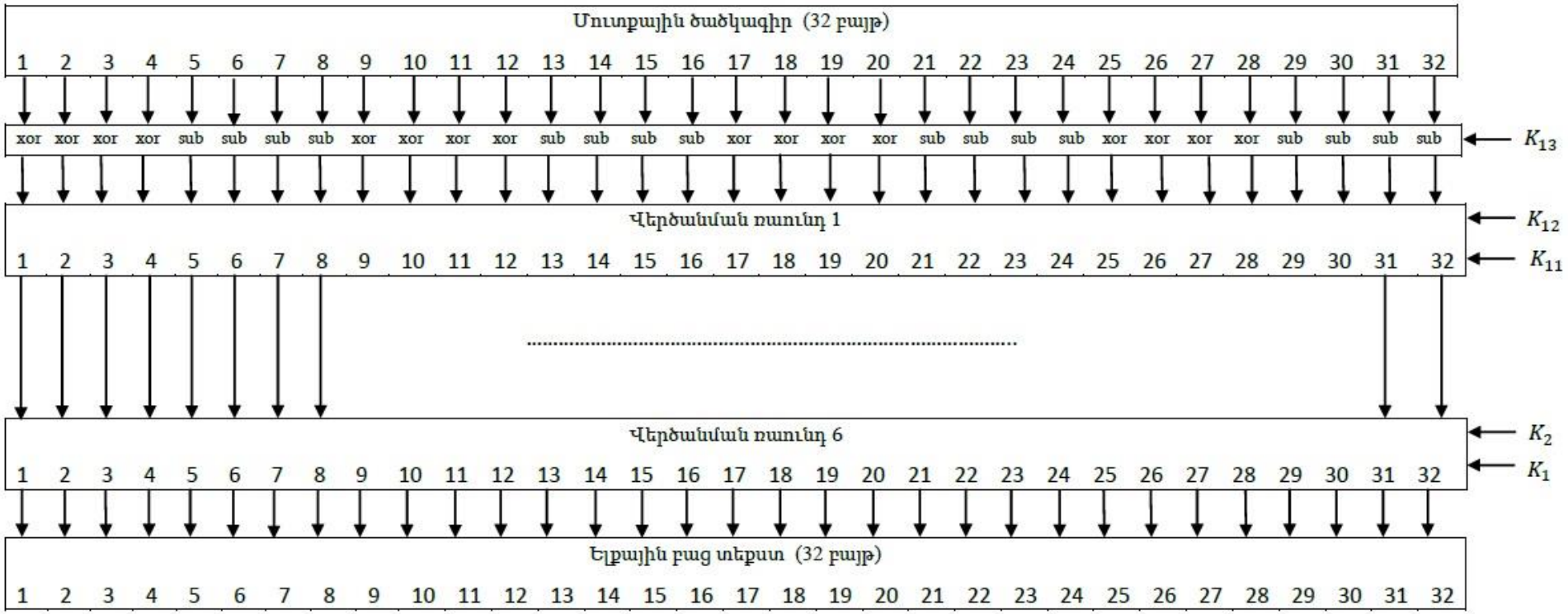
գաղտնի բանալուց: SAFER-256 համակարգի ենթաբանալիների գեներացման ալգորիթմի նկարագրությունը և սխեման բերված է 1.4.3 պարագրաֆում:

SAFER-256 համակարգի ելքային ձևափոխության ժամանակ ենթաբանալիներից վերջինը՝ K_{13}-ը, «ավելացվում» է ծածկագրման 6 ռաունդների ելքային բլոկին հետևյալ եղանակով. 1, 2, 3, 4, 9, 10, 11, 12, 17, 18, 19, 20, 25, 26, 27, 28 բայթերը գումարվում են բիթ առ բիթ ըստ մոդուլ 2-ի, իսկ 5, 6, 7, 8, 13, 14, 15, 16, 21, 22, 23, 24, 29, 30, 31, 32 բայթերը ըստ մոդուլ 256-ի (տե՛ս Նկար 9): Ելքային ձևափոխության արդյունքում ստացվում է 32 բայթ երկարության ծածկագիրը:

SAFER-256 համակարգի վերծանման պրոցեսը, որի մուտքը իրենից ներկայացնում է 32 բայթ երկարության ծածկագիրը, սկսվում է մուտքային ձևափոխությունից, որի ժամանակ K_{13} ենթաբանալին «հանվում» է ծածկագրված տեքստից. ենթաբանալու 1, 2, 3, 4, 9, 10, 11, 12, 17, 18, 19, 20, 25, 26, 27, 28 բայթերը գումարվում են ծածկագրի համապատասխան բայթերին բիթ առ բիթ ըստ մոդուլ 2-ի (բիթային “xor” գործողություն), իսկ ենթաբանալու 5, 6, 7, 8, 13, 14, 15, 16, 21, 22, 23, 24, 29, 30, 31, 32 բայթերը հանվում են ծածկագրի համապատասխան բայթերից ըստ ըստ մոդուլ 256-ի, այս կերպ «զրոյացնելով» ծածկագրման պրոցոսում առկա ելքային ձևափոխությունը (տե՛ս Նկար 10): Արդյունքում ստացված 32 բայթ երկարության բլոկը, որը համընկնում է ծածկագրման 6 ռաունդների ելքում ստացված 32 բայթ բլոկի հետ, անցնում է վերծանման 6 ռաունդներով: Վերծանման i-րդ ռաունդը ծածկագրման $(7-i)$-րդ ռաունդի հակադարձ ձևափոխությունն է, որտեղ $i=1,2,\dots,6$: SAFER-256 համակարգի վերծանման i-րդ ռաունդի և ծածկագրման $(7-i)$-րդ ռաունդի ենթաբանալիները նույնն են, սակայն կիրառվում են հակառակ հերթականությամբ: Վերծանման 6 ռաունդների ելքում ստացվում է մուտքային 32 բայթ երկարության բաց տեքստը:



Նկար 9: SAFER-256 համակարգի ծածկագրման ընդհանուր կառուցվածքը:



Նկար 10: SAFER-256 համակարգի վերծանման ընդհանուր կառուցվածքը:

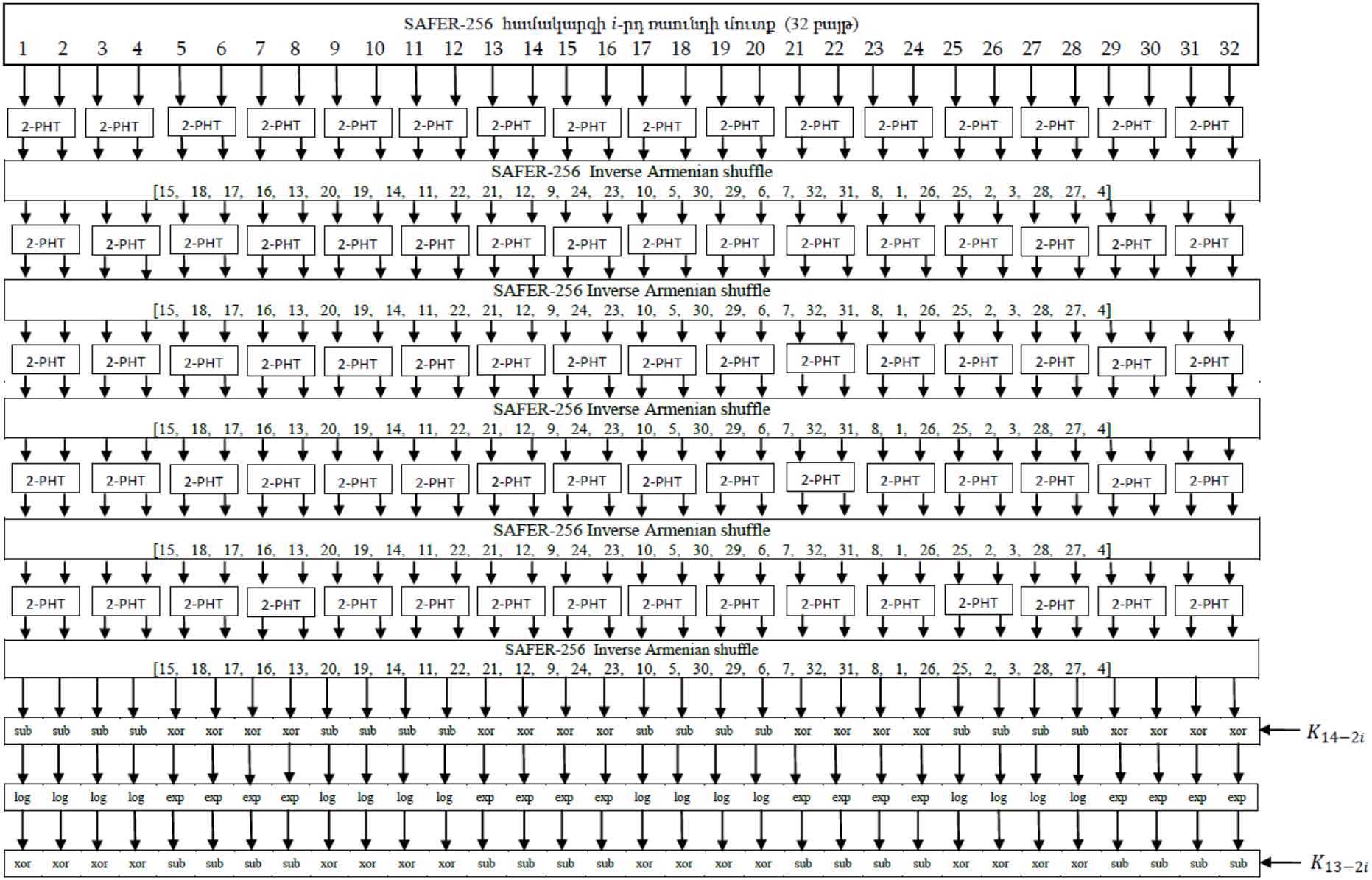
1.4.1. SAFER-256 համակարգի ծածկագրման ռաունդի նկարագրությունը

Նկար 11-ում պատկերված է SAFER-256 համակարգի ծածկագրման ռաունդի կառուցվածքը: Ինչպես երևում է նկարից, i-րդ ռաունդի 32-բայթ մուտքին «գումարվում» է ռաունդի առաջին K_{2i-1} ենթաբանալին հետևյալ եղանակով. 1, 2, 3, 4, 9, 10, 11, 12, 17, 18, 19, 20, 25, 26, 27, 28 բայթերը գումարվում են բլոկի համապատասխան բայթերին բիթ առ բիթ ըստ մոդուլ 2-ի, իսկ 5, 6, 7,8, 13, 14, 15, 16, 21, 22, 23, 24, 29, 30, 31, 32 բայթերը՝ ըստ մոդուլ 256-ի: Արդյունքում ստացված 32 բայթ երկարության բլոկը անցնում է ռաունդի ոչ գծային շերտով. $j = 1, 2, 3, 4, 9, 10, 11, 12, 17, 18, 19, 20, 25, 26, 27, 28$ բայթերի x արժեքների վրա կիրառվում է $45^x mod\ 257$ ցուցչային ֆունկցիան (պայմանով, որ երբ $x = 128$, ապա $45^{128} mod\ 257 = 256$ հավասար է 0-ի), իսկ 5, 6, 7, 8, 13, 14, 15, 16, 21, 22, 23, 24, 29, 30, 31, 32 բայթերի x արժեքների վրա՝ $\log_{45} x$ լոգարիթմական ֆունկցիան (պայմանով, որ երբ $x = 0$, $\log_{45} 0$ հավասար է 128-ի): Այնուհետև ոչ գծային շերտի ելքին «գումարվում» է ռաունդի երկրորդ K_{2i} ենթաբանալին. բլոկի 1, 2, 3, 4, 9, 10, 11, 12, 17, 18, 19, 20, 25, 26, 27, 28 բայթերին գումարվում են բանալու համապատասխան բայթերը ըստ մոդուլ 256-ի, իսկ 5, 6, 7, 8, 13, 14, 15, 16, 21, 22, 23, 24, 29, 30, 31, 32 բայթերին՝ բիթ առ բիթ ըստ մոդուլ 2-ի: Ստացված 32 բայթի նկատմամբ կիրառվում է հակադարձելի գծային ձևափոխությունը: SAFER-256 ծածկագրական համակարգի հակադարձելի գծային ձևափոխության գործողություններից մեկը SAFER+ համակարգից հայտնի պսևդո-Հադամարի 2-PHT հակադարձելի ձևափոխությունն է, իսկ մյուսը 32 բայթերի հետևյալ տեղադրությունը. [24, 28, 29, 32, 17, 20, 21, 24, 13, 16, 9, 12, 5, 8, 1, 4, 3, 2, 7, 6, 11, 10, 15, 14, 27, 26, 31, 30, 19, 18, 23, 22]: SAFER-256 համակարգի այս տեղադրությունը ևս անվանել ենք ***"Armenian Shuffle"***: SAFER-256 համակարգի հակադարձելի գծային ձևափոխությունը բաղկացած է 5 անգամ կիրառվող 16 հատ 2-PHT ձևափոխությունից և 5 անգամ կիրառվող բայթերի Armenian Shuffle տեղադրությունից, որոնք կիրառվում են մեկընդմեջ (տե՛ս Նկար 11):

Նկար 11: SAFER-256 համակարգի ծածկագրման i-րդ ռաունդի կառուցվածքը:

1.4.2. SAFER-256 համակարգի վերծանման ռաունդի նկարագրությունը

SAFER-256 համակարգի վերծանման ռաունդը սխեմատիկորեն պատկերված է նկար 12-ում: Ռաունդի մուտքում 32 բայթ երկարության բլոկի նկատմամբ կիրառվում է 2-PHT ձևափոխության հակադարձ 2-IPHT ձևափոխությունը: 2-IPHT ձևափոխությանը հաջորդող գործողությունը SAFER-256 համակարգի Armenian Shuffle տեղադրության հակադարձ տեղադրությունն է՝ [15, 18, 17, 16, 13, 20, 19, 14, 11, 22, 21, 12, 9, 24, 23, 10, 5, 30, 29, 6, 7, 32, 31, 8, 1, 26, 25, 2, 3, 28, 27, 4]: Այս երկու գործողությունները նշված հաջորդականությամբ կրկնվում են ևս չորս անգամ: 32 բայթ երկարության ելքային արդյունքից, որը համընկնում է ծածկագրման հակադարձելի գծային ձևափոխության մուտքի հետ $(7-i)$-րդ ռաունդում, «հանվում» է ծածկագրման համապատասխան ռաունդում կիրառվող երկրորդ ենթաբանալին՝ K_{14-2i}-ը. ենթաբանալու 1, 2, 3, 4, 9, 10, 11, 12, 17, 18, 19, 20, 25, 26, 27, 28 բայթերը հանվում են 32-բայթ բլոկի համապատասխան բայթերից ըստ մոդուլ 256-ի, իսկ 32 բայթ մուտքի և ենթաբանալու 5, 6, 7, 8, 13, 14, 15, 16, 21, 22, 23, 24, 29, 30, 31, 32 բայթերը գումարվում են բիթ առ բիթ ըստ մոդուլ 2-ի, որից հետո ստացված 32 բայթ երկարության արդյունքը անցնում է վերծանման ոչ գծային շերտով. $j = 1, 2, 3, 4, 9, 10, 11, 12, 17, 18, 19, 20, 25, 26, 27, 28$ բայթերի x արժեքները վրա կիրառվում է $\log_{45} x$ լոգարիթմական ֆունկցիան (երբ $x = 0$, ապա $\log_{45} 0$ հավասար է 128-ի), իսկ $j = 5, 6, 7, 8, 13, 14, 15, 16, 21, 22, 23, 24, 29, 30, 31, 32$ բայթերի x արժեքների վրա՝ $45^x mod\ 257$ ցուցչային ֆունկցիան (նորից պայմանով, որ երբ $x = 128$, ապա $45^{128} mod\ 257 = 256$ հավասար է 0-ի): Այնուհետև ծածկագրման $(7-i)$-րդ ռաունդի առաջին K_{13-2i} ենթաբանալին «հանվում» է 32 բայթ երկարության ելքից. ենթաբանալու 1, 2, 3, 4, 9, 10, 11, 12, 17, 18, 19, 20, 25, 26, 27, 28 բայթերը բիթ առ բիթ ըստ մոդուլ 2-ի գումարվում են բլոկի համապատասխան բայթերին, իսկ 5, 6, 7, 8, 13, 14, 15, 16, 21, 22, 23, 24, 29, 30, 31, 32 բայթերը հանվում են ձևափոխության մուտքային բլոկի համապատասխան բայթերից ըստ մոդուլ 256-ի՝ ձևավորելով տվյալ ռաունդի 32 բայթ երկարության ելքը:



Նկար 12: SAFER-256 համակարգի վերծանման i-րդ ռաունդի կառուցվածքը:

1.4.3. SAFER-256 համակարգի շեղումնային բառերի որոշումը և ենթաբանալիների գեներացման մեթոդը

SAFER-256 համակարգի նկատմամբ, նախորոք ստեղծված ծրագրային փաթեթի միջոցով կատարված դիֆերենցիալ վերլուծությունը (տե՛ս Հավելված 2) ցույց է տվել, որ այս նոր համակարգը կայուն է դիֆերենցիալ վերլուծության նկատմամբ նվազագույնը 5 ռաունդի դեպքում և կազմված է 6 ռաունդից (տե՛ս Գլուխ 2): Հետևաբար SAFER-256 համակարգի համար՝ օգտագործողի կողմից ընտրված բանալուց, անհրաժեշտ է գեներացնել 13 հատ 32 բայթ երկարության ենթաբանալի. երկու ենթաբանալի յուրաքանչյուր ռաունդի և մեկ ենթաբանալի ելքային ձևափոխության համար (տե՛ս Նկար 9):

SAFER-256 համակարգի ենթաբանալիների գեներացման ալգորիթմի հիմքում ընկած է SAFER+ համակարգի ենթաբանալիների գեներացման ալգորիթմի տրամաբանական կառուցվածքը: Այսինքն օգտագործվում են շեղումնային բառեր՝ ենթաբանալիների կամայական բաշխումն ապահովելու համար, օգտագործվում է բանալու բայթային ռեգիստրի գաղափարը, բանալու բայթային ռեգիստրի բայթերի 3 բիթով տեղաշարժ, և բանալու բայթային ռեգիստրից ենթաբանալիների բայթերի ընտրման նույն եղանակը:

SAFER-256 համակարգի ենթաբանալիների գեներացման համար անհրաժեշտ շեղումնային բառերի երկարությունը 32 բայթ է: $B_2, B_3, \dots, B_{13}$ շեղումնայի բառերի շեղումնային բայթերը որոշվում են SAFER+ համակարգի շեղումնայի բառերի շեղումնային բայթերի որոշման հետևյալ բանաձով.

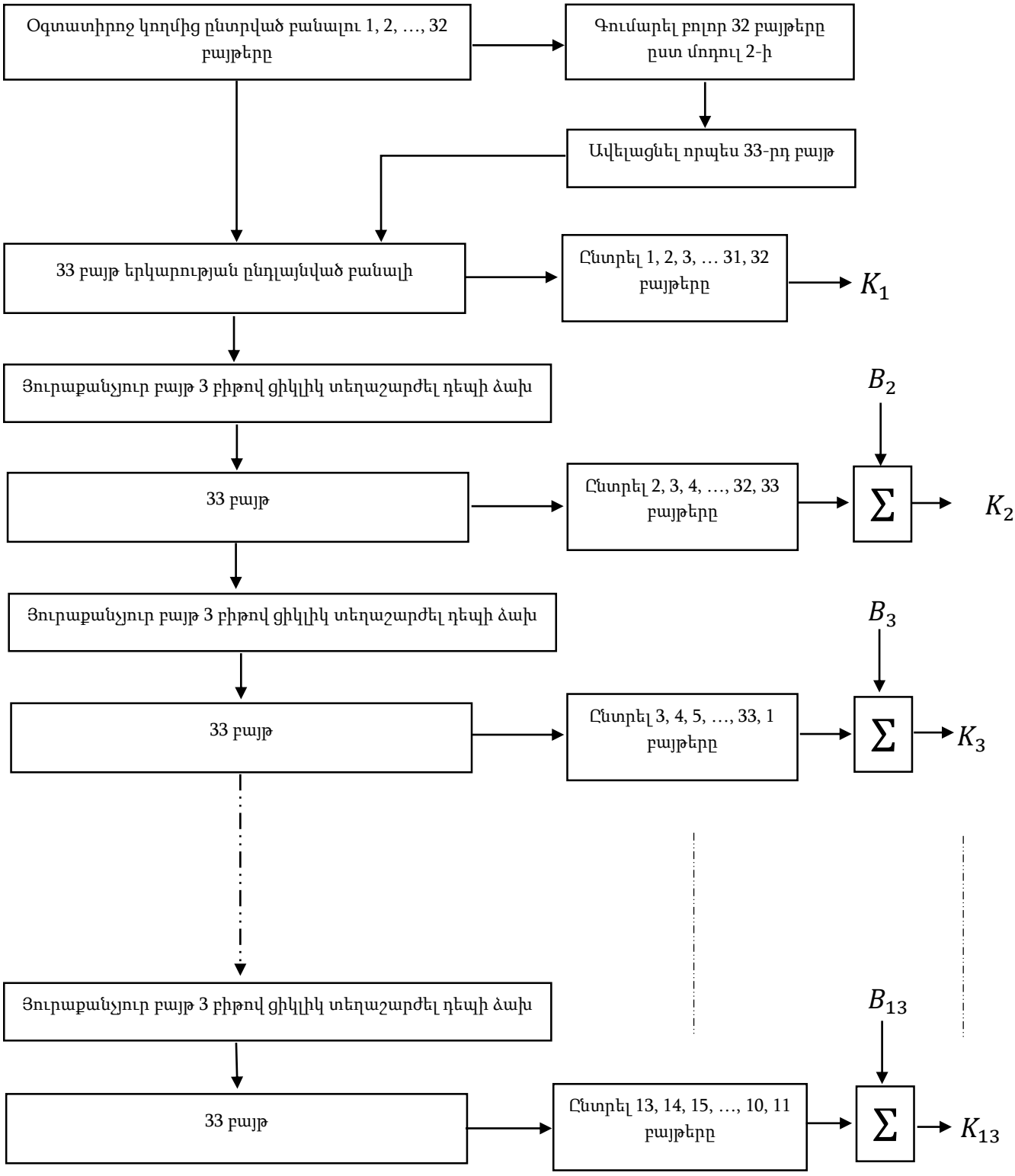
$$B_{i,j} = 45^{(45^{17i+j} mod 257)} mod 257,$$

որտեղ $i = 2, 3, \dots, 13$ և $j = 1, 2, \dots, 32$, ընդ որում, եթե այս արտահայտությունը հավասար է 256-ի , ապա $B_{i,j}$-ն ընդունում է 0 արժեք: Աղյուսակ 2-ում ներկայացված է SAFER-256 համակարգի ենթաբանալիների գեներացման համար անհրաժեշտ 13 շեղումնային բառերը: SAFER-256 համակարգի ենթաբանալիների գեներացման ալգորիթմը սխեմատիկորեն պատկերված է նկար 13-ում:

SAFER-256 համակարգի ենթաբանալիները գեներացվում են հետևյալ եղանակով. օգտագործողի կողմից ընտրված 32 բայթ (256 բիթ) երկարության բանալին օգտագործվում է որպես առաջին ենթաբանալի՝ K_1: Ստեղծվում է 33-բայթ բանալու ռեգիստր, որի առաջին 32 բայթերը օգտագործողի կողմից ընտրված բանալու 32 բայթերն են, իսկ վերջին 33-րդ բայթը այդ 32 բայթերի բիթ առ բիթ ըստ մոդուլ 2-ի գումարն է (տե՛ս Նկար 13): Բանալու ռեգիստրի յուրաքանչյուր բայթ 3 բիթով ցիկլիկ տեղաշարժվում է դեպի ձախ, և աղյուսակ 2-ի առաջին սյան B_2 շեղումնային բառի բայթերը գումարվում են բայթային ռեգիստրի 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33 բայթերին ըստ մոդուլ 256-ի՝ ձևավորելով երկրորդ K_2 ենթաբանալին: Այնուհետև բանալու ռեգիստրի բայթերը նորից 3 բիթով ցիկլիկ տեղաշարժվում են դեպի ձախ և K_3 ենթաբանալին ձևավորելու նպատակով աղյուսակ 2-ի երկրորդ սյան B_3 շեղումնային բառի 16 բայթերը ըստ մոդուլ 256-ի գումարվում են ռեգիստրի 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 1 բայթերին: Նույն տրամաբանությամբ որոշվում են համակարգի մնացած ենթաբանալիները: Ելքային ձևափոխության K_{13} ենթաբանալին ձևավորվում է B_{13} շեղումնային բառի 32 բայթերի (տես աղյուսակ 2-ի վերջին սյունը) և ռեգիստրի 3, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 բայթերի ըստ մոդուլ 256-ի գումարման արդյունքում:

Աղյուսակ 2: SAFER-256 համակարգի ենթաբանալիների գեներացման $B_2, B_3, \ldots, B_{13}$ շեղումնային բառերը:

B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}
70	236	138	93	42	221	58	125	192	252	250	24
151	171	195	87	97	4	208	173	141	32	11	180
177	170	216	146	184	128	28	178	207	66	33	7
186	198	137	31	52	222	209	239	169	199	0	132
163	103	106	213	50	231	48	194	129	8	26	234
183	149	233	113	25	49	62	135	226	228	249	91
16	88	54	92	253	214	18	206	196	9	166	164
10	13	73	187	251	127	161	117	39	85	185	200
197	248	67	34	23	1	205	6	47	94	232	14
55	154	191	193	64	162	15	19	108	140	158	203
179	246	235	190	230	247	224	2	122	20	98	72
201	110	212	123	81	57	168	144	159	118	76	105
90	102	150	188	29	218	175	79	8	96	217	75
40	220	155	153	65	111	130	46	225	255	145	78
172	5	104	99	68	35	89	114	21	223	80	156
100	61	160	148	143	202	44	51	56	215	210	53
165	211	101	95	41	254	245	133	43	152	238	121
236	138	93	42	221	58	125	192	252	250	24	69
171	195	87	97	4	208	173	141	32	11	180	77
170	216	146	184	128	28	178	207	66	33	7	84
198	137	31	52	222	209	239	169	199	0	132	229
103	106	213	50	231	48	194	129	8	26	234	37
149	233	113	25	49	62	135	226	228	249	91	60
88	54	92	253	214	18	206	196	9	166	164	12
13	73	187	251	127	161	117	39	85	185	200	74
248	67	34	23	1	205	6	47	94	232	14	139
154	191	193	64	162	15	19	108	140	158	203	63
246	235	190	230	247	224	2	122	20	98	72	204
110	212	123	81	57	168	144	159	118	76	105	167
102	150	188	29	218	175	79	82	96	217	75	219
220	155	153	65	111	130	46	225	255	145	78	107
5	104	99	68	35	89	114	21	223	80	156	174



Նկար 13: SAFER-256 համակարգի ենթաբանալիների գեներացման սխեման:

1.4.4. SAFER-256 համակարգի ծրագրային իրականացումը և ծրագրային փաթեթի համառոտ նկարագիրը

SAFER-256 համակարգը 256 բիթ երկարությամբ բլոկի ծածկագրման ալգորիթմ է: Սիմետրիկ բանալին 256 բիթ երկարությամբ վեկտոր է, որից գեներացվում են 256 բիթ երկարությամբ ենթաբանալիներ: Նկար 9-ում և 11-ում սխեմատիկորեն պատկերված ծածկագրման ալգորիթմի, նկար 10-ում և 12-ում սխեմատիկորեն պատկերված վերծանման ալգորիթմի, ինչպես նաև նկար 13-ում սխեմատիկորեն պատկերված ենթաբանալիների գեներացման ալգորիթմի հիման վրա կատարվել է SAFER-256 համակարգի ծրագրային իրականացումը C++ լեզվով Visual Studio 2012-ի միջոցով Windows 7 օպերացիոն համակարգի վրա: Տվյալների նկարագրության համար ընտրվել են այնպիսի անվանումներ, որոնք օգնում են հեշտ հասկանալ կոդը (տե՛ս Հավելված 1) և աշխատեցնել: Ծրագրում փոփոխականները 32 երկարության `Key[KEY_SIZE]` և `input[BLOCK_SIZE]` բայթային վեկտորներն (զանգվածներն) են, որոնց կոմպոնենտները $\mathbb{Z}_{256}$ օղակից են և նրանց փոփոխությունը կատարվում է ծրագրի ներսից: Ծրագիրը կազմված է հետևյալ հիմնական ֆունկցիաներից.

1. `void GenerateKeySchedule(unsigned char key[KEY_COUNT][BLOCK_SIZE], unsigned char* Key)` – գեներացնում է ենթաբանալիները (subkeys),
2. `void EncryptBlock(unsigned char input[BLOCK_SIZE])` – կատարում է 32 բայթ երկարության նշված բաց տեքստի (plaintext) ծածկագրում (ciphertext),
3. `void DecryptBlock(unsigned char input[BLOCK_SIZE])` – կատարում է 32 բայթ երկարության բաց տեքստի վերծանում (recoverd plaintext):

Black_box_SAFER-256.exe ֆայլի վրա կարելի է տեսնել նշված ֆունկցիաների արժեքները, որոնք 32 երկարության բայթային վեկտորներ են (subkeys, plaintext/ciphertext, recoverd plaintext):

SAFER-256 համակարգի ծրագրային իրականացման օրինակ.

256-բիթ գաղտնի բանալի (user-selected key)

75 199 59 17 253 138 14 173 241 147 48 71 201 91 65 171 161 108 193 20 110 7 168 207 238 56 97 81 199 41 162 235

Մուտքային բաց տեքստ (plaintext)

188 214 84 60 8 113 124 14 252 207 88 192 187 17 28 161 254 223 102 41 167 154 202 168 186 100 61 44 171 199 55 157

Ծածկագիր (ciphertext)

27 63 245 252 209 239 1 216 221 152 11 126 173 152 79 179 62 2 155 220 132 67 24 68 159 119 172 222 236 95 149 139

$K_1, K_2, \dots, K_{13}$ *ենթաբանալիները (round subkeys)*

75 199 59 17 253 138 14 173 241 147 48 71 201 91 65 171 161 108 193 20 110 7 168 207 238 56 97 81 199 41 162 235
132 112 57 169 247 39 125 153 97 184 237 23 52 50 9 113 8 250 75 29 254 172 19 207 206 3 36 52 183 123 59 71
186 239 41 104 234 0 212 241 4 107 104 68 182 198 109 88 67 143 94 153 179 93 164 68 161 151 176 53 124 144 173 58
172 190 237 165 197 204 93 169 209 82 161 86 237 222 64 35 141 57 101 227 190 178 225 30 93 177 19 3 82 76 47 242
60 255 114 249 244 170 95 47 190 118 210 53 214 95 127 213 69 154 235 180 34 181 47 18 119 169 106 164 213 209 189 247
111 104 142 44 251 49 160 223 196 224 187 33 83 33 78 198 172 49 235 247 250 151 217 185 19 82 151 27 222 189 12 171
21 186 71 44 167 78 253 236 6 80 125 234 225 191 220 230 160 121 139 252 86 117 93 182 43 124 48 13 199 155 198 80
239 14 142 215 24 119 125 201 66 67 109 224 49 79 57 65 238 90 180 222 25 186 172 34 242 15 124 250 247 113 237 195
110 64 226 54 139 226 15 32 167 127 195 164 189 53 26 2 115 248 238 32 112 170 132 175 111 122 51 181 176 79 107 35
92 14 9 247 91 236 33 52 146 122 26 18 138 38 147 175 236 7 170 128 16 29 67 75 175 156 101 156 101 180 111 76
8 241 180 157 88 206 113 112 206 145 175 55 138 242 154 229 240 78 252 107 168 20 11 120 170 182 226 225 238 92 252 204
136 158 215 130 113 60 126 60 16 122 112 157 120 110 192 148 144 167 6 76 91 122 241 51 62 48 198 93 133 166 49 195
180 105 27 62 4 33 192 9 244 59 210 101 57 209 178 74 245 215 119 18 105 217 184 191 91 106 231 172 129 250 164 177

1.5. Ձևափոխած SAFER+ և SAFER-256 բլոկային ծածկագրական համակարգերի տրամաբանական հիմնավորումները

Ձևափոխած SAFER+ և SAFER-256 համակարգերի հիմքում ընկած են SAFER+ համակարգի հետևյալ տրամաբանական սկզբունքները.

- *Ծածկագրման ընթացակարգը*
- *Բայթային կողմնորոշվածությունը*
- *Խմբային գործողությունները ռաունդի (ցիկլի) սկզբում*
- *Երկու ադիտիվ խմբերի կիրառումը*
- *Աստիճանային և լոգարիթմական ֆունկցիաների կիրառումը ալգորիթմի ոչ գծային շերտում*
- *Դիֆուզիայի ապահովումը մատրիցի միջոցով*
- *Բանալիների հաջարդականության ընտրությունը*
- *Ռաունդների քանակը*

Վերոնշյալ սկզբունքները, որոնք բերված են նաև [21] աշխատանքում, առանձին-առանձին ներկայացված են այս պարագրաֆի ենթապարագրաֆներում:

1.5.1. Ծածկագրման ընթացակարգը

Հարկ է նշել, որ ի տարբերություն տվյալների ծածկագրման ալգորիթմի (Data Encryption Algoritm (DEA)), որը կիրառվում է տվյալներ ծածկագրման ստանդարտի (Data Encryption Standart (DES)), ինչպես նաև այլ ավելի ուշ ներկայացված իտերատիվ բլոկային ծածկագրական համակարգերի մեջ, SAFER+ և SAFER-256 համակարգերը չեն հանդիսանում Ֆեիստելյան համակարգեր (Feistel cipher): Ֆեիստելյան համակարգում ռաունդի մուտքը բաժանվում է աջ և ձախ մասերի. i-րդ ռաունդի մուտքը աջ և ձախ մասերի

բաժանելուց հետո նշանակենք համապատասխանաբար R_{i-1} և L_{i-1}: f ոչ գծային ֆունկցիան կիրառվում է աջ մասի ու ռաունդի K_i բանալու վրա և ստացված արդյունքը բիթ առ բիթ ըստ մոդուլ 2-ի գումարվում է ռաունդի մուտքի ձախ մասին՝ $L_{i-1} \oplus f(R_{i-1}, K_i)$, ձևավորելով «նոր» ձախ մաս: Ռաունդ վերջում կատարվում է աջ և «նոր» ձախ մասերի տեղափոխություն: Այսպիսով հաջորդ $(i+1)$-րդ ռաունդի մուտքային աջ և ձախ մասերը կլինեն. $L_i = R_{i-1}, \quad R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$: Ֆեիստելյան ստրուկտուրայի հիմնական հատկությունը այն է, որ անկախ f ֆունկցիայի ընտրությունից ռաունդ ֆունկցիան հակադարձելի է: Վերջին ռաունդում աջ և ձախ մասերի տեղափոխություն չի կատարվում, ինչի արդյունքում վերծանումը և ծածկագրումը կատարվում են նման ընթացակարգով, բացի դրանից օգտագործվում է նույն բանալիները միայն թե հակառակ հերթականությամբ: Ծածկագրման և վերծանման ալգորիթմների նմանությունը Ֆեիստելյան համակարգի կիրառական առավելությունն է: Մյուս կողմից Ֆեիստելյան համակարգում դիֆուզիան [30] կատարվում է փոքր ինչ դանդաղ: Օրինակ, երկու ռաունդ է պահանջվում որպեսզի ալգորիթմը ձևափոխի ալգորիթմի յուրաքնաչյուր մուտքային բիթ: Ավելին, f ֆունկցիայի տեսքը, որից է հիմնականում կախված համակարգի կրիպտոկայունությունը, ևս առաջացնում է որոշակի դժվարություններ:

Ֆեիստելյան համակարգի ընդհանրացումը ևս իտերատիվ բլոկային ծածկագրական համակարգ է, որը երբեմն անվանում են նաև տեղադրություն/տեղափոխություն համակարգ: Այսպիսի համակարգերի յուրաքնչյուր ռաունդի մուտքին սկզբում ավելացվում է ռաունդի բանալին հետո հակադարձելի ֆունկցիան, այնուհետև կատարվում է բայթերի տեղադրություն: Տեղադրություն/տեղափոխություն համակարգի ստրուկտուրան ի տարբերություն Ֆեիստելյան համակարգի ավելի մեծ հնարավորություններ է տալիս համակարգ մշակողին, ինչը թույլ է տալիս ապահովել ավելի լավ դիֆուզիա, հետևաբար ստեղծել ավելի արագագործ համակարգ: Միայն թե այս համակարգի ծածկագրման և վերծանման ալգորիթմները էականորեն տարբերվում են:

SAFER ընտանիքի ծածկագրական համակարգերը չեն հանդիսանում Ֆեիստելյան կամ տեղադրություն/տեղափոխություն համակարգեր: Ձևափոխած SAFER+ և SAFER-256

համակարգերին, ինչպես որ SAFER ընտանիքի SAFER+ և SAFER++ համակարգերին, կարելի է անվանել տեղադրություն/գծային ձևափոխություն համակարգեր այն իմաստով, որ յուրաքանչյուր ռաունդի մուտքում կիրառվում է հակադարձելի ֆունկցիան ռաունդի բանալիների մասնակցությամբ, որից հետո կիրառվում է հակադարձելի գծային ձևափոխությունը: Տեղադրություն/գծային ձևափոխություն համակարգը կարելի է ասել տեղադրություն/տեղափոխություն համակարգի ընդհանրացումն է, քանի որ բայթերի տեղադրությունը ևս հակադարձելի գծային ձևափոխություն է: Սակայն տեղադրություն/գծային ձևափոխություն համակարգերի կառուցվածքը համակարգ ստեղծողին ավելի շատ հնարավորություններ է տալիս, մասնավորապես այստեղ կարելի է ապահովել ավելի լավ դիֆուզիա հակադարձելի գծային ձևափոխության ճիշտ և հիմնավորված ընտրության դեպքում (տե՛ս Պարագրաֆ 1.6):

1.5.2. Բայթային կողմնորոշվածությունը

SAFER ընտանիքի բոլոր համակարգերը, այդ թվում նաև առաջարկված նոր համակարգերը, ունեն բայթային կողմնորոշվածություն այն իմաստով, որ ենթաբանալիների գեներացման, ծածկագրման և վերծանման ալգորիթմներում օգտագործվում են միայն բայթը բայթին արտապատկերող ֆունկցիաներ: Այսպիսի բայթային կողմնորոշվածությունը հեշտացնում է ծրագրային իրագործումը, ինչպես նաև ծածկագրման և վերծանման ալգորիթմների միկրոէլեկտրոնային իրականացումը: Մասնավորապես, այն թույլ է տալիս հեշտությամբ կիրառել ձևափոխած SAFER+ և SAFER-256 համակարգերը ինչպես 8-բիթ, այնպես էլ 32-բիթ միկրոպրոցեսորների վրա:

Հարկ է նշել, որ 45 հիմքով աստիճանային և լոգարիթմական ֆունկցիաների համապատասխան աղյուսակները կազմելու ընթացքում կատարվում են ըստ մոդուլ 257-ի որոշ գործողություններ ամբողջ թվերի հետ, բայց և այնպես այդ գործողությունները կատարվում են մեկ անգամ:

1.5.3. Խմբային գործողություններ ռաունդի սկզբում

Ինչպես երևում է նկար 8-ից և նկար 11-ից, ձևափոխած SAFER+ և SAFER-256 համակարգերի առաջին գործողությունը առաջին ենթաբանալու և բաց տեքստի «գումար» գործողությունն է, որը կատարվում է այնպես ինչպես նկարագրված է համապատասխանաբար 1.3 և 1.4.1 պարագրաֆներում: Այսպիսի «գումար» գործողությունը իրենից ներկայացնում է խմբայի գործողություն համապատասխանաբար 16 և 32 բայթ երկարության վեկտորների բազմության վրա սահմանված (վեկտորների կոմպոնենտները $\mathbb{Z}_{256}$ օղակից են): Ընդ որում առաջին ենթաբանալին SAFER-256 համակարգի դեպքում իրենից ներկայացնում է հենց օգտագործողի կողմից ընտրված բանալին, իսկ ձևափոխած SAFER+ համակարգի դեպքում՝ օգտագործողի կողմից ընտրված բանալու առաջին 16 բայթերը, քանի որ համակարգը ունի բանալու երեք դեպք (128, 192 կամ 256 բիթ):

Եթե օգտագործողը բանալիները միշտ ընտրում է կամայականության սկզբունքով, ապա կամայականորեն ընտրված բաց տեքստի համար խմբային գործողության արդյունքը հավասար հավանականությամբ կարող է լինել 16 բայթ երկարության բոլոր հնարավոր 2^{128} բլոկներից մեկը՝ ձևափոխած SAFER+ համակարգի դեպքում, իսկ SAFER-256 համակարգի դեպքում այն հավասար հավանականությամբ կարող է լինել 32 բայթ երկարության բոլոր հնարավոր 2^{256} բլոկներից մեկը: Դա նշանակում է, որ խմբային գործողության արդյունքը վիճակագրորեն անկախ է բաց տեքստի բլոկից, հետևաբար այսպիսի խմբային գործողությունը ապահովում է իդեալական գաղտնիություն Շենոնի իմաստով [29], երբ բանալին օգտագործվում է միայն մեկ անգամ: Ակնհայտ է, որ ձևափոխած SAFER+ և SAFER-256 համակարգերում միևնույն բանալին կարող է օգտագործվել բազմաթիվ իրարից տարբեր բլոկների ծածկագրման համար, բայց և այնպես փորձը ցույց է տալիս, որ այսպիսի գործողության ներմուծումը ծածկագրական համակարգերում ապահովում է կատարյալ գաղտնիություն:

1.5.4. Երկու ադիտիվ խմբերի կիրառումը

Ենթաբանալիների ներմուծման խմբային «գումար» գործողությունը ձևափոխած SAFER+ և SAFER-256 համակարգերում բաղկացած է երկու իրարից տարբեր խմբային գործողություններից. ըստ մոդուլ 256-ի բայթային գումար գործողություն և ըստ մոդուլ 2-ի բիթային գումար գործողություն: Իրարից տարբեր այս երկու խմբային գործողությունների օգտագործումը թույլ է տալիս ապահովել բայթերի լավ «ցրում»: Դիտարկենք օրինակ հետևյալ ցուցչային $\exp\text{tab}(\cdot)$ ֆունկցիան. $\exp(x) = 45^{x} mod\ 257$, որտեղ $0 \le x \le 255$ (պայմանով, որ երբ $x = 128$ ապա $45^{128} mod\ 257 = 256$ ընդունում է 0 արժեքը): Այս ֆունկցիան համապատասխանում է աշխատանքի բոլոր նկարներում առկա "exp" նշանակմանը: Այս ֆունկցիայի համար.

$$\exp(x_1 + x_2) = \exp(x_1) \times \exp(x_2),$$

որտեղ "+" նշանակում է գումար ըստ մոդուլ 256-ի, իսկ "×" արտադրյալ ըստ մոդուլ 257-ի: Այսինքն, երբ երկու բայթեր գումարվում են ըստ մոդուլ 256-ի և արդյունքի վրա կիրառվում է $\exp\text{tab}(\cdot)$ ֆունկցիան, ապա դա համարժեք է, երբ բայթերի վրա կիրառվում է $\exp\text{tab}(\cdot)$ ֆունկցիան, հետո արդյունքները բազմապատկվում են: Սակայն, երբ գումար գործողությունը բիթային ըստ մոդուլ 2-ի գործողությունն է, այդ հատկությունը չի պահպանվում.

$$\exp(x_1 \oplus x_2) \neq \exp(x_1) \times \exp(x_2),$$

որտեղ "$\oplus$" նշանակում է բիթային գումար ըստ մոդուլ 2-ի, իսկ "×" արտադրյալ ըստ մոդուլ 257-ի: Հետևաբար, ինչպես երևում է նկար 3-ից, նկար 8-ից և նկար 11-ից, բլոկի այն բայթերը, որոնց վրա կիրառվում է ցուցչային $\exp\text{tab}(\cdot)$ ֆունկցիան, ենթաբանալու համապատասխան բայթերը գումարվում են ըստ մոդուլ 2-ի, իսկ մնացած բայթերը՝ ըստ մոդուլ 256-ի: "+" և "$\oplus$" գործողությունների նկատմամբ $\exp\text{tab}(\cdot)$ ֆունկցիայի նման հատկությունը շատ նշանակալից է: Բլոկի և ենթաբանալու համապատասխան բայթերի գումարումը ըստ մոդուլ 256-ի նույնպես կարևոր է տարաբնույթ հարձակումների

նկատմամբ համակարգի կրիպտոկայունության մակարդակը բարձրացնելու համար (մասնավորապես գծային ծածկավերլուծության նկատմամբ): Ձևափոխած SAFER+ և SAFER-256 համակարգերում երկու իրարից տարբեր գործողությունների կիրառումը հնարավորություն է տալիս կատարել կրիպտոկայունության մասնակի վերլուծություն՝ գործողություններից յուրաքանչյուրի համար առանձին:

1.5.5. Աստիճանային և լոգարիթմական ֆունկցիաների կիրառումը ալգորիթմի ոչ գծային շերտում

Նկար 8-ից և 11-ից երևում է, որ ծածկագրման յուրաքանչյուր ռաունդում՝ ռաունդի առաջին ենթաբանալին «գումարելուց» հետո, բայթերի վրա կիրառվում է ցուցչային $\mathrm{exptab}(\cdot)$ (նկարներում համապատասխանում է "exp"-ին) կամ նրա հակադարձ լոգարիթմական $\mathrm{logtab}(\cdot)$ (նկարներում համապատասխանում է "log"-ին) ֆունկցիան: Ակնհայտ է, որ $\log(y) = \log_{45}(y)$, որտեղ $0 \leq y \leq 255$, $\log_{45}(0) = 128$: $\mathrm{exptab}(\cdot)$ և $\mathrm{logtab}(\cdot)$ փոխադարձաբար հակադարձ ֆունկցիաների ընտրությունը համակարգերի ոչ գծային շերտի համար, պատահական չի և պայմանավորված է մի քանի հանգամանքներով: Նախ և առաջ այս ֆունկցիաները հստակ սահմանված (well-defined or unambiguous) մաթեմատիկական ֆունկցիաներ են, որոնց կիրառությունը թույլ է տալիս խուսափել համակարգի ամրության կանխամտածված թուլությունից, ինչը կարող էր տեղի ունենալ պատահականորեն ընտրված ոչ գծային փոխադարձաբար հակադարձ ֆունկցիաների դեպքում: [20] աշխատանքում ցույց է տրված, որ $\mathrm{logtab}(\cdot)$ և $\mathrm{exptab}(\cdot)$ ֆունկցիաների առանձին բիթերը որոշող բուլյան ֆունկցիաների համար հանրահաշվական նորմալ տեսքում տվյալ ոչ գծային կարգի անդամների թիվը ունի նույն բաշխվածությունը, ինչ պատահականորեն ընտրված բուլյան ֆունկցիայի դեպքում: Ավելին, SAFER K-64 համակարգի վերլուծությունից հետո Վոդնեյը [32] եկել է այն եզրակացության, որ $\mathrm{logtab}(\cdot)$ և $\mathrm{exptab}(\cdot)$ ֆունկցիաների ընտրությունը որպես փոխադարձաբար հակադարձ ոչ գծային ֆունկցիաներ համակարգի

համար ճիշտ է, այն իմաստով, որ հակադարձ ոչ գծային ֆունկցիաների զգալի մասի կիրառման դեպքում հայտնի բաց տեքստ հարձակումը (known plaintext attack) կգործի ավելի արագ, քան բանալու համապարփակ որոնումը (exhaustive key search or brute force attack): Հայտնի բաց տեքստ հարձակումը դա ծածկագրական հարձակման տեսակ է, որի ժամանակ հարձակվողին հայտնի է բաց տեքստերի և համապատասխան ծածկագրերի N զույգ: Այս հարձակումը օգտագործվում է վերականգնելու համար համակարգի գաղտնի բանալին: Ժամանակակից ծածկագրաբանության գծային ծածկավերլուծությունը հայտնի բաց տեքստ հարձակում է:

1.5.6. Դիֆուզիայի ապահովումը մատրիցի միջոցով

SAFER ընտանիքի բլոկային ծածկագրական համակարգերում դիֆուզիան ապահովում է հակադարձելի գծային ձևափոխությունը (տե՛ս Նկար 3, 8, 11): Հակադարձելի գծային ձևափոխությունը հիմնված է պսևդո-Հադամարի 2-PHT կամ 4-PHT գծային ձևափոխության և բայթերի Armenian Shuffle տեղադրության վրա: Ձևափոխած SAFER+ և SAFER-256 համակարգերում կիրառվող 2-PHT ձևափոխությունը կատարվում է

$$H = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$$

մատրիցի միջոցով: 2-PHT ձևափոխությունը երկու հաջորդական բայթից բաղկացած $[a \;\; b]$ մուտքը փոխակերպում է երկու բայթից բաղկացած $[A \;\; B]$ ելքին հետևյալ կերպ՝

$$[A \;\; B] = [a \;\; b] \cdot H,$$

որտեղ թվաբանությունը կատարվում է ըստ մոդուլ 256-ի: 2-IPHT ձևափոխությունը 2-PHT ձևափոխության հակադարձ ձևափոխությունն է, հետևաբար վերծանման ալգորիթմի գծային ձևափոխության մատրիցը կլինի ծածկագրման ալգորիթմում օգտագործվող H մատրիցի հակադարձ մատրիցը.

$$H^{-1} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}:$$

2-IPHT ձևափոխությունը երկու բայթից բաղկացած $[A\ B]$ մուտքը փոխակերպում է երկու բայթից բաղկացած $[a\ b]$ ելքին հետևյալ կերպ՝

$$[a\ b] = [A\ B] \cdot H^{-1}:$$

Նկար 3-ից և 8-ից երևում է, որ SAFER+ և ձևափոխած SAFER+ ծածկագրական համակարգերի բայթային Armenian shuffle տեղադրությունները իրարից տարբեր են, ավելին երկրորդ համակարգում այն կիրառվում է ոչ թե 3, այլ 4 անգամ:

Ձևափոխած SAFER+ ալգորիթմի հակադարձելի գծային ձևափոխությունը բաղկացած է մուտքային 16 բայթերի [7, 12, 9, 14, 5, 8, 13, 10, 11, 4, 3, 6, 15, 2, 1, 16] տեղադրությունից, որին հաջորդում է 8 հատ 2-PHT ձևափոխություն (16 բայթերի չկրկնվող երկու հաջորդական բայթեր բազմապատկվում են պսևդո-Հադամարի մատրիցով): Նշված գործողությունները նույն հաջորդականությամբ կրկնվում են 4 անգամ (տե՛ս նկար 8):

SAFER-256 ծածկագրական համակարգի հակադարձելի գծային ձևափոխությունը, բաղկացած է մուտքային 32 բայթերի [24, 28, 29, 32, 17, 20, 21, 24, 13, 16, 9, 12, 5, 8, 1, 4, 3, 2, 7, 6, 11, 10, 15, 14, 27, 26, 31, 30, 19, 18, 23, 22] տեղադրությունից և նրան հաջորդող 2-PHT ձևափոխությունից (կատարվում է 16 հատ 2-PHT ձևափոխություն): Այս երկու գործողությունները նույն հաջորդականությամբ կրկնվում են 5 անգամ (տե՛ս նկար 11):

Նկար 14-ում պատկերված M մատրիցը SAFER-256 համակարգի հակադարձելի գծային ձևափոխության մատրիցն է, այսինքն հակադարձելի գծային ձևափոխությունը համարժեք է M մատրիցով բազմապատկմանը՝ $y = xM$, որտեղ $x = [x_1, x_2, x_3, \dots, x_{31}, x_{32}]$ վեկտորը հակադարձելի գծային ձևափոխության 32 բայթ երկարության մուտքն է, իսկ $y = [y_1, y_2, y_3, \dots, y_{31}, y_{32}]$ վեկտորը՝ ելքը:

$$M=\begin{vmatrix}
8 & 8 & 1 & 1 & 2 & 2 & 8 & 4 & 4 & 2 & 1 & 1 & 1 & 1 & 4 & 4 & 8 & 4 & 4 & 2 & 2 & 1 & 2 & 2 & 32 & 16 & 2 & 1 & 8 & 4 & 2 & 2 \\
4 & 2 & 4 & 2 & 8 & 4 & 2 & 2 & 1 & 1 & 16 & 8 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 4 & 4 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 4 & 2 \\
4 & 2 & 8 & 4 & 8 & 4 & 4 & 4 & 1 & 1 & 32 & 16 & 4 & 2 & 2 & 1 & 1 & 1 & 2 & 2 & 8 & 8 & 2 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 8 & 4 \\
4 & 4 & 1 & 1 & 1 & 1 & 8 & 4 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 4 & 2 & 4 & 2 & 2 & 1 & 1 & 1 & 16 & 8 & 2 & 1 & 4 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 & 1 & 1 & 32 & 16 & 2 & 1 & 4 & 4 & 2 & 2 & 1 & 1 & 2 & 1 & 8 & 4 & 8 & 4 & 1 & 1 & 8 & 4 & 4 & 2 & 4 & 2 & 8 & 8 \\
4 & 2 & 4 & 2 & 16 & 8 & 1 & 1 & 2 & 2 & 8 & 4 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 4 & 2 & 1 & 1 & 1 & 1 & 4 & 4 & 2 & 1 \\
8 & 4 & 4 & 2 & 32 & 16 & 1 & 1 & 4 & 4 & 8 & 4 & 2 & 1 & 4 & 2 & 2 & 2 & 1 & 1 & 2 & 2 & 8 & 4 & 2 & 2 & 1 & 1 & 8 & 8 & 2 & 1 \\
2 & 2 & 1 & 1 & 1 & 1 & 16 & 8 & 2 & 1 & 2 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 4 & 2 & 4 & 2 & 1 & 1 & 8 & 4 & 2 & 1 & 4 & 2 & 4 & 4 \\
2 & 2 & 1 & 1 & 8 & 8 & 4 & 2 & 8 & 4 & 1 & 1 & 2 & 2 & 2 & 2 & 4 & 2 & 8 & 4 & 2 & 1 & 4 & 4 & 8 & 4 & 2 & 1 & 32 & 16 & 1 & 1 \\
8 & 4 & 16 & 8 & 4 & 2 & 1 & 1 & 1 & 1 & 4 & 2 & 4 & 2 & 2 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 2 & 1 & 2 & 2 & 4 & 4 & 1 & 1 & 2 & 1 \\
8 & 4 & 32 & 16 & 4 & 2 & 2 & 2 & 1 & 1 & 8 & 4 & 8 & 4 & 2 & 1 & 1 & 1 & 4 & 4 & 2 & 2 & 2 & 1 & 2 & 2 & 8 & 8 & 1 & 1 & 4 & 2 \\
1 & 1 & 1 & 1 & 4 & 4 & 4 & 2 & 4 & 2 & 1 & 1 & 2 & 2 & 1 & 1 & 2 & 1 & 8 & 4 & 2 & 1 & 2 & 2 & 4 & 2 & 2 & 1 & 16 & 8 & 1 & 1 \\
1 & 1 & 4 & 4 & 2 & 2 & 8 & 4 & 2 & 1 & 2 & 2 & 8 & 8 & 1 & 1 & 2 & 1 & 32 & 16 & 4 & 2 & 1 & 1 & 4 & 2 & 8 & 4 & 8 & 4 & 2 & 2 \\
16 & 8 & 8 & 4 & 4 & 2 & 1 & 1 & 1 & 1 & 4 & 2 & 2 & 1 & 4 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 4 & 4 & 2 & 2 & 1 & 1 & 2 & 1 \\
32 & 16 & 8 & 4 & 8 & 4 & 1 & 1 & 2 & 2 & 4 & 2 & 2 & 1 & 8 & 4 & 4 & 4 & 1 & 1 & 1 & 1 & 4 & 2 & 8 & 8 & 2 & 2 & 2 & 2 & 2 & 1 \\
1 & 1 & 2 & 2 & 2 & 2 & 4 & 2 & 2 & 1 & 1 & 1 & 4 & 4 & 1 & 1 & 2 & 1 & 16 & 8 & 2 & 1 & 1 & 1 & 4 & 2 & 4 & 2 & 8 & 4 & 1 & 1 \\
1 & 1 & 2 & 2 & 1 & 1 & 8 & 4 & 4 & 2 & 8 & 8 & 2 & 2 & 2 & 2 & 8 & 4 & 4 & 2 & 32 & 16 & 1 & 1 & 2 & 1 & 8 & 4 & 2 & 1 & 4 & 4 \\
2 & 1 & 4 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 16 & 8 & 8 & 4 & 2 & 2 & 4 & 4 & 1 & 1 & 4 & 2 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 2 \\
2 & 1 & 8 & 4 & 2 & 1 & 2 & 2 & 1 & 1 & 4 & 2 & 32 & 16 & 8 & 4 & 2 & 2 & 8 & 8 & 2 & 2 & 4 & 2 & 1 & 1 & 4 & 4 & 1 & 1 & 8 & 4 \\
1 & 1 & 1 & 1 & 1 & 1 & 4 & 2 & 4 & 2 & 4 & 4 & 1 & 1 & 2 & 2 & 8 & 4 & 2 & 1 & 16 & 8 & 1 & 1 & 2 & 1 & 4 & 2 & 2 & 1 & 2 & 2 \\
4 & 4 & 1 & 1 & 2 & 2 & 2 & 1 & 8 & 4 & 2 & 2 & 1 & 1 & 8 & 8 & 32 & 16 & 2 & 1 & 8 & 4 & 2 & 2 & 8 & 4 & 4 & 2 & 4 & 2 & 1 & 1 \\
4 & 2 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 8 & 4 & 16 & 8 & 4 & 4 & 2 & 2 & 1 & 1 & 4 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 4 & 2 \\
8 & 4 & 2 & 1 & 4 & 2 & 1 & 1 & 2 & 2 & 2 & 1 & 8 & 4 & 32 & 16 & 8 & 8 & 2 & 2 & 1 & 1 & 8 & 4 & 4 & 4 & 1 & 1 & 2 & 2 & 4 & 2 \\
2 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 4 & 2 & 2 & 2 & 1 & 1 & 4 & 4 & 16 & 8 & 2 & 1 & 8 & 4 & 1 & 1 & 4 & 2 & 4 & 2 & 2 & 1 & 1 & 1 \\
1 & 1 & 8 & 8 & 1 & 1 & 4 & 2 & 8 & 4 & 2 & 2 & 4 & 4 & 1 & 1 & 4 & 2 & 8 & 4 & 8 & 4 & 2 & 2 & 2 & 1 & 32 & 16 & 2 & 1 & 2 & 2 \\
2 & 1 & 2 & 1 & 2 & 1 & 4 & 4 & 2 & 2 & 4 & 2 & 4 & 2 & 4 & 2 & 1 & 1 & 1 & 1 & 2 & 2 & 8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 16 & 8 \\
2 & 1 & 4 & 2 & 2 & 1 & 8 & 8 & 2 & 2 & 8 & 4 & 8 & 4 & 4 & 2 & 1 & 1 & 2 & 2 & 4 & 4 & 8 & 4 & 1 & 1 & 2 & 2 & 1 & 1 & 32 & 16 \\
1 & 1 & 4 & 4 & 1 & 1 & 2 & 1 & 8 & 4 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 2 & 4 & 2 & 4 & 2 & 2 & 2 & 2 & 1 & 16 & 8 & 2 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 & 4 & 4 & 2 & 1 & 32 & 16 & 1 & 1 & 1 & 1 & 2 & 2 & 8 & 4 & 2 & 1 & 4 & 2 & 8 & 8 & 4 & 2 & 8 & 4 & 8 & 4 & 1 & 1 \\
2 & 1 & 2 & 1 & 4 & 2 & 2 & 2 & 4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 16 & 8 & 1 & 1 & 1 & 1 & 2 & 2 & 8 & 4 \\
4 & 2 & 2 & 1 & 8 & 4 & 2 & 2 & 8 & 8 & 2 & 1 & 4 & 2 & 8 & 4 & 2 & 2 & 1 & 1 & 1 & 1 & 32 & 16 & 2 & 2 & 1 & 1 & 4 & 4 & 8 & 4 \\
1 & 1 & 2 & 2 & 2 & 2 & 2 & 1 & 16 & 8 & 1 & 1 & 11 & 1 & 1 & 1 & 4 & 2 & 2 & 1 & 4 & 2 & 4 & 4 & 2 & 1 & 8 & 4 & 4 & 2 & 1 & 1
\end{vmatrix}$$

Նկար 14: SAFER-256 համակարգի գծային ձևափոխության մատրիցը:

Ձևափոխած SAFER+ համակարգի հակադարձելի գծային ձևափոխությունը համարժեք է հետևյալ մատրիցով բազմապատկմանը՝

$$M = \begin{pmatrix} 16 & 8 & 1 & 1 & 4 & 4 & 4 & 2 & 2 & 2 & 4 & 2 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 & 4 & 2 & 1 & 1 & 2 & 2 & 8 & 4 \\ 4 & 2 & 1 & 1 & 2 & 2 & 2 & 1 & 4 & 4 & 16 & 8 & 4 & 2 & 1 & 1 \\ 2 & 2 & 2 & 1 & 2 & 1 & 1 & 1 & 8 & 4 & 1 & 1 & 1 & 1 & 4 & 2 \\ 2 & 2 & 4 & 2 & 16 & 8 & 1 & 1 & 4 & 2 & 4 & 4 & 1 & 1 & 2 & 1 \\ 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 2 & 2 & 8 & 4 & 4 & 2 & 1 & 1 \\ 4 & 2 & 4 & 4 & 1 & 1 & 16 & 8 & 1 & 1 & 2 & 1 & 4 & 2 & 2 & 2 \\ 1 & 1 & 4 & 2 & 8 & 4 & 1 & 1 & 2 & 1 & 2 & 2 & 1 & 1 & 2 & 1 \\ 2 & 1 & 16 & 8 & 1 & 1 & 4 & 4 & 1 & 1 & 4 & 2 & 2 & 2 & 4 & 2 \\ 1 & 1 & 1 & 1 & 4 & 2 & 2 & 1 & 2 & 1 & 1 & 1 & 8 & 4 & 2 & 2 \\ 4 & 4 & 4 & 2 & 2 & 1 & 2 & 2 & 16 & 8 & 1 & 1 & 1 & 1 & 4 & 2 \\ 4 & 2 & 2 & 2 & 1 & 1 & 8 & 4 & 1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 2 & 4 & 2 & 4 & 2 & 2 & 1 & 1 & 1 & 16 & 8 & 4 & 4 \\ 2 & 1 & 8 & 4 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 2 & 1 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 & 4 & 2 & 1 & 1 & 4 & 2 & 2 & 2 & 4 & 4 & 16 & 8 \\ 8 & 4 & 1 & 1 & 2 & 2 & 4 & 2 & 1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 \end{pmatrix}$$

Այստեղ ձևափոխության մուտքը և ելքը 16 բայթ երկարության վեկտորներ են:

Նկար 14-ում պատկերված SAFER-256 համակարգի հակադարձելի գծային ձևափոխության մատրիցի տեսքից երևում է, որ տողերից յուրաքանչյուրը պարունակում է առնվազն ութ հատ մեկ «1», հետևաբար յուրաքանչյուր մեկ հատ ոչ զրոյական բայթ պարունակող մուտքային բլոկի ելքային բլոկը կպարունակի նվազագույնը ութ հատ ոչ զրոյական բայթ: Իսկ ձևափոխած SAFER+ համակարգի մատրիցի տեսքից կարելի է եզրակացնել, որ յուրաքանչյուր մեկ հատ ոչ զրոյական բայթ պարունակող մուտքային բլոկի ելքային բլոկը կպարունակի նվազագույնը հինգ հատ ոչ զրոյական բայթ: Ակնհայտ է, որ SAFER-256 և ձևափոխած SAFER+ համակարգերի հակադարձելի գծային ձևափոխության հակադարձ ձևափոխության մատրից կհանդիսանա նրա մատրիցի հակադարձ մատրիցը:

Համակարգերի բայթային տեղադրությունների ընտրությունը պայմանավորված է նրանով, որ նրանք ապահովում են օպտիմալ դիֆուզիա համապատասխան ծածկագրական

համակարգերում, այսինքն կրիպտոկայունություն հնարավոր նվազագույն թվով ռաունդների կիրառմամբ, որը ստուգվել և հիմնավորվել է դիֆերենցիալ վերլուծության միջոցով (տե՛ս Գլուխ 2): Այս տեղադրություններից յուրաքանչյուրն անվանում են տվյալ համակարգի “Armenian shuffle”:

Այսպիսով, համակարգերում առկա բայթային տեղադրությունները, հետևաբար նաև մատրիցները ընտրվել են այնպես, որ համակարգերում դիֆուզիա լինի օպտիմալ (տե՛ս Պարագրաֆ 1.6):

1.5.7. Բանալիների հաջորդականության ընտրությունը

SAFER ընտանիքի առաջին SAFER K-64 համակարգի [20] ենթաբանալիների գեներացման ալգորիթմում միշտ ընտրվում էր բանալիների ռեգիստրի միևնույն բայթերը: Կնուդսենը աշխատանք [8]-ում ցույց է տվել, որ բանալիների ռեգիստրում բայթային նման «ստացիոնարությունը» վտանգավոր է: Ավելին Կնուդսենը նաև առաջարկել է ձևափոխություն ենթաբանալիների գեներացման ալգորիթմում, որը Մեսսին օգտագործել է SAFER ընտանիքի մյուս համակարգերում, այն բանից հետո, երբ Կնուդսենը նամակագրությամբ հաստատել է, որ ինքը հրաժարվում է այդ ձևափոխության նկատմամբ ունեցած «ինտելեկտուալ» սեփականության իրավունքից: Կնուդսենը առաջարկել է ընդլայնել բանալիների ռեգիստրը՝ ավելացնելով զույգության բայթ, որը օգտագործողի կողմից ընտրված բանալու բոլոր բայթերի ըստ մոդուլ 2-ի գումարն է: Բանալիների ռեգիստրի նման ընդլայնումը առավելություն է այն իմաստով, որ եթե ընտրվում է երկու իրարից տարբեր բանալիներ, ապա նրանց համապատասխան ռեգիստրները միմիյանցից կտարբերվեն նվազագույնը երկու բայթերով, որոնցից մեկը զույգության բայթն է: Այս ձևափոխությունը և [7] աշխատանքը էական տարբերություն են մտցնում ենթաբանալիների գեներացման ալգորիթմի միջոցով գեներացվող բանալիների հաւջորդականությունների մեջ (տե՛ս Նկար 5, 6, 13):

1.5.8. Ռաունդների քանակը

Իտերատիվ բլոկային ծածկագրական համակարգի ռաունդը բաց տեքստի նկատմամբ մի քանի անգամ կիրառվող ձևափոխությունն է: Ռաունդների քանակը որոշվում է ծածկավերլուծությունների միջոցով և իրենից ներկայացնում է ռաունդ ֆունկցիայի կիրառման նվազագույն քանակը, որը հարկավոր է համակարգը հայտնի և արդյունավետ հարձակումների նկատմամբ կրիպտոկայուն և միևնույն ժամանակ մաքսիմալ արագագործ համարելու համար: Համակարգի արագագործությունը հակադարձ համեմատական է ռաունդների քանակին: Ինչպես արդեն նշվել է, ձևափոխած SAFER+ և SAFER-256 համակարգերի նկատմամբ արդյունավետ ծածկավերլուծաբանական հարձակումները դրանք դիֆերենցիալ և գծային վերլուծություններն են [1],[4]: SAFER+ համակարգը ձևափոխություններից հետո ունի համարժեք կրիպտոկայունություն (Գլուխ 2, 3), հետևաբար ձևափոխած SAFER+ համակարգի ռաունդների թիվը 128 բիթ երկարության բանալու դեպքում 6 է, 192 բիթ երկարության բանալու դեպքում 8 է, իսկ 256 բիթ երկարության բանալու դեպքում 9, ինչպես SAFER+ համակարգինը: Ինչ վերաբերվում է SAFER-256 համակարգին, ապա այն ունի միայն 256 բիթ երկարության բանալու դեպքը և դիֆերենցիալ վերլուծության նկատմամբ կայուն է նվազագույնը 5 ռաունդի դեպքում, իսկ գծային վերլուծության նկատմամբ՝ 3 ռաունդի դեպքում (տե՛ս Գլուխ 2): Այսպիսով՝ 256 բիթ բլոկի և բանալու երկարությամբ SAFER-256 համակարգը կազմված է 6 ռաունդից:

1.6. Դիֆուզիայի օպտիմալությունը ձևափոխած SAFER+ և SAFER-256 բլոկային ծածկագրական համակարգերում

SAFER ընտանիքի բլոկային ծածկագրական համակարգերի դիֆերենցիալ վերլուծության նպատակն է r ռաունդից կազմված համակարգի համար գտնել $(r-1)$-ռաունդ քվազի-դիֆերենցիալների շղթաներ, որոնց անցումային հավանականությունները մեծ է կամ հավասար $\frac{1}{2^n-1} \approx 2^{-n}$, որտեղ n-ը բլոկի երկարությունն է (128 բիթ կամ 256 բիթ) (տե՛ս Գլուխ 2): Հետևաբար որպեսզի համակարգը լինի կայուն դիֆերենցիալ վերլուծության նկատմամբ հարկավոր է, որ ռաունդի փոքր կշռով մուտքի դեպքում ստանալ հնարավորինս ավելի մեծ կշռով ելք, այսինքն ապահովել լավ դիֆուզիա: Ինչպես արդեն նշվել է պարագրաֆ 1.5-ում դիֆուզիան համակարգում ապահովում է ռաունդի հակադարձելի գծային ձևափոխությունը: Եթե համակարգը կայուն է դիֆերենցիալ վերլուծության նկատամամբ հնարավորինս մինիմալ թվով ռաունդներից հետո, հետևաբար հնարավորինս ավելի արագագործ է, ապա կասենք, որ համակարգում ապահովվել է օպտիմալ դիֆուզիա:

Այս պարագրաֆի հիմնական նպատակն է հետազոտել և ցույց տալ թե, որ դեպքերում է հնարավոր ապահովել օպտիմալ դիֆուզիա SAFER ընտանիքի SAFER+, ձևափոխած SAFER+ և SAFER-256 բլոկային ծածկագրական համակարգերում: Հարկ է նշել, որ բոլոր հետազոտությունները և եզրակացությունները կատարվել են C++ ծրագրային լեզվով ստեղծված ծրագրային փաթեթների միջոցով (տե՛ս Հավելված 2, 3):

1.6.1. Դիֆուզիայի օպտիմալությունը ձևափոխած SAFER+ համակարգում

SAFER+ համակարգի Armenian shuffle բայթային տեղադրությունը (տե՛ս Նկար 3) համակարգում ապահովում է օպտիմալ դիֆուզիա [21],[22]: Սական դիֆերենցիալ վերլուծության համար ատենախոսության շրջանակներում ստեղծված ծրագրային փաթեթի (տե՛ս Հավելված 3) միջոցով կատարված ուսումնասիրությունները ցույց են տվել, որ համակարգը կայուն է դիֆերենցիալ վերլուծության նկատմամբ մինիմալ թվով ռաունդներից հետո ոչ միայն SAFER+ համակարգում օգտագործված բայթային տեղադրության [21] այլ նաև մի շարք բայթային տեղադրությունների դեպքում: Այսինքն գոյություն ունեն այնպիսի տեղադրություններ, որոնց մի մասը համակարգում ապահովում է լավ դիֆուզիա, իսկ մյուս մասը օպտիմալ դիֆուզիա: Օպտիմալ դիֆուզիա ապահովող տեղադրությունները, որոնք SAFER+ համակարգի Armenian shuffle տեղադրության համարժեք տեղադրություններ են, այն իմաստով, որ, եթե համակարգում օգտագործվող 16 բայթերի տեղադրությունը փոխարինենք նրանցից որևէ մեկով դրանից համակարգի կրիպտոկայունությունը և արագագործությունը չի փոխվի, մենք կանվանենք Armenian shuffle: Այսպիսով՝ փոխելով SAFER+ ալգորիթմի բայթային տեղադրությունը կստանենք նրան համարժեք մեկ այլ նոր բլոկային ծածկագրական համակարգ:

Եթե մատրիցը 0 չի պարունակում, ապա այն ապահովում է բանալու բայթերի լավ խառնվածություն ու դիֆուզիա և վերջին ռաունդ հարձակումը (last round attack) դարձնում է անհնարին [4]: Ինչպես արդեն նշվել է գլուխ 1-ի 5-րդ մասում լավ դիֆուզիա են ապահովում 16 բայթերի այն տեղադրությունները, որոնց համապատասխան հակադարձելի գծային ձևափոխության մատրիցի յուրաքանչյուր տող պարունակում է գոնե հինգ հատ «1» (բոլոր կենտ ինդեքսով տողերը պարունակում են հենց հինգ հատ «1», իսկ զույգ ինդեքսով տողերը՝ ութ հատ «1»), որը նշանակում է որ մուտքային մեկ բայթի փոփոխությունը կազդի ելքային բայթերից ամենաքիչը հինգի վրա: Օրինակ 7, 6, 15, 2, 3, 14, 5, 8, 1, 16, 9, 12, 11, 10, 13, 4 տեղադրության դեպքում հակադարձելի գծային

ձևափոխության հետևաբար նաև ռաունդի $Y = Y1Y2Y3\ldots Y16$ ելքը ձևավորելու համար $X = X1X2X3\ldots X16$ մուտքը բազմապատկվում է հակադարձելի գծային ձևափոխության համապատասխան մատրիցով.

$$\begin{pmatrix} Y1 \\ Y2 \\ Y3 \\ Y4 \\ Y5 \\ Y6 \\ Y7 \\ Y8 \\ Y9 \\ Y10 \\ Y11 \\ Y12 \\ Y13 \\ Y14 \\ Y15 \\ Y16 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 16 & 8 & 4 & 2 & 2 & 2 & 1 & 1 & 2 & 1 & 4 & 4 & 4 & 2 \\ 1 & 1 & 8 & 4 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 2 & 2 & 4 & 2 \\ 2 & 2 & 2 & 1 & 4 & 2 & 1 & 1 & 4 & 4 & 16 & 8 & 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 1 & 4 & 2 & 1 & 1 & 2 & 2 & 8 & 4 & 1 & 1 & 2 & 1 \\ 2 & 1 & 4 & 4 & 2 & 2 & 4 & 2 & 4 & 2 & 1 & 1 & 16 & 8 & 1 & 1 \\ 2 & 1 & 2 & 2 & 1 & 1 & 2 & 1 & 4 & 2 & 1 & 1 & 8 & 4 & 1 & 1 \\ 4 & 4 & 4 & 2 & 2 & 1 & 1 & 1 & 2 & 2 & 4 & 2 & 1 & 1 & 16 & 8 \\ 2 & 2 & 4 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 8 & 4 \\ 1 & 1 & 4 & 2 & 16 & 8 & 4 & 4 & 1 & 1 & 4 & 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 2 & 1 & 8 & 4 & 2 & 2 & 1 & 1 & 4 & 2 & 1 & 1 & 2 & 1 \\ 4 & 2 & 2 & 2 & 4 & 4 & 16 & 8 & 2 & 1 & 1 & 1 & 4 & 2 & 1 & 1 \\ 4 & 2 & 1 & 1 & 2 & 2 & 8 & 4 & 2 & 1 & 1 & 1 & 2 & 1 & 1 & 1 \\ 16 & 8 & 1 & 1 & 1 & 1 & 4 & 2 & 4 & 2 & 2 & 2 & 2 & 1 & 4 & 4 \\ 8 & 4 & 1 & 1 & 1 & 1 & 4 & 2 & 2 & 1 & 1 & 1 & 2 & 1 & 2 & 2 \\ 4 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 16 & 8 & 4 & 4 & 4 & 2 & 2 & 2 \\ 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 8 & 4 & 2 & 2 & 4 & 2 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} X1 \\ X2 \\ X3 \\ X4 \\ X5 \\ X6 \\ X7 \\ X8 \\ X9 \\ X10 \\ X11 \\ X12 \\ X13 \\ X14 \\ X15 \\ X16 \end{pmatrix} (\mathrm{mod}\ 256):$$

Ծրագրային հետազոտությունները ցույց են տվել, որ վերոնշյալ հատկությամբ մատրիցներ ստացվում են, եթե կենտ և զույգ ինդեքսներով բայթերը խառնվում են իրար մեջ. այսինքն կենտ ինդեքսով բայթը տեղափոխվում է միմիայն կենտ ինդեքսով բայթի հետ, իսկ զույգ ինդեքսով բայթը՝ զույգ ինդեքսով բայթի հետ: Գոյություն ունի այդպիսի $(8!)^2 = 1\,625\,702\,400$ բայթային տեղադրություն, սակայն դրանցից միայն 967 680-ի դեպքում կարելի է ստանել վերոնշյալ հատկությամբ մատրից: Դիֆերենցիալ վերլուծութունը ցույց է տալիս, որ այդ տեղադրությունների կիրառման դեպքում SAFER+ համակարգի ալգորիթմը կրիպտոկայուն է նվազագույնը 5 կամ նվազագույնը 6 ռաունդի դեպքում: Մասնավորապես SAFER+ համակարգում օգտագործված բայթային տեղադրությունը կրիպտոկայունություն ապահովում է 5 ռաունդի դեպքում (բանալու 128 բիթ երկարության դեպքի համար):

SAFER+ համակարգի հակադարձելի գծային ձևափոխուխությունը սկսվում է 2-PHT ձևափոխությամբ (տե՛ս Նկար 3), իսկ ձևափոխած SAFER+ համակարգում 2-PHT

ձևափոխությունից առաջ կատարվում է բայթերի տեղադրություն (տե՛ս Նկար 8): Հետևաբար հակադարձելի գծային ձևափոխության մատրիցները նախորդ օրինակում բերված 7, 6, 15, 2, 3, 14, 5, 8, 1, 16, 9, 12, 11, 10, 13, 4 տեղադրության դեպքում տարբեր են և ձևափոխած SAFER+ ալգորիթմի դեպքում նրա համապատասխան մատրիցը ունի հետևյալ տեսքը.

$$\begin{pmatrix}
1 & 1 & 4 & 2 & 4 & 4 & 2 & 1 & 1 & 1 & 2 & 2 & 4 & 2 & 16 & 8 \\
8 & 4 & 1 & 1 & 4 & 2 & 2 & 2 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 1 \\
4 & 2 & 4 & 4 & 2 & 1 & 2 & 2 & 16 & 8 & 4 & 2 & 1 & 1 & 1 & 1 \\
2 & 2 & 2 & 1 & 1 & 1 & 8 & 4 & 1 & 1 & 1 & 1 & 4 & 2 & 2 & 1 \\
2 & 1 & 1 & 1 & 4 & 2 & 1 & 1 & 4 & 2 & 16 & 8 & 4 & 4 & 2 & 2 \\
1 & 1 & 2 & 1 & 1 & 1 & 4 & 2 & 1 & 1 & 2 & 2 & 8 & 4 & 2 & 1 \\
1 & 1 & 2 & 1 & 2 & 2 & 4 & 2 & 1 & 1 & 4 & 4 & 16 & 8 & 4 & 2 \\
2 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 4 & 2 & 8 & 4 & 2 & 2 & 1 & 1 \\
2 & 2 & 16 & 8 & 1 & 1 & 4 & 2 & 4 & 4 & 1 & 1 & 2 & 1 & 4 & 2 \\
4 & 2 & 1 & 1 & 8 & 4 & 1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 & 2 & 2 \\
4 & 2 & 1 & 1 & 16 & 8 & 1 & 1 & 2 & 1 & 4 & 2 & 2 & 2 & 4 & 4 \\
1 & 1 & 8 & 4 & 1 & 1 & 2 & 1 & 2 & 2 & 1 & 1 & 2 & 1 & 4 & 2 \\
4 & 4 & 4 & 2 & 1 & 1 & 16 & 8 & 2 & 2 & 1 & 1 & 4 & 2 & 2 & 1 \\
2 & 1 & 2 & 2 & 2 & 1 & 1 & 1 & 8 & 4 & 4 & 2 & 1 & 1 & 1 & 1 \\
16 & 8 & 2 & 2 & 4 & 2 & 4 & 4 & 4 & 2 & 2 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 4 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 8 & 4
\end{pmatrix}$$

Ատենախոսության շրջանակներում կատարված հետազոտությունները ցույց են տվել, որ SAFER+ և ձևափոխած SAFER+ ծածկագրական համակարգերում լավ դիֆուզիա ապահովող տեղադրությունների բազմությունը համընկնում է.

1, 10, 3, 14, 7, 12, 5, 16, 11, 8, 13, 4, 15, 6, 9, 2

1, 16, 15, 2, 13, 10, 11, 6, 9, 14, 7, 4, 3, 8, 5, 12

3, 10, 1, 14, 5, 12, 7, 16, 11, 6, 13, 2, 15, 8, 9, 4

3, 16, 15, 4, 13, 10, 11, 6, 9, 14, 7, 2, 1, 8, 5, 12

5, 4, 1, 12, 3, 8, 7, 16, 9, 14, 13, 10, 15, 6, 11, 2

5, 14, 15, 10, 13, 6, 11, 4, 9, 16, 7, 2, 3, 12, 1, 8

7, 10, 1, 14, 3, 12, 5, 16, 9, 8, 11, 4, 15, 6, 13, 2

7, 16, 15, 8, 13, 4, 11, 10, 9, 12, 5, 2, 3, 14, 1, 6

9, 2, 1, 8, 3, 16, 5, 12, 7, 14, 11, 6, 15, 4, 13, 10

9, 16, 15, 10, 13, 4, 11, 8, 7, 12, 5, 2, 3, 14, 1, 6

11, 4, 1, 8, 3, 12, 5, 14, 7, 2, 9, 16, 13, 6, 15, 10

11, 14, 15, 4, 13, 10, 9, 6, 7, 2, 5, 12, 3, 16, 1, 8

13, 2, 1, 10, 3, 14, 5, 16, 7, 12, 9, 4, 11, 8, 15, 6

13, 12, 15, 4, 11, 10, 9, 6, 7, 2, 3, 16, 5, 14, 1, 8

15, 2, 1, 12, 3, 16, 5, 8, 7, 6, 9, 14, 11, 4, 13, 10

15, 12, 13, 2, 11, 6, 9, 16, 7, 4, 3, 8, 1, 14, 5, 10

⋮

Սակայն դիֆերենցիալ վերլուծությունը ցույց է տվել, որ օպտիմալ դիֆուզիա ապահովող տեղադրությունները այդ երկու համակարգերի դեպքում տարբեր է: Օրինակ SAFER+ համակարգի տեղադրությունը (տե՛ս Նկար 3) ձևափոխած SAFER+ համակարգում նույն կրիպտոկայունությունը չի ապահովում այն իմաստով, որ առաջին համակարգը այդ տեղադրության դեպքում դիֆերենցիալ վերլուծության նկատմամբ կրիպտոկայուն է նվազագույնը 5 ռաունդի դեպքում, իսկ երկրորդ համակարգը՝ նվազագույնը 6 ռաունդի դեպքում: Այդ իսկ պատճառով՝ ձևափոխած SAFER+ համակարգում որպես Armenian shuffle ընտրվել է օպտիմալ դիֆուզիա ապահովող բայթային տեղադրություններից 7, 12, 9, 14, 5, 8, 13, 10, 11, 4, 3, 6, 15, 2, 1, 16 տեղադրությունը (տե՛ս Նկար 8):

Ռաունդի հակադարձելի գծային ձևափոխության շերտը համարժեք է այդ շերտի մուտքային վեկտորը հակադարձելի գծային ձևափոխության մատրիցով բազմապատկմանը: Հետևաբար ուսումնասիրել բայթային տեղադրությունը, նույնն է ուսումնասիրել հակադարձելի գծային ձևափոխության մատրիցը, առավել ևս որ գոյություն ունեն այնպիսի տեղադրություններ, որոնց համապատասխան մատրիցը նույնն է: SAFER+ ալգորիթմի դեպքում միևնույն մատրիցը կարող են նկարագրել չորս կամ մեկ տեղադրություն: Օրինակ.

9, 8, 7, 10, 3, 16, 5, 14, 15, 4, 11, 2, 13, 6, 1, 12

13, 6, 5, 14, 11, 2, 7, 10, 1, 12, 3, 16, 9, 8, 15, 4

15, 4, 3, 16, 7, 10, 11, 2, 9, 8, 5, 14, 1, 12, 13, 6

1, 12, 11, 2, 5, 14, 3, 16, 13, 6, 7, 10, 15, 4, 9, 8

բայթային տեղադրությունների դեպքում հակադարձելի գծային ձևափոխության մատրիցը նույնն է, իսկ

13, 4, 15, 14, 11, 16, 9, 8, 1, 6, 3, 12, 5, 2, 7, 10

տեղադրությունը այն տեղադրություններից է որը ապահովում է լավ դիֆուզիա և նրա համապատասխան հակադարձելի գծային ձևափոխության մատրիցը այլևս ոչ մի տեղադրության դեպքում հնարավոր չի ստանալ: Ձևափոխած SAFER+ ալգորիթմի դեպքում պատկերը այլ է, այն իմաստով որ միևնույն մատրիցը ստացվում է ութ, երկու կամ մեկ տեղադրության դեպքում, օրինակ

1. 3, 14, 1, 8, 5, 12, 9, 16, 13, 4, 7, 2, 11, 6, 15, 10

   5, 12, 15, 10, 3, 14, 7, 2, 11, 6, 9, 16, 13, 4, 1, 8

   7, 2, 13, 4, 9, 16, 5, 12, 1, 8, 3, 14, 15, 10, 11, 6

   9, 16, 11, 6, 7, 2, 3, 14, 15, 10, 5, 12, 1, 8, 13, 4

   11, 6, 9, 16, 13, 4, 1, 8, 5, 12, 15, 10, 3, 14, 7, 2

   13, 4, 7, 2, 11, 6, 15, 10, 3, 14, 1, 8, 5, 12, 9, 16

   15, 10, 5, 12, 1, 8, 13, 4, 9, 16, 11, 6, 7, 2, 3, 14

   1, 8, 3, 14, 15, 10, 11, 6, 7, 2, 13, 4, 9, 16, 5, 12
2. 7, 2, 1, 8, 9, 4, 11, 6, 5, 12, 13, 16, 15, 14, 3, 10

   15, 14, 13, 16, 5, 12, 3, 10, 9, 4, 1, 8, 7, 2, 11, 6
3. 7, 12, 9, 14, 5, 8, 13, 10, 11, 4, 3, 6, 15, 2, 1, 16

Հետևաբար ավելի նպատակահարմար է համակարգերի կրիպտոկայունությունը ուսումնասիրել ոչ թե առանձին տեղադրությունների, այլ հակադարձելի գծային ձևափոխության մատրիցների միջոցով:

1.6.2. Դիֆուզիայի օպտիմալությունը SAFER-256 համակարգում

SAFER-256 ալգորիթմում լավ դիֆուզիա (կրիպտոկայունություն դիֆերենցիալ վերլուծության նկատմամբ մինիմալ թվով ռաունդներից հետո) ապահովում են այն մատրիցները, որոնց յուրաքանչյուր տողում առկա է նվազագույնը ութ հատ «1» (կենտ ինդեքսով տողերում կա ճիշտ ութ հատ «1», իսկ զույգ ինդեքսով տողերում ՝ 13 հատ) (օրինակ տե՛ս Նկար 14), ինչը նշանակում է յուրաքանչյուր մեկ ոչ զրոյական մուտքային բայթով բլոկի ելքային բլոկը պարունակում է նվազագույնը 8 հատ ոչ զրոյական բայթ: Ավելին 0-ից տարբեր նվազագույն քանակով ելքային բայթեր են ստացվում այն դեպքում, երբ այդ մեկ 0-ից տարբեր ելքային բայթը հավասար է 128-ի և նրա համապատասխան ինդեքսը կենտ է: Նման հատկությամբ մատրիցներ կարելի է ստանալ այն դեպքում, եթե ռաունդի հակադարձելի գծային ձևափոխության բայթային տեղադրության մեջ կենտ ինդեքսով բայթերը խառնվեն իրար հետ, զույգ ինդեքսներովը՝ իրար հետ (օրինակ տե՛ս Նկար 11): Այդ դեպքում $(32!)^2$ բայթային տեղադրություններից կարելի է դիֆերենցիալ վերլուծության միջոցով առանձնացնել այն տեղադրությունները, որոնց դեպքում համակարգը կլինի կրիպտոկայուն դիֆերենցիալ վերլուծության նկատմամբ մինիմալ թվով ռաունդներից հետո: Համեմատաբար լավ դիֆուզիա ապահովող հետևյալ տեղադրություններից որոշները SAFER-256 ալգորիթմում ապահովում են օպտիմալ դիֆուզիա, որոնցից մեկը համակարգի Armenian Shuffle տեղադրությունն է (տե՛ս Նկար 11).

17 20 21 24 25 28 29 32 13 16 9 12 5 8 1 4 3 2 7 6 11 10 15 14 19 18 23 22 27 26 31 30

17 20 21 24 25 28 29 32 13 16 9 12 5 8 1 4 3 2 7 6 11 10 15 14 19 18 23 22 31 30 27 26

17 20 21 24 25 28 29 32 13 16 9 12 5 8 1 4 3 2 7 6 11 10 15 14 23 22 19 18 31 30 27 26

17 20 21 24 25 28 29 32 13 16 9 12 5 8 1 4 3 2 7 6 15 14 11 10 23 22 19 18 31 30 27 26

17 20 21 24 25 28 29 32 9 12 13 16 1 4 5 8 7 6 3 2 15 14 11 10 23 22 19 18 31 30 27 26

17 20 21 24 29 32 25 28 9 12 13 16 1 4 5 8 7 6 3 2 15 14 11 10 23 22 19 18 31 30 27 26

21 24 17 20 29 32 25 28 9 12 13 16 1 4 5 8 7 6 3 2 15 14 11 10 23 22 19 18 31 30 27 26

21 24 17 20 25 28 29 32 13 16 9 12 5 8 1 4 3 2 7 6 11 10 15 14 19 18 23 22 27 26 31 30

21 24 17 20 29 32 25 28 13 16 9 12 5 8 1 4 3 2 7 6 11 10 15 14 19 18 23 22 27 26 31 30

25 28 29 32 17 20 21 24 13 16 9 12 5 8 1 4 3 2 7 6 11 10 15 14 19 18 23 22 27 26 31 30

25 28 29 32 17 20 21 24 13 16 9 12 5 8 1 4 3 2 7 6 11 10 15 14 27 26 31 30 19 18 23 22

25 28 29 32 17 20 21 24 5 8 1 4 13 16 9 12 11 10 15 14 3 2 7 6 27 26 31 30 19 18 23 22

13 16 9 12 5 8 1 4 17 20 21 24 25 28 29 32 19 18 23 22 27 26 31 30 3 2 7 6 11 10 15 14

3 2 7 6 11 10 15 14 19 18 23 22 27 26 31 30 17 20 21 24 25 28 29 32 13 16 9 12 5 8 1 4

7 6 3 2 11 10 15 14 19 18 23 22 27 26 31 30 17 20 21 24 25 28 29 32 13 16 9 12 5 8 1 4

7 6 3 2 11 10 15 14 19 18 23 22 27 26 31 30 17 20 21 24 25 28 29 32 13 16 9 12 1 4 5 8

11 10 15 14 3 2 7 6 19 18 23 22 27 26 31 30 17 20 21 24 25 28 29 32 13 16 9 12 5 8 1 4

11 10 15 14 3 2 7 6 19 18 23 22 27 26 31 30 17 20 21 24 25 28 29 32 5 8 1 4 13 16 9 12

11 10 15 14 3 2 7 6 19 18 23 22 27 26 31 30 17 20 21 24 25 28 29 32 5 8 1 4 13 16 9 12

11 10 15 14 3 2 7 6 23 22 19 18 27 26 31 30 17 20 21 24 25 28 29 32 5 8 1 4 13 16 9 12

11 10 15 14 3 2 7 6 23 22 19 18 31 30 27 26 21 24 17 20 29 32 25 28 5 8 1 4 13 16 9 12

21 24 17 20 29 32 25 28 5 8 1 4 13 16 9 12 11 10 15 14 3 2 7 6 23 22 19 18 31 30 27 26

17 20 21 24 29 32 25 28 5 8 1 4 13 16 9 12 11 10 15 14 3 2 7 6 23 22 19 18 31 30 27 26

17 20 21 24 29 32 25 28 5 8 1 4 13 16 9 12 11 10 15 14 3 2 7 6 23 22 19 18 27 26 31 30

21 20 17 24 25 28 29 32 13 16 9 12 5 8 1 4 3 2 7 6 11 10 15 14 19 18 23 22 27 26 31 30

17 24 21 20 25 28 29 32 13 16 9 12 5 8 1 4 3 2 7 6 11 10 15 14 19 18 23 22 27 26 31 30

17 20 21 24 25 28 29 32 13 16 9 12 5 8 1 4 3 2 7 6 11 14 15 10 19 18 23 22 27 26 31 30

⋮

ԳԼՈՒԽ 2

ՁԵՎԱՓՈԽԱԾ SAFER+ ԵՎ SAFER-256 ԾԱԾԿԱԳՐԱԿԱՆ ՀԱՄԱԿԱՐԳԵՐԻ ԴԻՖԵՐԵՆՑԻԱԼ ՎԵՐԼՈՒԾՈՒԹՅՈՒՆԸ

Դիֆերենցիալ վերլուծությունը մշակվել և առաջարկվել է Բիհամի ու Շամիրի կողմից 1990թ. [1] և իրենից ներկայացնում է հարձակում իտերատիվ բլոկային ծածկագրական համակարգերի նկատմամբ: Այն հանդիսանում է բաց տեքստի ընտրմամբ հարձակում (chosen plaintext attack). հարձակվողը կարող է ընտրել բաց տեքստ և ստանալ համապատասխան ծածկագիրը: Դիֆերենցիալ վերլուծությունը սիմետրիկ ծածկագրաբանության SAFER ընտանիքի իտերատիվ բլոկային ծածկագրական համակարգերի նկատամաբ արդյունավետ հարձակումներից մեկն է [8],[9],[20]: Իտերատիվ ծածկագրական համակարգի ռաունդի մուտքը և ելքը ընդունում են արժեքներ միևնույն բազմությունից, SAFER-256 համակարգի դեպքում այդ բազմությունը 32 երկարության վեկտորների բազմությունն է, իսկ ձևափոխած SAFER+ համակարգում՝ 16 երկարության: Երկու դեպքում էլ վեկտորների կոմպոնենտները, այսինքն բայթերի արժեքները $\mathbb{Z}_{256}$ օղակի էլեմենտներ են:

Այս պարագրաֆում ներկայացած են SAFER-256 և ձևափոխած SAFER+ համակարգերի կրիպտոկայունությունը դիֆերենցիալ վերլուծության նկատմամբ՝ [16],[20],[21] աշխատանքներում առկա նշանակումների և հասկացությունների հիման վրա:

2.1. Բայթային դիֆերենցիալների և քվազի-դիֆերենցիալների վերլուծությունը SAFER-256 համակարգում

Այս պարագրաֆում ներկայացված է SAFER-256 բլոկային ծածկագրական համակարգի բայթային դիֆերենցիալների և քվազի-դիֆերենցիալների վերլուծությունը՝ [21] աշխատանքում ներկայացված և SAFER ընտանիքի բոլոր ծածկագրական համակարգերի նկատմամբ լայնորեն կիրառվող բայթային դիֆերենցիալների և քվազի-դիֆերենցիալների վերլուծության հիման վրա:

Ինչպես երևում է նկար 11-ից, SAFER-256 համակարգի յուրաքնչյուր ռաունդի սկզբում 32 բայթ երկարության ռաունդի $X = [X_1, X_2, X_3, \dots, X_{32}]$ մուտքին բայթ առ բայթ «գումարվում» է ռաունդի 32 բայթ երկարության առաջին $K_1 = [K_{1,1}, K_{1,2}, K_{1,3}, \dots, K_{1,32}]$ ենթաբանալին՝ $X \otimes K_1$, որտեղ

$$\otimes = [\oplus, \oplus, \oplus, \oplus, +, +, +, +, \oplus, \oplus, \oplus, \oplus, +, +, +, +, \oplus, \oplus, \oplus, \oplus, +, +, +, +, \oplus, \oplus, \oplus, \oplus, +, +, +, +],$$

"$\oplus$" բիթային xor գործողությունն է, իսկ “+” ըստ մոդուլ 256-ի բայթային գումար գործողությունն է:

32 բայթ երկարության $V = [V_1, V_2, V_3, \dots, V_{32}]$ և $V' = [V'_1, V'_2, V'_3, \dots, V'_{32}]$ բայթային վեկտորների ΔV տարբերություն կամ դիֆերենցիալ սահմանվում է այսպես՝

$$\Delta V = V \ominus V' = [V_1 \oplus V'_1, V_2 \oplus V'_2, V_3 \oplus V'_3, V_4 \oplus V'_4, V_5 - V'_5, V_6 - V'_6, V_7 - V'_7, V_8 - V'_8,$$
$$V_9 \oplus V'_9, V_{10} \oplus V'_{10}, V_{11} \oplus V'_{11},\ V_{12} \oplus V'_{12}, V_{13} - V'_{13}, V_{14} - V'_{14}, V_{15} - V'_{15},\ V_{16} - V'_{16},$$
$$V_{17} \oplus V'_{17}, V_{18} \oplus V'_{18}, V_{19} \oplus V'_{19}, V_{20} \oplus V'_{20}, V_{21} - V'_{21},\ V_{22} - V'_{22}, V_{23} - V'_{23}, V_{24} - V'_{24},$$
$$V_{25} \oplus V'_{25}, V_{26} \oplus V'_{26}, V_{27} \oplus V'_{27}, V_{28} \oplus V'_{28}, V_{29} - V'_{29}, V_{30} - V'_{30}, V_{31} - V'_{31},\ V_{32} - V'_{32}],$$

իսկ $\widetilde{\Delta V}$ քվազի-տարբերությունը կամ քվազի-դիֆերենցիալը այսպես՝

$$\widetilde{\Delta V} = V - V' = [V_1 - V'_1, V_2 - V'_2, V_3 - V'_3, V_4 - V'_4, V_5 - V'_5, V_6 - V'_6, V_7 - V'_7, V_8 - V'_8, V_9 - V'_9, V_{10} -$$
$$V'_{10}, V_{11} - V'_{11},\ V_{12} - V'_{12}, V_{13} - V'_{13}, V_{14} - V'_{14}, V_{15} - V'_{15},\ V_{16} - V'_{16}, V_{17} - V'_{17}, V_{18} - V'_{18}, V_{19} -$$

$$V'_{19}, V_{20} - V'_{20}, V_{21} - V'_{21},\ V_{22} - V'_{22}, V_{23} - V'_{23}, V_{24} - V'_{24}, V_{25} - V'_{25}, V_{26} - V'_{26}, V_{27} - V'_{27}, V_{28} - V'_{28}, V_{29} - V'_{29}, V_{30} - V'_{30}, V_{31} - V'_{31},\ V_{32} - V'_{32}]:$$

Դիցուք $S = [S_1, S_2, S_3, \dots, S_{32}]$ վեկտորը ռաունդի հակադարձելի գծային ձևափոխության մուտքն է: Ինչպես երևում է նկար 11-ից.

$$S_j = 45^{(X_j \oplus K_{1,j})} + K_{2,j}, \qquad j \in \{1, 2, 3, 4, 9, 10, 11, 12, 17, 18, 19, 20, 25, 26, 27, 28\}, \tag{2.1}$$

$$S_j = \log_{45}(X_j + K_{1,j}) \oplus K_{2,j}, \quad j \in \{5, 6, 7, 8, 13, 14, 15, 16, 21, 22, 23, 24, 29, 30, 31, 32\}, \tag{2.2}$$

որտեղ $K_{1,j}$-ն և $K_{2,j}$-ն ռաունդի համապատասխանաբար առաջին՝ K_1, և երկրորդ՝ K_2 32 բայթ երկարության բանալիների բայթերն են: $1, 2, 3, 4, 9, 10, 11, 12, 17, 18, 19, 20, 25, 26, 27, 28$ բայթերը կդիտարկենք որպես ցուցչային բայթեր, իսկ $5, 6, 7, 8, 13, 14, 15, 16, 21, 22, 23, 24, 29, 30, 31, 32$ բայթերը՝ լոգարիթմական բայթեր:

$(\Delta X_j, \Delta S_j)$ զույգի (α, β) արժեքին կանվանենք ցուցչային բայթային դիֆերենցիալ, եթե $j \in \{1, 2, 3, 4, 9, 10, 11, 12, 17, 18, 19, 20, 25, 26, 27, 28\}$, իսկ եթե $j \in \{5, 6, 7, 8, 13, 14, 15, 16, 21, 22, 23, 24, 29, 30, 31, 32\}$, ապա՝ լոգարիթմական բայթային դիֆերենցիալ: Ինչպես երևում է դիֆերենցիալի և քվազի-դիֆերենցիալի սահմանումներից, ցուցչային բայթային քվազի-դիֆերենցիալը սահմանվում է ոչ թե xor գործողության միջոցով, ինչպես որ ցուցչային դիֆերենցիալի դեպքում, այլ ըստ մոդուլ 256-ի բայթային տարբերություն գործողության միջոցով:

Եթե $S = [S_1, S_2, S_3, \dots, S_{32}]$ վեկտորը ռաունդի հակադարձելի գծային ձևափոխության մուտքային վեկտորն է, ապա ռաունդի ելքային $Y = [Y_1, Y_2, Y_3, \dots, Y_{32}]$ վեկտորը կորոշվի այսպես.

$$Y = S \cdot M = SM,$$

որտեղ M-ը նկար 14-ում պատկերված SAFER-256 համակարգի հակադարձելի գծային ձևափոխության մատրիցն է. բոլոր գործողությունները կատարվում են ըստ մոդուլ 256-ի:

Հետևանք 2.1 [20]: SAFER ընտանիքի ծածկագրական համակարգը մարկովյան համակարգ է. կամայականորեն և իրարից անկախ ընտրված ռաունդի բանալիների

դեպքում, ռաունդի երկու իրարից տարբեր X և X^* մուտքերի և նրանց համապատասխան Y և Y^* ելքերի համար, $P(\Delta Y = \beta | \Delta X = \alpha,\ X = \gamma)$ պայամանական հավանականությունը անկախ է γ-ից:

Դիցուք $X(i)$-ն և $Y(i)$-ն SAFER-256 համակարգի i-րդ ռաունդի համապատասխանաբար մուտքային և ելքային վեկտորներն են: Դիֆերենցիալ վերլուծության ժամանակ, ինչպես միշտ, ենթադրվում է, որ ռաունդի ենթաբանալիները ընտրվում են կամայականորեն և իրարից անկախ (SAFER-256 համակարգի ենթաբանալիները գեներացվում են միևնույն գաղտնի բանալուց): Ենթադրենք ոչ զրոյական 32 երկարության բայթային (α, β) զույգը SAFER-256 համակարգի $(\Delta X(1), \Delta Y(i))$ i-ռաունդ դիֆերենցիալի արժեքն է:

Համաձայն հետևանք 2.1-ի SAFER-256 համակարգը հանդիսանում է մարկովյան շղթա, այսինքն

$$P(\Delta Y(i) = \beta | \Delta X(1) = \alpha,\ X(1) = \gamma)$$

պայմանական հավանականությունը անկախ է γ-ից: Հետևաբար

$$\Delta X(1), \Delta Y(1), \Delta Y(2), \dots, \Delta Y(i)$$

հաջորդականությունը հանդիսանում է մարկովյան շղթա (մարկովյան շղթան սահմանված է [16] աշխատանքում): Ենթադրենք $(\alpha, \beta_1, \beta_2, \dots, \beta_{i-1}, \beta)$ հաջորդականությունը

$$(\Delta X(1), \Delta Y(1), \Delta Y(2), \dots, \Delta Y(i))$$

շղթայի արժեքն է, որին անվանում են i-ռաունդ բնութագրիչ: i-ռաունդ դիֆերենցիալի

$$P(\Delta Y(i) = \beta | \Delta X(1) = \alpha)$$

հավանականությունը հավասար կլինի i-ռաունդ $(\alpha, \beta_1, \beta_2, \dots, \beta_{i-1}, \beta)$ բնութագրիչի $\beta_1, \beta_2, \dots, \beta_{i-1}$ հաջորդականության բոլոր հնարավոր ոչ զրոյական արժեքների

$$P(\Delta Y(1) = \beta_1, \dots, \Delta Y(i-1) = \beta_{i-1}, \Delta Y(i) = \beta | \Delta X(1) = \alpha)$$

հավանականությունների գումարին:

Եթե S և S' վեկտորները հակադարձելի գծային ձևափոխության երկու իրարից տարբեր մուտքային վեկտորներ են, ապա ռաունդի ելքային համապատասխան Y և Y' վեկտորների քվազի-դիֆերենցիալը կլինի

$$\widetilde{\Delta Y} = SM - S'M = (\widetilde{\Delta S})M: \tag{2.3}$$

Այս հավասարությունը այն հիմնական փաստն է, որի վրա հիմնվելով դիֆերենցիալ վերլուծության համար կարելի է օգտագործել բայթային քվազի-դիֆերենցիալը բայթային դիֆերենցիալի փոխարեն, ինչը հիմնավորում է հետևյալ հետևանքը:

Հետևանք 2.2 (Մեսսի, [20]): SAFER ընտանիքի բլոկային ծածկագրական համակարգում ռաունդի երկու իրարից տարբեր X և X^* մուտքերի դիֆերենցիալի և համապատասխան Y և Y^* ելքերի քվազի-դիֆերենցիալի համար $P(\widetilde{\Delta Y} = \beta|\Delta X = \alpha, X = \gamma)$ պայմանական հավանականությունը անկախ է γ-ից:

Բայթային դիֆերենցիալների և քվազի-դիֆերենցիալների հիմնական հատկությունները բերված են աղյուսակ 3-ում:Լոգարիթմական բայթային դիֆերենցիալիների համար, որտեղ մուտքային և ելքային տարբերությունները երկուսն էլ ըստ մոդուլ 256-ի են.

$$P(\Delta S = \tau|\Delta X = \alpha) = P(\Delta S = -\tau|\Delta X = -\alpha),$$

իսկ ցուցչային բայթային քվազի-դիֆերենցիալների համար, որտեղ միայն ելքն է ըստ մոդուլ 256-ի.

$$P(\Delta S = \tau|\Delta X = \alpha) = P(\Delta S = -\tau|\Delta X = \alpha):$$

Աղյուսակ 3-ում ներկայացված են $P(\Delta S = \tau|\Delta X = \alpha)$ և $P(\widetilde{\Delta S} = \tau|\Delta X = \alpha)$ անցումային հավանականությունների հաշվարկային արդյունքները, որոնք ստացվել են (2.1) և (2.2) բանաձևերի միջոցով: Որպես բանալիների բայթեր կամայականության սկզբունքով ընտրվել են $\mathbb{Z}_{256}$ օղակի 256 հնարավոր արժեքներից: Աղյուսակ 3-ում առկա $avg[P(\tau|\alpha)]$ նշանակումը հետևյալ արտահայտությունն է.

$$\frac{1}{2^8}\sum_{\tau=0}^{255} P(\tau|\alpha):$$

Աղյուսակ 3: Բայթային դիֆերենցիալների և քվազի-դիֆերենցիալների հիմնական հատկությունները:

Լոգարիթմական	Ցուցչային	Ցուցչային (քվազի)
Մուտքային տարբերություն. mod256 Ելքային տարբերություն. mod256	Մուտքային տարբերություն. xor Ելքային տարբերություն. xor	Մուտքային տարբերություն. xor Ելքային տարբերություն. mod 256
$P(\tau\|\alpha) = P(-\tau\|-\alpha)$		$P(\tau\|\alpha) = P(-\tau\|\alpha)$
$P(\tau\|\alpha) = 0;\ \tau, \alpha\epsilon\{\pm 64, 128, \}$	$P(\tau\|\alpha) = 0;\ \tau, \alpha\epsilon\{\pm 64, 128\}$	$P(\tau\|\alpha) = 0;\ \tau, \alpha\epsilon\{\pm 64, 128\}$
$P(128\|\alpha) = 2^{-7};$ α-ն կենտ է	$P(\tau\|128) = 2^{-7};$ τ-ն կենտ է	$P(\tau\|128) = 2^{-7};$ τ-ն կենտ է
$P(128\|\alpha) = 0;$ α-ն զույգ է		$avg[P(128\|\alpha)] = 2^{-8,2};$ α-ն կենտ է
$\max P(\tau\|\alpha) = 2^{-6,4};$ $(\alpha, \tau)\epsilon\{(128, 48), (128, -48)\}$	$\max P(\tau\|\alpha) = 2^{-5};$ $(\alpha, \tau)\epsilon$ $\{(-16,32), (103,64),$ $(18,28), (-108,128),$ $(48,128), (-78,128),$ $(54,128), (-115,128),$ $(-23,128), (102,128),$ $(-2,128), (103, -64)\}$	$\max P(\tau\|\alpha) = 2^{-4,7};$ $(\alpha, \tau)\epsilon\{(79, 68), (79, -68)\}$
$\max P(\tau\|\alpha) = 2^{-6,2};$ $(\alpha, \tau)\epsilon\{(48, -48)\}$	$\max P(128\|\alpha) = 2^{-5};$ $\alpha\ \epsilon\ \{18, 48, 54, 102, -115,$ $-108, -78, -23, -2\}$	$\max P(128\|\alpha) = 2^{-5};$ $\alpha\ \epsilon\ \{18, 48, 54, 102, -115,$ $-108, -78, -23, -2\}$
$avg[P(\tau\|\alpha)] = 2^{-7}$		$avg[P(\tau\|\alpha)] = 2^{-8}$

Դիֆերենցիալ վերլուծության ընթացքում կիրառվում են բայթային դիֆերենցիալների և քվազի-դիֆերենցիալների միջև գործող հետևյալ առնչությունները:

Հատկություն 2.1 [20]: Բայթային $\Delta V = V \ominus V'$ դիֆերենցիալի և $\widetilde{\Delta V} = V - V'$ քվազի-դիֆերենցիալի համար

1) $\widetilde{\Delta V} = 128$, այն և միայն այն դեպքում, երբ $\Delta V = 128$,

2) $\widetilde{\Delta V} = 0$, այն և միայն այն դեպքում, երբ $\Delta V = 0$,

3) $\widetilde{\Delta V}$-ն կենտ է այն և միայն այն դեպքում, երբ ΔV-ն կենտ է:

Ենթադրենք $\left(\Delta X(i), \widetilde{\Delta Y}(i)\right)$ զույգը i-րդ ռաունդի քվազի-դիֆերենցիալն է: $S(i)$-ով նշանակենք i-րդ ռաունդի գծային ձևափոխության մուտքը: $\left(\Delta X(i), \widetilde{\Delta Y}(i)\right)$ քվազի-դիֆերենցիալի մեջ $S(i)$-ի կարևորությունը շեշտելու համար այն վերանշանակենք այսպես՝ $\left(\Delta X(i), \widetilde{\Delta S}(i), \widetilde{\Delta Y}(i)\right)$, որտեղ $\widetilde{\Delta S}(i)$-ն i-րդ ռաունդի գծային ձևափոխության մուտքային քվազի-դիֆերենցիալն է: (2.3) արտահայտությունից հետևում է, որ

$$\widetilde{\Delta Y}(i) = \widetilde{\Delta S}(i):$$

$\Delta X(i)$-ից $\widetilde{\Delta Y}(i)$-ի անցման հավանականությունը, ոչ զրոյական արժեքների դեպքում, համարժեք է $\Delta X(i)$-ից $\widetilde{\Delta S}(i)$-ի անցման հավանականությանը, քանի որ $\widetilde{\Delta S}(i)$-ից $\widetilde{\Delta Y}(i)$-ի անցումը սահմանվում է ֆիքսված M մատրիցի միջոցով, այսինքն որոշվում է միարժեքորեն:

Հեշտ է տեսնել, որ $\Delta X(i)$-ից $\widetilde{\Delta S}(i)$-ի անցման հավանականությունը $\Delta X(i)$-ի և $\widetilde{\Delta S}(i)$-ի համապատասխան բայթերի դիֆերենցիալնների (լոգարիթմական բայթերի) և քվազի-դիֆերենցիալների (ցուցչային բայթերի) անցումային հավանականությունների արտադրյալն է: Հետևաբար անցումային հավանականությունը մեծանում է $\widetilde{\Delta S}(i)$ վեկտորի ոչ զրոյական բայթերի քանակի ավելացման զուգնթաց, որը սովորաբար համընկնում է $\Delta X(i)$ վեկտորի Հեմինգի կշռի (այսինքն վեկտորի ոչ զրոյական բայթերի քանակի) հետ:

Դիտարկենք հետևալ օրինակը.

$$\widetilde{\Delta S}(i) = V_2[1,5,9,15](a, -a, -b, b),$$

$$\Delta X(i) = V_0[1,5,9,15](a_1, a_2, a_3, a_4):$$

$V_2[1,5,9,15](a, -a, -b, b)$, $a \neq 0$ և $b \neq 0$ պարամետրերով, 4 կշռով բայթային վեկտոր է՝ 1, 5, 9, 15 ոչ զրոյական բայթերով և $a, -a, -b, b$ համապատասխան բայթերի արժեքներով: a և b պարամետրերը կարող են ընդունել 255 հնարավոր ոչ զրոյական արժեքներ: $V_0[1,5,9,15](a_1, a_2, a_3, a_4)$ զրո պարամետրերով, 4 կշռով բայթային վեկտոր է՝ 1, 5, 9, 15 ոչ զրոյական բայթերով և a_1, a_2, a_3, a_4 համապատասխան բայթերի արժեքներով: $P\left(\widetilde{\Delta S}(i)\middle|\Delta X(i)\right)$ անցումային հավանականությունը հաշվելու համար պետք է հաշվի առնել, որ 1 և 9 ցուցչային բայթերի համար միջին անցումային հավանականությունը հավասար է 2^{-8}, իսկ 5 և 15 լոգարիթմական բայթերի համար՝ 2^{-7}: Բացի այդ a և b պարամետրերից յուրաքանչյուրը կարող է ընդունել 2^8 հնարավոր արժեք, չնայած որ նախնական պայմանի համաձայն իրականում հնարավոր արժեքները $2^8 - 1$ հատ են. a և b պարամետրերը 0-ից տարբեր են: Այսպիսով $\Delta X(i)$-ից $\widetilde{\Delta S}(i)$-ի անցման հավանականությունը բերված կլինի.

$$P\left(\widetilde{\Delta S}(i)\middle|\Delta X(i)\right) \approx \sum_a \sum_b avg[P(a|a_1)]avg[P(-a|a_2)]avg[P(-b|a_3)]avg[P(b|a_4)]$$
$$\approx 2^8 2^8 (2^{-8} 2^{-7} 2^{-8} 2^{-7}) = 2^{-14}:$$

Անցումային հավանականությունը հաշվելիս մենք դիտավորյալ օգտագործում ենք 2^8 արժեքը՝ դիֆերենցիալ վերլուծության նկատմամբ «լրացուցիչ» կայունություն ապահովելու նկատառումով:

2.1.1. SAFER-256 համակարգի դիֆերենցիալ վերլուծությունը և դիֆերենցիալ շղթաները

Դիֆերենցիալ վերլուծությամբ հարձակումը, r ռաունդից կազմված SAFER-256 համակարգի վրա, հիմնված է հետևյալ հանգամանքի վրա, գտնել $(r-1)$-ռաունդ դիֆերենցիալ, որի հավանականությունը էականորեն ավելի մեծ է $\frac{1}{2^{256}-1} \approx 2^{-256}$ միջին հավանականությունից: Այսինքն դիֆերենցիալ ծածկագրավերլուծության հիմնական նպատակն է r ռաունդների համար գտնել մեծ անցումային հավանականությամբ քվազի-դիֆերենցիալներ: Այսպիսով՝ որպեսզի r ռաունդից կազմված SAFER-256 համակարգը լինի կայուն դիֆերենիալ վերլուծության նկատմամբ, պետք է գոյություն չունենա $(r-1)$-ռաունդ դիֆերենցիալ, որի հավանականությունը մեծ կամ հավասար լինի 2^{-256}-ի:

Դիֆերենցիալ վերլուծությունը ցույց է տվել, որ 32 բայթ երկարության բաց տեքստի ծածկագրման SAFER-256 համակարգի ամենամեծ անցումային հավանականությունը փոքր է 2^{-256}-ից, 5 և ավելի մեծ ռաունդ դիֆերենցիալ շղթաների ($r=5$ երկարության դիֆերենցիալ շղթաների) համար, որը նշանակում է, որ դիֆերենցիալ վերլուծությամբ հարձակման միջոցով համակարգի գաղտնի բանալին գտնելու համար կպահանջվի նույնքան հաշվարկ, ինչքան հաշվարկ անհրաժեշտ է համապարփակ որոնմամբ (բոլոր հնարավոր բանալիների փորձարկմամբ) բանալին գտնելու համար: Այժմ նկարագրենք r-ռաունդ դիֆերենցիալ շղթայի գաղափարը հետևյալ օրինակի միջոցով:

$$V_0[4,10](128,128) \rightarrow V_1[5,7,8](85a, 85a, a)$$
$$\rightarrow V_0[1,2,5,8,9,11,15,16](170,1,171,171,1,170,172,171)$$

իրնեից ներկայացնում է 2-ռաունդ դիֆերենցիալ շղթա, որը կազմված է 2-ռաունդ քվազի-դիֆերենցիալներից: Առաջին ռաունդի մուտքային դիֆերենցիալը 2 կշռով և 0 պարամետրով $V_0[4,10](128,128)$ վեկտորն է, որտեղ 4-ը և 10-ը ոչ զրոյական բայթերն են, իսկ 128-ը այդ բայթերի արժեքները: $V_1[5,7,8](85a, 85a, a)$ վեկտորը 3 կշռով և 1 պարամետրով (3 կշռով 2^8-1 վեկտորներից մեկը) առաջին ռաունդի ելքային դիֆերենցիալն է, որը նաև հանդիսանում է երկրորդ ռաունդի մուտքային դիֆերենցիալը: 8 կշռով և 0 պարամետրով $V_0[1,2,5,8,9,11,15,16](170,1,171,171,1,170,172,171)$ վեկտորը

երկրորդ ռաունդի ելքային քվազի-դիֆերենցիալն է: Դիֆերենցիալ շղթաների առաջին և վերջին վեկտորները (մուտքային և ելքային) պետք է լինեն միարժեքորեն որոշված վեկտորներ:

r-ռաունդ դիֆերենցիալ շղթաների կարևոր հատկությունները պարունակվում են r-ռաունդ *կշիռ-պարամետրական* շղթաների մեջ: Վերևի օրինակում բերված 2-ռաունդ դիֆերենցիալ շղթայի համապատասխան կշիռ-պարամետրական շղթան ունի հետևյալ տեսքը.

$$2(0) \to 3(1) \to 8(0),$$

որտեղ $2(0)$-ն 2 կշռով մեկ բայթային վեկտորից բաղկացած 0-պարամետրով բազմություն է, $3(1)$-ը 1-պարամետրով բազմություն է, որը բաղկացած է 3 կշռով բայթային վեկտորներից, իսկ $8(0)$-ն 8 կշռով մեկ բայթային վեկտորից բաղկացած 0-պարամետրով բազմություն է:

r-ռաունդ դիֆերենցիալ շղթայի հավանականությունը դա շղթան բնութագրող բոլոր r-ռաունդ քվազի-դիֆերենցիալների հավանականությունների գումարն է: Հիմնվելով նախորդ պարագրաֆում նկարագրված մեկնաբանությունների վրա կարելի է եզրակացնել, որ մեծ հավանականությամբ մի քանի ռաունդներից կազմված դիֆերենցիալ շղթայի որոնումը դա նույն է գտնել հնարավորինս փոքր կշռով դիֆերենցիալ շղթաներ: Որպես դիֆերենցիալ շղթային *մինիմալ կշիռ* դիտարկվում է շղթայի բոլոր վեկտորների ամենափոքր կշռով վեկտորի կշիռը, իսկ *էֆեկտիվ կշիռը*՝ շղթայի բոլոր վեկտորների կշիռների գումարի և շղթայի պարամետրերի գումարի տարբերությունը: Այս պարագրաֆի օրինակում բերված շղթայի մինիմալ կշիռը կլինի հավասար 2-ի, իսկ էֆեկտիվ կշիռը՝ $(2 + 3 + 8) - (0 + 1 + 0) = 12$:

Դիֆերենցիալ շղթայի էֆեկտիվ կշռի գաղափարը ներմուծված է հետևյալ նախադրյալով. յուրաքանչյուր պարամետրի համար գոյություն ունեն ոչ ավել քան 2^8 հնարավոր արժեք (սակայն, ինչպես արդեն նշել ենք վերևում, իրականում գոյություն ունի ընտրության $2^8 - 1 = 255$ հնարավոր արժեք), հետևաբար t պարամետրից բաղկացած հնարավոր դիֆերենցիալ շղթաների առավելագույն քանակը կլինի 2^{8t}: Ավելին, տրված ոչ զրոյական բայթային դիֆերենցիալից մեկ այլ ոչ զրոյական բայթային

դիֆերենցիալի անցումային միջին հավանականությունը 2^{-8} է, այնպես որ, եթե դիֆերենցիալ շղթայի գումարային կշիռը հավասար է w, ապա շղթայի անցումային հավանականությունը կլինի հավասար 2^{-8w}: Հետևաբար դիֆերենցիալ շղթայի հավանականությունը կարելի է համարել մոտավորապես $2^{-8w}2^{8t} = 2^{-8(w-t)}$, որտեղ $(w-t)$-ն շղթայի էֆեկտիվ կշիռն է: Փաստորեն, բացի առաջին ռաունդից, որտեղ $V_0[i](18) \rightarrow V_0[i](128)$ բայթային անցումային հավանականությունը 2^{-5} է, բոլոր բայթային անցումային հավանականությունները կլինեն 2^{-7}-ից փոքր:

Այսպիսով 5 կամ ավել ռաունդից բաղկացած ցանկացած C դիֆերենցիալ շղթայի անցումային $P(C)$ հավանականությունը կբավարարի հետևյալ անհավասարությանը.

$$P(C) < 2^{-7W_{eff}(C)}$$

որտեղ $W_{eff}(C)$-ն C դիֆերենցիալ շղթային էֆեկտիվ կշիռն է:

Քանի, որ 256 բիթային SAFER-256 բլոկային ծածկագրական համակարգը ևս մարկովյան համակարգ է, ապա 1-ռաունդ դիֆերենցիալ շղթաների անցումային հավանականությունների հայտնի լինելը թույլ կտա որոշել մի քանի հաջորդական ռաունդների անցումային հավանականությունները:

2.1.2. SAFER-256 համակարգի դիֆերենցիալ վերլուծության ծրագրային փաթեթի համառոտ նկարագիրը և նվազագույն կշռով շղթաները

Դիֆերենցիալ վերլուծության նկատմամբ SAFER-256 բլոկային ծածկագրական համակարգի կայունությունը ստուգելու համար ստեղծվել է ծրագրային փաթեթ՝ գրված C++ լեզվով Visual Studio 2012-ի միջոցով Windows 7 օպերացիոն համակարգի վրա (հավելված 2-ում բերված է այդ կոդի մի մասը, ամբողջական կոդը կոշտ սկավառակի վրա է): Ծրագրի աշխատանքը սկսվում է "Root" կոճակից աջ սեղմում կատարելով: "New Task" կոճակի միջոցով ազդարարվում է r-ռաունդ դիֆերենցիալ շղթայի սկիզբը և այն

սեղմելուց հետո բացվում է “Task Properties” պատուհանը: Մուտքային դիֆերենցիալի (32 բայթ երկարության վեկտորի) բնութագիրը (օր. տրված կշիռով) հարկավոր է ընտրել “Input Type” հատվածից (Given Weights), իսկ հետո էկրանից ներմուծել քանակական տվյալները (Օր. Input Weights: 2): Եթե մուտքային դիֆերենցիալի պարամետրերի քանակը 0-ից մեծ է, “Task Type” հատվածից ընտրում ենք “Equations” տիպը և ակտիվացված “Variable Count” պատուհանում ներմուծում ենք մեզ անրաժեշտ պարամետրերի քանակը: “Output Type” հատվածում ընտրում ենք պայմանները որոնց համապատասխան ելքային տվյալ ուզում ենք ստանալ. կարելի է ստանալ փոքր կշռով (Minimum), ֆիքսված կշռով (Given Weights) կամ բոլոր (All) հնարավոր ելքային դիֆերենցիալները: Շղթա կազմելուց հետ պետք է սեղմել “R” կոճակը արդյունքը տեսնելու համար: 1-ռաունդ դիֆերենցիալ ստեղծելուց հետո, եթե աջ սեղմում կատարենք այդ շղթայի վրա “Chain” հրամանի միջոցով կարող ենք ստեղծել մինչև 5-ռաունդ դիֆերենցիալ շղթաներ (յուրաքանչյուր ռաունդի ելքում նվազագույն կշռով վեկտորնեն են): “Static” հատվածի “Inverse Matrix” հրամանը ընտրվում է վերծանման պրոցեսի դիֆերենցիալ շղթաները ուսումնասիրելու համար: Դիֆերենցիալ շղթաների էֆեկտիվ կշիռը և անցումային հավանականությունները հաշվում ենք այնպես, ինչպես ցույց է տրված նախորդ պարագրաֆի օրինակում:

Ծրագրային վերը նշված փաթեթի միջոցով դիֆերենցիալ վերլուծության արդյունքները ցույց են տվել, որ 256 բիթ բլոկի և բանալու երկարության SAFER-256 համակարգի դիֆերենցիալ շղթաների հավանականությունները էականորեն փոքր են 2^{-256}-ից 5 և ավելի ռաունդի դեպքում, հետևաբար համակարգը կլինի լիովին կայուն դիֆերենցիալ վերլուծության նկատմամբ 6 ռաունդի դեպքում: Ներքևում բերված են նվազագույն էֆեկտիվ կշռով մի քանի դիֆերենցիալ շղթաներ:

40 էֆեկտիվ կշռով կշիռ-պարամետրական դիֆերենցիալ շղթաներ.

$$4(0) \rightarrow 3(1) \rightarrow 20(1) \rightarrow 10(0) \rightarrow 6(1),$$

$$5(0) \rightarrow 2(1) \rightarrow 24(1) \rightarrow 6(0) \rightarrow 6(1):$$

41 էֆեկտիվ կշռով կշիռ-պարամետրական դիֆերենցիալ շղթաներ.

$$2(0) \to 4(1) \to 24(1) \to 7(0) \to 7(1),$$

$$2(0) \to 5(1) \to 24(1) \to 7(0) \to 6(1),$$

$$3(0) \to 3(1) \to 25(1) \to 6(0) \to 7(1).$$

$$3(0) \to 6(1) \to 14(1) \to 9(0) \to 12(1):$$

42 էֆեկտիվ կշռով կշիռ-պարամետրական դիֆերենցիալ շղթաներ.

$$6(0) \to 3(1) \to 22(1) \to 7(0) \to 7(1):$$

43 էֆեկտիվ կշռով կշիռ-պարամետրական դիֆերենցիալ շղթաներ.

$$6(0) \to 2(1) \to 24(1) \to 7(0) \to 7(1):$$

44 էֆեկտիվ կշռով կշիռ-պարամետրական դիֆերենցիալ շղթաներ.

$$4(0) \to 3(1) \to 20(1) \to 10(1) \to 11(1),$$

$$5(0) \to 3(1) \to 25(1) \to 6(0) \to 8(1):$$

45 էֆեկտիվ կշռով կշիռ-պարամետրական դիֆերենցիալ շղթաներ.

$$4(0) \to 3(1) \to 20(1) \to 8(0) \to 11(1),$$

$$4(0) \to 6(1) \to 14(1) \to 14(0) \to 10(1),$$

$$6(0) \to 5(1) \to 13(0) \to 10(1) \to 14(1),$$

$$6(0) \to 6(1) \to 14(1) \to 13(0) \to 9(1)$$

2.2. Ձևափոխած SAFER+ համակարգի բայթային դիֆերենցիալների և քվազի-դիֆերենցիալների վերլուծությունը և դիֆերենցիալ շղթաները

Ձևափոխած SAFER+ համակարգի դիֆերենցիալ վերլուծությունը կատարվել է նույն տրամաբանությամբ և դատողություններով, ինչպես SAFER+ համակարգի դիֆերենցիալ վերլուծությունը:

Ձևափոխած SAFER+ համակարգի մուտքը և ելքը 16 երկարության վեկտորների բազմությունից են (վեկտորների կոմպոնենտները $\mathbb{Z}_{256}$ օղակի էլեմենտներ են), որի վրա սահմանված է հետևյալ գործողությունը

$$\otimes = [\oplus, \oplus, \oplus, \oplus, +, +, +, +, \oplus, \oplus, \oplus, \oplus, +, +, +, +],$$

որտեղ "$\oplus$" բիթային xor գործողությունն է, իսկ "$+$" ըստ մոդուլ 256-ի բայթային գումարում գործողությունն է:

Ենթադրենք $X = [X_1, X_2, \dots, X_{16}]$ վեկտորը ռաունդի 16 բայթ երկարության մուտքային վեկտորն է, $K_1 = [K_{1,1}, K_{1,2}, \dots, K_{1,16}]$ վեկտորը ռաունդի առաջին բանալին է, իսկ $K_2 = [K_{2,1}, K_{2,2}, \dots, K_{2,16}]$ վեկտորը ռաունդի երկրորդ բանալին է: Ռաունդի հակադարձելի գծային ձևափոխության մուտքային $S = [S_1, S_2, \dots, S_{16}]$ վեկտորի կոորդինատները որոշվում են (2.1) և (2.2) բանաձևերով, միայն թե (2.1)-ում $j \in \{1, 2, 3, 4, 9, 10, 11, 12\}$, (2.2)-ում $j \in \{5, 6, 7, 8, 13, 14, 15, 16\}$: Ի տարբերություն SAFER+ համակարգի ձևափոխած SAFER+ համակարգի ցուցչային բայթեր դիտարկվում են $1, 2, 3, 4, 9, 10, 11, 12$ բայթերը ($1, 4, 5, 8, 9, 12, 13, 16$ բայթերի փոխարեն), իսկ *լոգարիթմական բայթեր* $5, 6, 7, 8, 13, 14, 15, 16$ բայթերը ($2, 3, 6, 7, 10, 11, 14, 15$ բայթերի փոխարեն)*:*

16 բայթ երկարության երկու $V = [V_1, V_2, \dots, V_{16}]$ և $V' = [V'_1, V'_2, \dots, V'_{16}]$ վեկտորների ΔV տարբերություն կամ դիֆերենցիալ կանվանենք.

$$\Delta V = V \oslash V' = [V_1 \oplus V'_1, V_2 \oplus V'_2, V_3 \oplus V'_3, V_4 \oplus V'_4, V_5 - V'_5, V_6 - V'_6, V_7 - V'_7, V_8 - V'_8,$$
$$V_9 \oplus V'_9, V_{10} \oplus V'_{10}, V_{11} \oplus V'_{11}, V_{12} \oplus V'_{12}, V_{13} - V'_{13}, V_{14} - V'_{14}, V_{15} - V'_{15}, V_{16} - V'_{16}],$$

իսկ V և V' վեկտորների $\widetilde{\Delta V}$ քվազի-տարբերություն կամ քվազի-դիֆերենցիալ՝

$$\widetilde{\Delta V} = V - V' = [V_1 - V_1', V_2 - V_2', V_3 - V_3', V_4 - V_4', V_5 - V_5', V_6 - V_6', V_7 - V_7', V_8 - V_8', V_9 - V_9', V_{10} - V_{10}', V_{11} - V_{11}',\ V_{12} - V_{12}', V_{13} - V_{13}', V_{14} - V_{14}', V_{15} - V_{15}',\ V_{16} - V_{16}']:$$

$(\Delta X_j, \Delta S_j)$ զույգի (α, τ) արժեքը այս դեպքում կանվանենք ցուցչային բայթային դիֆերենցիալ, եթե $j \in \{1, 2, 3, 4, 9, 10, 11, 12\}$, իսկ եթե $j \in \{5, 6, 7, 8, 13, 14, 15, 16, \}$ ապա կանվանենք լոգարիթմական բայթային դիֆերենցիալ:

Ձևափոխած SAFER+ համակարգի դիֆերենցիալ վերլուծության համար ևս օգտագործվել է բայթային քվազի-դիֆերենցիալը բայթային դիֆերենցիալի փոխարեն, քանի որ այս համակարգը համաձայն հետևանք 2.2-ի ևս մարկովյան համակարգ է:

16 բայթ երկարության բաց տեքստի ծածկագրման ձևափոխած SAFER+ համակարգի ամենամեծ անցումային հավանականությունը 128 բիթ երկարության բանալու դեպքում փոքր է 2^{-128}-ից 5 և ավելի մեծ երկարության դիֆերենցիալ շղթաների դեպքում, ինչը նշանակում է ձևափոխած SAFER+ համակարգը լիովին կլինի կայուն դիֆերենցիալ վերլուծության նկատմամբ 6 ռաունդի դեպքում, իսկ 256 բիթ երկարության բանալու դեպքում ամենամեծ անցումային հավանականությունը փոքր է 2^{-256}-ից 8 և ավելի մեծ երկարության դիֆերենցիալ շղթաների դեպքում, այսինքն համակարգը կլինի լիովին կայուն դիֆերենցիալ վերլուծության նկատմամբ 9 ռաունդի դեպքում: Այսինքն դիֆերենցիալ վերլուծությամբ հարձակման միջոցով r ռաունդից բաղկացած համակարգի գաղտնի բանալին գտնելու համար կպահանջվի նույնքան հաշվարկ, ինչքան որ անհրաժեշտ է համապարփակ որոնմամբ (բոլոր հնարավոր բանալիների փորձարկմամբ) բանալին գտնելու համար:

Ձևափոխած SAFER+ համակարգի դիֆերենցիալ վերլուծության ընթացքում ևս օգտագործվել է r-ռաունդ կշիռ-պարամետրական շղթայի գաղափարը: Ինչպես նաև համակարգի 5 կամ ավելի մեծ թվով ռաունդներից կազմված C դիֆերենցիալ շղթայի անցումային $P(C)$ հավանականությունը բավարարում է

$$P(C) < 2^{-7W_{eff}(C)}$$

անհավասարությանը, որտեղ $W_{eff}(C)$-ն C դիֆերենցիալ շղթայի էֆեկտիվ կշիռն է:

2.2.1. Ձևափոխած SAFER+ համակարգի նվազագույն կշռով շղթաները

C++ ծրագրային լեզվով Visual Studio 2012-ի միջոցով Windows 7 օպերացիոն համակարգի վրա ստեղծվել է ծրագրային փաթեթ դիֆերենցիալ վերլուծության նկատմամբ ձևափոխած SAFER+ համակարգի կայունությունը ստուգելու և ռաունդների քանակը որոշելու համար (տե՛ս Հավելված 3): Ձևափոխած SAFER+ համակարգի դիֆերենցիալ վերլուծության ծրագրային իրականացման նկարագիրը նույնն է, ինչ և SAFER-256 համակարգինը (տե՛ս Պարագրաֆ 2.1.2), քանի որ այս ծրագրային համակարգերի կիրառության նպատակը նույնն է, տարբեր են ծրագրերի կառուցվածքները (տե՛ս Հավելված 2 և 3): Ծրագրային վերոնշյալ փաթեթի միջոցով ուսումնասիրվել է ձևափոխած SAFER+ համակարգի 5-ռաունդ դիֆերենցիալ շղթաների բոլոր հնարավոր անցումային հավանականությունները: Դիֆերենցիալ վերլուծությունը արդյունքում պարզվել է, որ համակարգի դիֆերենցիալ շղթաների անցումային հավանականությունները փոքր են 2^{-128}-ից, եթե ռաունդների թիվը մեծ է կամ հավասար 5-ի, 128 բիթ երկարության բանալու դեպքում, իսկ 256 բիթ երկարության բանալու դեպքում դիֆերենցիալ շղթաների անցումային հավանականությունները փոքր են 2^{-256}-ից 8 և ավելի մեծ թվով ռաունդներից հետո: Հետևաբար այս համակարգը SAFER+ համակարգը ձևափոխություններից հետ և առաջ կայուն է դիֆերենցիալ վերլուծության նկատմամբ 6 և 9 ռաունդի դեպքում, կախված բանալու երկարությունից: Ներքևում բերված են նվազագույն էֆեկտիվ կշռով մի քանի դիֆերենցիալ շղթաներ:

19 էֆեկտիվ կշռով կշիռ-պարամետրական դիֆերենցիալ շղթաներ.

$$2(0) \to 2(1) \to 11(0) \to 3(1) \to 3(0),$$

$$2(0) \to 2(1) \to 11(1) \to 4(0) \to 3(1),$$

$$3(0) \to 2(1) \to 11(1) \to 5(0) \to 1(1):$$

20 էֆեկտիվ կշռով կշիռ-պարամետրական դիֆերենցիալ շղթաներ.

$$3(0) \to 3(1) \to 10(1) \to 6(0) \to 3(1):$$

22 էֆեկտիվ կշռով կշիռ-պարամետրական դիֆերենցիալ շղթաներ.

$$2(0) \to 2(1) \to 9(1) \to 6(1) \to 6(0):$$

23 էֆեկտիվ կշռով կշիռ-պարամետրական դիֆերենցիալ շղթաներ.

$$2(0) \to 2(1) \to 9(1) \to 6(0) \to 6(1),$$

$$3(0) \to 3(1) \to 10(1) \to 6(0) \to 3(1),$$

$$4(0) \to 2(1) \to 10(1) \to 6(0) \to 4(1),$$

$$4(0) \to 3(1) \to 8(1) \to 8(0) \to 3(1),$$

$$4(0) \to 2(1) \to 11(1) \to 5(0) \to 4(1):$$

24 էֆեկտիվ կշռով կշիռ-պարամետրական դիֆերենցիալ շղթաներ.

$$3(0) \to 3(1) \to 11(1) \to 3(0) \to 7(1),$$

$$3(0) \to 4(1) \to 10(1) \to 6(0) \to 4(1),$$

$$3(0) \to 2(1) \to 11(1) \to 5(1) \to 6(0),$$

$$3(1) \to 8(1) \to 6(0) \to 4(1) \to 6(0),$$

$$4(1) \to 4(0) \to 4(1) \to 12(1) \to 3(0),$$

$$4(1) \to 4(0) \to 4(1) \to 12(0) \to 3(1):$$

ԳԼՈՒԽ 3

ՁԵՎԱՓՈԽԱԾ SAFER+ ԵՎ SAFER-256 ԾԱԾԿԱԳՐԱԿԱՆ ՀԱՄԱԿԱՐԳԵՐԻ ԳԾԱՅԻՆ ՎԵՐԼՈՒԾՈՒԹՅՈՒՆԸ

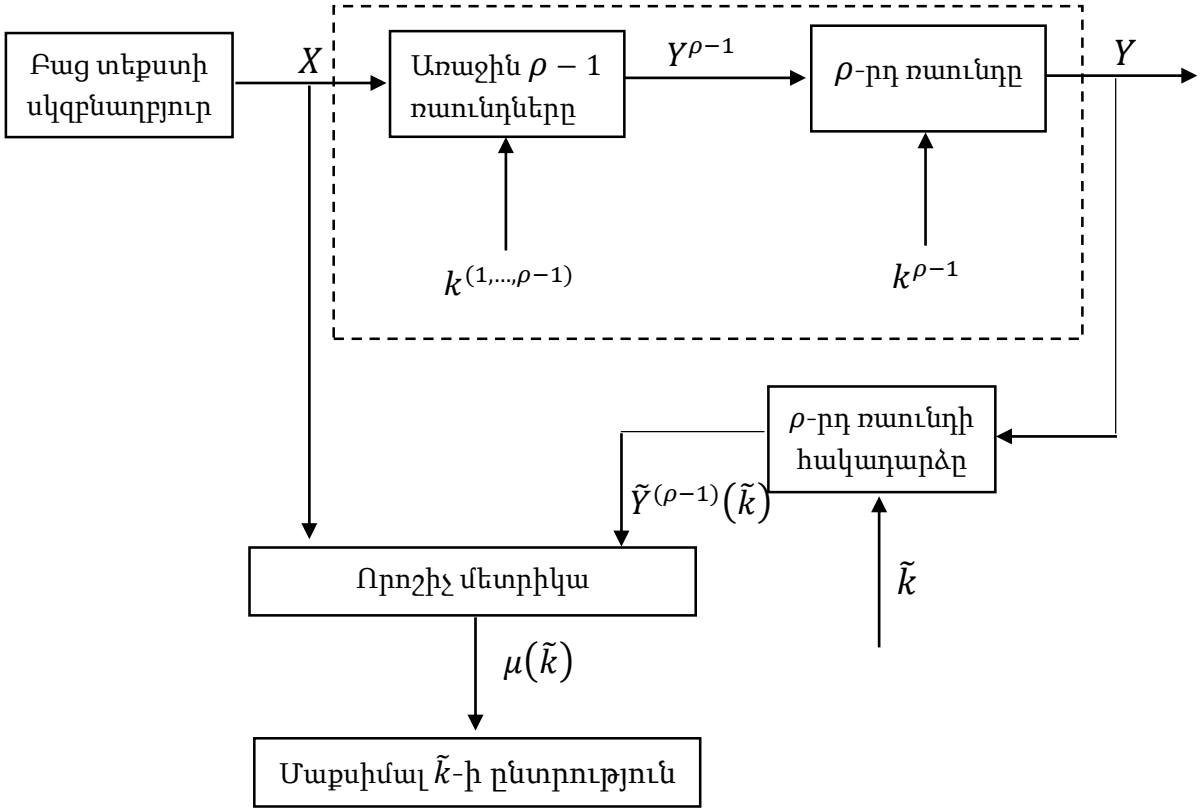
Բլոկային ծածկագրական համակարգերի նկատմամբ մյուս էֆեկտիվ հայտնի հարձակումը գծային վերլուծությունն է, որը առաջարկվել է Մաթսուի կողմից [18]: Գծային վերլուծությունը առավել էֆեկտիվ է հատկապես այնպիսի համակարգերի նկատմամբ, որոնցում գումար գործողության մոդուլը 2 է [5]: Օրինակ SAFER-K և SAFER-SK համակարգերը գծային վերլուծության նկատմամբ կայուն են 3 ռաունդի դեպքում [4],[5],[26],[27]: Գծային վերլուծությունը հայտնի բաց տեքստ հարձակում է (known plaintext attack), այսինքն հարձակվողին նախապես հայտնի է բաց տեքստերի և համապատասխան ծածկագրերի (բաց տեքստ/ծածկագիր) մի որոշ բազմություն:

Ծածկագրական համակարգի գծային արտահայտությունները ըստ մոդուլ 2-ի հավասարումներ են, որոնց մեջ օգտագործվում են բաց տեքստի, ենթաբանալու և ծածկագրի (նախավերջին ռաունդի $Y^{(\rho-1)}$ ելքի (տե՛ս Նկար 16)) բիթերը: Գծային արտահայտությունը կոչվում է չբալանսավորված, եթե հավանականությունը այն բանի, որ տեղի ունի տրված հավասարությունը $1/2$-ից տարբեր է, երբ բաց տեքստը և բանալիները իրարից անկախ կամայական վեկտորներ են:

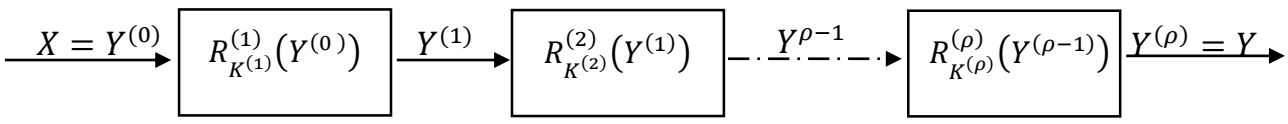
Հարձակվողը իրական բանալին գտնելու համար վերջին ռաունդի համար առանձնացնում է հնարավոր բանալիների բազմություն և կիրառում է վերջին ռաունդ հարձակումը [4],[5], որի նպատակն է, օգտվելով նախորդ ռաունդներում նկատվող թերություններից, գտնել վերջին ռաունդի բանալին: Յուրաքանչյուր ենթադրելի $\tilde{k}$ բանալու միջոցով կատարում է բոլոր (X,Y) բաց տեքստ/ծածկագիր զույգերի վերջին ռաունդի վերծանում, այսինքն հաշվում է հետևյալ արտահայտությունը $\tilde{Y}^{(\rho-1)}(\tilde{k}) := R_{\tilde{k}}^{-1}(Y)$: $\mu(\tilde{k})$ *որոշիչ մետրիկան* (decision metric) $\tilde{Y}^{(\rho-1)}(\tilde{k})$-ից և X-ից ստացված վիճակագրություն է, օրինակի համար այս փոփոխականներից կախված ֆունկցիայի սպասումը: Որոշիչ մետրիկան բանալու մոտավոր գնահատականի ֆունկցիան է, որը

ընտրվում է այնպես, որ այն իր մեծագույն արժեքը ընդունում է այն դեպքում, երբ ենթադրելի բանալին համընկնում է վերջին ռաունդի իրական բանալու հետ: Նկար 15-ում պատկերված է վերջին ռաունդ հարձակումը (X, Y) բաց տեքստ/ծածկագիր զույգերից մեկի համար:

Գծային վերլուծության նպատակն է. համակարգի գաղտնի բանալիներով գծային արտահայտությունների չբալանսավորվածությունը լինի ավելի մեծ սխալ բանալիներով գծային արտահայտությունների չբալանսավորվածությունից:



Նկար 15: Վերջին ռաունդ հարձակումը սխեմատիկորեն:



Նկար 16: Իտերատիվ բլոկային ծածկագրական համակարգի ընդհանուր կառուցվածքը (X-ը բաց տեքստն է, $K^{(1)}, K^{(2)}, \dots, K^{(\rho)}$ համապատասխան ռաունդների բանալիներն են, R-ը բանալուց կախված ռաունդ ֆունկցիան է, Y-ը ծածկագիրն է):

3.1. Հիմնական հասկացություններ և պնդումներ

Այս պարագրաֆում բերված է SAFER ընտանիքի նոր մշակված համակարգերի գծային վերլուծության համար անհրաժեշտ և նախորդ համակարգերի գծային վերլուծության համար օգտագործված [4] և [5] աշխատանքների սահամանումները և նշանակումները:

Բինար ֆունկցիան կոչվում է *հավասարակշռված կամ բալանսավորված (balanced)*, եթե արգումենտի բոլոր հնարավոր արժեքների կեսի դեպքում ընդունում է 0 արժեք, իսկ մյուս կեսի դեպքում՝ 1 արժեք: Իտերատիվ ծածկագրական համակարգի i-րդ ռաունդի $S^{(i)}$ I/O (Input/Output-մուտք/ելք) գումար անվանում են հետևյալ գումարին

$$S^{(i)} := f_i\left(Y^{(i-1)}\right) \oplus g_i\left(Y^{(i)}\right),$$

որտեղ $X^{(i)} = Y^{(i-1)}$ i-րդ ռաունդի մուտքն է, $Y^{(i)}$-ն i-րդ ռաունդի ելքը, իսկ f_i և g_i ֆունկցիաները բինար բալանսավորված ֆունկցիաներ են:

f_i և g_i ֆունկցիաները կոչվում են $S^{(i)}$ I/O գումարի համապատասխանաբար մուտքային և ելքային ֆունկցիաներ: Ասում են, որ հաջորդական ռաունդների I/O գումարները *կապակցված են*, եթե այդ I/O գումարներից յուրաքանչյուրի ելքային ֆունցիան հանդիսանում է հաջորդ ռաունդի I/O գումարի մուտքային ֆունկցիան (այսինքն $g_i = f_{i+1}$), բացառությամբ վերջի I/O գումարի: Եթե $S^{(1)}, S^{(2)}, \dots, S^{(\rho)}$ I/O գումարները կապակցված են, ապա նրանց ըստ մոդուլ 2-ի գումարը ևս հանդիսանում է I/O գումար, որին անվանում են ρ-ռաունդ I/O գումար և նշանակում են այսպես.

$$S^{(1\dots\rho)} := \bigoplus_{i=1}^{\rho} S^{(i)} = f_1\left(Y^{(1)}\right) \oplus g_\rho\left(Y^{(\rho)}\right),$$

որտեղ $Y^{(1)} = X^{(1)} = X$ առաջին ռաունդի մուտքն է, այսինքն բաց տեքստը, իսկ $Y^{(\rho)}$-ն վերջին ռաունդի ելքն է, այսինքն ծածկագիրը:

Գծային վերլուծության մեջ օգտագործվում է պատահական բինար մեծության չբալանսավորվածության գաղափարը: $|2P[V=0]-1|$ իրական թիվը անվանում են V պատահական բինար մեծության *չբալանսավորվածություն* և նշանակում են $I(V)$-ով: $P[V=0]$ հավանականություն է այն բանի, որ V-ն կընդունի 0 արժեք: Նկատենք, որ

$$0 \le I(V) \le 1,$$

ընդ որում $I(V)$ հավասար է 1-ի այն և միայն այն դեպքում, եթե V-ն հաստատուն է, և հավասար է 0-ի այն և միայն այն դեպքում, եթե $P[V = 0] = 1/2$:

ρ ռաունդից կազմված իտերատիվ բլոկային ծածկագրական համակարգը ծածկագրում է n երկարությամբ բաց տեքստը ρ անգամ կիրառելով ռաունդ ֆունկցիան և յուրաքանչյուր ռաունդում օգտագործելով իրարց տարբեր բանալիներ: $K^{(1\dots\rho)} = \left(K^{(1)}, K^{(2)}, \dots, K^{(\rho)}\right)$ կնշանակենք համակարգում օգտագործվող բանալիների բազմությունը, որտեղ $K^{(i)}$-ն i-րդ ռաունդի բանալին է, $i = 1,2, \dots, \rho$: SAFER ընտանիքի բլոկային ծածկագրական համակարգերում $K^{(i)} = (K_{2i-1}, K_{2i})$:

Գծային վերլուծության ժամանակ ենթադրվում է, որ բաց տեքստը և համակարգում օգտագործվող բոլոր բանալիները իրարից անկախ են և համապատասխան բազմությունների վրա հավասարաչափ բաշխված: Այնումենայնիվ բանալիները գեներացվում են օգտատիրոջ կողմից ընտրված գաղտնի բանալուց բանալիների գեներացման ալգորիթմի միջոցով:

$S^{(1\dots\rho)}$ I/O գումարի բանալուց կախված չբալանսավորվածությունը, երբ $K^{(1\dots\rho)}$=$k^{(1\dots\rho)}$՝ կնշանակենք այսպես $I\left(S^{(1\dots\rho)}|k^{(1\dots\rho)}\right)$ և կանվանենք $S^{(1\dots\rho)}$ I/O գումարի *բանալուց կախված չբալանսավորվածություն:* Իսկ բանալուց կախված չբալանսավորվածության մաթեմատիկական սպասումը, երբ ռաունդի բանալիները ընտրված են կամայականության սկզբունքով և իրարից անկախ, կանվանենք $S^{(1\dots\rho)}$ I/O գումարի *բանալուց կախված միջին չբալանսավորվածություն.*

$$\bar{I}\left(S^{(1\dots\rho)}\right) := E\left[I\left(S^{(1\dots\rho)}|K^{(1\dots\rho)}\right)\right] = \frac{1}{|\mathcal{K}^{\rho}|}\sum_{k^{(1\dots\rho)}\in\mathcal{K}^{\rho}} I\left(S^{(1\dots\rho)}|k^{(1\dots\rho)}\right)$$

որտեղ $|\mathcal{K}^{\rho}|$-ն հնարավոր բոլոր բանալիների բազմության չափողականությունն է:

I/O գումարը կոչվում է *էֆեկտիվ (effective),* եթե այն ունի մեծ բանալուց կախված միջին չբալանսավորվածություն և *երաշխավորված (guaranteed)*, եթե բանալուց կախված միջին չբալանսավորվածությունը 1 է, այսինքն ընդունում է հնարավոր ամենամեծ արժեքը:

Վերջին ռաունդ հարձակման ժամանակ վերջին ռաունդի բանալին գտնելու համար օգտագործվում է էֆեկտիվ $(\rho-1)$-ռաունդ $S^{(1\dots\rho-1)}=g_0(X)\oplus g_{\rho-1}\big(Y^{(\rho-1)}\big)$ I/O գումարը, իսկ որպես որոշիչ մետրիկա դիտարկվում է.

$$\mu\big(\tilde{k}\big)=I\big(\tilde{S}^{(1\dots\rho-1)}\big(\tilde{k}\big)|k^{(1\dots\rho)}\big)$$

ֆունկցիան, որտեղ

$$\tilde{S}^{(1\dots\rho-1)}\big(\tilde{k}\big):=g_0(X)\oplus g_{\rho-1}\Big(\tilde{Y}^{(\rho-1)}\big(\tilde{k}\big)\Big),$$

որտեղ $\tilde{Y}^{(\rho-1)}\big(\tilde{k}\big):=R_{\tilde{k}}^{-1}(Y)$, իսկ $\tilde{k}$-ը հարձակվողի կողմից գուշակած ենթադրելի բանալին է:

i-րդ ռաունդի *եռանդամ կամ եռակի գումար* անվանում են ըստ մոդուլ 2-ի հետևալ

$$T^{(i)}:=f_i\big(Y^{(i-1)}\big)\oplus g_i\big(Y^{(i)}\big)\oplus h_i\big(K^{(i)}\big)$$

գումարը, որտեղ $X^{(i)}=Y^{(i-1)}$-ը i-րդ ռաունդի մուտքն է, $Y^{(i)}$-ն i-րդ ռաունդի ելքն է, $K^{(i)}$-ն i-րդ ռաունդի բանալին է, f_i, g_i ֆունկցիաները բալանսավորված բինար ֆունկցիաներ են, իսկ h_i-ն՝ կամայական բինար ֆունկցիա:

h_i-ն անվանում են է $T^{(i)}$ գումարի *առանցքային ֆունկցիա (key function)*, իսկ $S^{(i)}=f_i\big(Y^{(i-1)}\big)\oplus g_i\big(Y^{(i)}\big)$ գումարը՝ $T^{(i)}$-ի *ծնիչ I/O գումար (parent sum)*:

Թեորեմ 3.1 (Եռակի գումարի չբալանսավորվածության և I/O գումարի բանալուց կախված միջին չբալանսավորվածության մասին) [4]: Թող $T^{(1\dots\rho)}=S^{(1\dots\rho)}\oplus h_i\big(K^{(1\dots\rho)}\big)$ եռակի գումար է: Այդ դեպքում այդ գումարը ծնող $S^{(1\dots\rho)}$ I/O գումարի բանալուց կախված միջին չբալանսավորվածությունը սահմանափակ է ներքևից եռակի գումարի չբալանսավորվածությամբ.

$$\bar{I}\big(S^{(1\dots\rho)}\big)\geq I\big(T^{(1\dots\rho)}\big):$$

Ավելին, հավասարությունը տեղի ունի այն և միայն այն դեպքում, եթե h-ը հավասար է h_{max}, որտեղ

$$h_{max}=\begin{cases}0 & if \quad P[S=0/K=0]>1/2\\ 1 & if \quad P[S=0/K=0]<1/2\\ arbitary & if \quad P[S=0/K=0]=1/2\end{cases}$$

կամ հանդիսանում է այդպիսի ֆունկցիայի լրացումը:

Եթե $h = h_{max}$, ապա h առանցքային ֆունկցիային անվանում են $T^{(1\dots\rho)}$ ֆունկցիայի մաքսիմալացնող առանցքային ֆունկցիա (maximizing key function):

Եռակի գումարը կոչվում *երաշխավորված*, եթե նրա չբալանսավորվածությունը հավասար է 1-ի: Թեորեմ 3.1-ից հետևում է, որ $T^{(1\dots\rho)} = S^{(1\dots\rho)} \oplus h\left(K^{(1\dots\rho)}\right)$ եռակի գումարը երաշխավորված է այն և միայն այն դեպքում, եթե նրա ծնող $S^{(1\dots\rho)}$ I/O գումարը երաշխավորված է, իսկ h-ը մաքսիմալացնող առանցքային ֆունկցիա է:

Լեմմա 3.1 (Matsui's Piling-up): Իրարից անկախ $V^{(1)}, V^{(2)}, \dots, V^{(\rho)}$ բինար պատահական մեծությունների ըստ մոդուլ 2-ի գումարի չբալանսավորվածությունը հավասար է գումարելիների չբալանսավորվածությունների արտադրյալին, այսինքն

$$I\left(\bigoplus_{i=1}^{\rho} V^{(i)}\right) = \prod_{i=1}^{\rho}\left(I(V)^{(i)}\right):$$

Դիտողություն 3.1: Եթե $T^{(1)}, T^{(2)}, \dots, T^{(\rho)}$ եռակի գումարներ են, ապա Լեմմա 3.1-ից հետևում է, որ

$$I\left(\bigoplus_{i=1}^{\rho} T^{(i)}\right) = \prod_{i=1}^{\rho}\left(I(T)^{(i)}\right),$$

որտեղ $\oplus_{i=1}^{\rho} T^{(i)}$ ρ-ռաունդ եռակի գումարն է, պայմանով որ $T^{(1)}, T^{(2)}, \dots, T^{(\rho)}$ եռակի գումարները կապակցված են:

Համակարգի ռաունդ ֆունկցիան կարող ենք ներկայացնել հետևյալ տեսքով.

$$Y^{(i)} = R^{(i)}_{K^{(i)}}\left(Y^{(i-1)}\right) = \phi\left(Y^{(i-1)} \otimes_i K_{2i-1}, K_{2i}\right),$$

որտեղ $\otimes_i$ խմբային գործողություն է (տե՛ս Նկար 8, 11):

Հետագայում ցույց կտրվի, որ ռաունդի այսպիսի կառուցվածքը ապահովում է ցանկացած ռաունդի հոմոմորֆ եռակի գումարի և այդ ռաունդի մուտքի անկախությունը:

Սահմանում 3.1: Դիցուք B^n-ը n երկարության բինար վեկտորների բազմություն է. $B^n = \{0, 1, 2, \dots, 2^n - 1\}$: $(B^n, \otimes)$ խմբից $(B, \oplus)$ խմբի մեջ հոմոմորֆ արտապատկերումը կամ

հոմոմորֆիզմը կոչվում է $\otimes$[1] գործողության բինար հոմոմորֆիզմ: i-րդ ռաունդից j-րդ ռաունդ $(i \le j)$ անցման I/O գումարը հոմոմորֆ է, եթե մուտքային ֆունկցիան հանդիսանում է բինար հոմոմրֆիզմ $\otimes_i$ գործողության համար, իսկ ելքային ֆունկցիան բինար հոմոմրֆիզմ $\otimes_{j+1}$ գործողության համար: Եռակի գումարը հոմոմրֆ է, եթե հոմոմրֆ է նրա ծնող I/O գումարը:

Եթե i-րդ ռաունդի $\otimes_i$ գործողությունը միայն բիթային XOR գործողությունն է B^n-ում, ապա միակ բինար հոմոմորֆիզմը դա $l_a(x) = a \circ x$ գծային ֆունկցիան է, որտեղ $x \in B^n$, a-ն զրոյից տարբեր n երկարության վեկտոր է, իսկ $a \circ x$ սկալյար արտադրյալ $GF(2)$ դաշտի վրա սահմանված գործողություն է: I/O (կամ եռակի) գումարը, որի մուտքային և ելքային ֆունկցիաները համապատասխանաբար l_a և l_b ֆունկցիաներն են կանվանենք (a, b) գծային դիմակով (linear mask) գծային գումար:

Թեորեմ 3.2: Դիտարկենք ρ ռաունդների հաջորդականությունը, որոնց ռաունդ ֆունկցիաներն են $R^{(1)}, R^{(2)}, \dots, R^{(\rho)}$.

$$Y^{(i)} = R^{(i)}_{K^{(i)}}\left(Y^{(i-1)}\right) = \phi_i(Y^{(i-1)} \otimes_i K_{2i-1}, K_{2i})$$

որտեղ $\otimes_i$-ով նշանակված է խմբային գործողությունը B^n-ում, $\phi_i(., k_{2i})$ ֆունկցիան B^n-ի բիեկտիվ արտապատկերումն է բոլոր k_{2i}-երի համար (տե՛ս Նկար 8, 11): Դիցուք

$$T^{(i)} = f_i\left(Y^{(i-1)}\right) \oplus f_{i+1}\left(Y^{(i)}\right) \oplus \left(f_i(K_{2i-1}) \oplus h_i(K_{2i})\right) \tag{3.1}$$

i-րդ ռաունդի հոմոմորֆ եռակի գումարն է, այնպիսին, որ $T^{(1)}, T^{(2)}, \dots, T^{(\rho)}$ կապակցված են: Այդ դեպքում, $T^{(1\dots\rho)} := \oplus_{i=1}^{\rho} T^{(i)}$ եռակի գումարը ծնող $S^{(1\dots\rho)}$ I/O գումարի բանալուց կախված միջին չբալանսավորվածությունը ներքևից սահմանափակ է առանձին ռաունդների եռակի գումարների չբալանսավորվածությունների արտադրյալով.

$$\bar{I}\left(S^{(1\dots\rho)}\right) \ge \prod_{i=1}^{\rho} I(T^{(i)}):\tag{3.2}$$

[1] f արտապատկերումը հոմոմորֆիզմ է, եթե $f(U \otimes V) = f(U) \oplus f(V)$ ցանկացած $U, V \in B^n$ տարրերի համար և f-ը միարժեքորեն 0 չէ:

Նկատենք, որ քանի որ $\phi_i(.,k_{2i})$ ֆունկցիան բիեկտիվ է, իսկ X-ը հավասարաչափ բաշխված, ապա $Y^{(0)}, Y^{(1)}, \dots, Y^{(\rho-1)}$ մոտքային բլոկները ևս հավասարաչափ բաշխված են: Այսպիսով հեշտությամբ կարելի է գտնել $I(T^{(i)})$-ն ցանկացած $i = 2, 3, \dots, \rho$ համար:

Տրված $S^{(1\dots\rho)}$ գումարի համար կարող ենք գնահատել (2.5) անհավասարության աջ մասը առանձին ռաունդների կապակցված հոմոմորֆ եռակի գումարների տարբեր բազմությունների համար, որոնց գումարի ծնող I/O գումարը հենց $S^{(1\dots\rho)}$ է: Այնուհետև կարող ենք մոտարկել $\bar{I}\left(S^{(1\dots\rho)}\right)$-ն.

$$\bar{I}\left(S^{(1\dots\rho)}\right) \approx \max_{\substack{f_2,\dots,f_\rho \\ h_1,\dots,h_\rho}} \prod_{i=1}^{\rho} I(T^{(i)}), \tag{3.3}$$

որտեղ $T^{(i)}$ որոշվում է (2.4) բանաձևի միջոցով:

Էֆեկտիվ ρ-ռաունդ հոմոմորֆ I/O գումարներ գտնելու էֆեկտիվ մեթոդը.

1. $i = 1,2, \dots, \rho + 1$ արժեքների համար գտնել B^n-ի բոլոր բինար ֆունկցիաների $\mathcal{H}_i$ բազմությունը, որոնք հանդիսանում է բինար հոմոմրֆիզմ $\otimes_i$ գործողության համար:
2. $i = 1,2, \dots, \rho$ արժեքների համար գտնել i-ռաունդ բոլոր հոմոմորֆ եռակի գումարների չբալանսավորվածությունը, $f_i \in \mathcal{H}_i$ մուտքային ֆունկցիայի, $f_{i+1} \in \mathcal{H}_{i+1}$ ելքային ֆունկցիայի և մաքսիմալացնող առանցքային ֆունկցիայի դեպքում: Բացառել փոքր չբալանսավորվածությամբ եռակի գումարները:
3. Դիտարկել ρ կապակցված եռակի գումարների յուրաքանչյուր հնարավոր ընտրություն, որը պարունակում 2-րդ քայլում գտած ռաունդների եռակի գումարներից մեկը:

(2.6)-ի աջ մասը օգտագործել ρ-ռաունդ I/O գումարների չբալանսավորվածությունը գնահատելու համար: Որպես արդյունք վերցնել չբալանսավորվածության ամենամեծ գնահատականով ρ-ռաունդ I/O գումարները:

3.2. Ձևափոխած SAFER+ համակարգի գծային վերլուծությունը

Էֆեկտիվ ρ-ռաունդ հոմոմորֆ I/O գումարը գտնելու մեթոդի միջոցով գտնենք ձևափոխած SAFER+ ալգորիթմի կիսառաունդների էֆեկտիվ հոմոմորֆ I/O գումարները: Ձևափոխած SAFER+ համակարգի նկար 8–ում պատկերված ռաունդ ֆունկցիան բաժանանենք երկու կիսառաունդների:

1. Ոչ գծային կիսառաունդ $(\mathrm{NL\ half-round})$

   1.1. $\mathrm{XOR/ADD}$ – ռաունդի մուտքին «գումարվում» է ռաունդի առաջին ենթաբանալին ըստ մոդուլ 2-ի (XOR) կամ ըստ մոդուլ 256-ի (ADD). $\mathrm{U = XOR/ADD(X, K_{2i-1}})$:

   1.2. $\mathrm{NL\ (Non\ Linear)}$ –ոչ գծային այս շերտում մոտքային բայթերի վրա կիրառվում են երկու ոչ գծային $\mathrm{EXP: x \to 45^{x} modulo 257}$ $(45^{128} = 0)$ կամ նրա հակադարձ LOG ֆունցիաները. $\mathrm{V = NL(U)}$:

2. PHT կիսառաունդ $(\mathrm{PHT\ half-round})$

   2.1. $\mathrm{ADD/XOR}$ – ոչ գծային շերտի ելքային բայթերին «գումարվում» է ռաունդի երկրորդ եմթաբանալին հետևյալ սկզբունքով. այն բայթերը որոնց առաջին ենթաբանալու բայթերը գումարվել է ըստ մոդուլ 2-ի (XOR) երկրորդ ենթաբանալլի բայթերը կգումարվեն ըստ մոդուլ 256-ի (ADD), իսկ այն բայթերը որոնց առաջին ենթաբանալու բայթերը գումարվել է ըստ մոդուլ 256-ի (ADD) երկրորդ ենթաբանալլի բայթերը կգումարվեն ըստ մոդուլ 2-ի (XOR) (տե՛ս Նկար 8). $\mathrm{W = ADD/XOR(V, K_{2i}})$:

   2.2. $\mathrm{PHT(Pseudo\ Hadamard\ Transform)}$ –հակադարձելի գծային ձևափոխությունը կազմված է 4 անգամ կիրառվող 2-PHT +"Armenian Shuffle" շերտից և այս ձևափոխությունը համարժեք է հակադարձելի գծային ձևափոխության մատրիցով բազմապատկմանը (տե՛ս Պարագրաֆ 1.5.6). $\mathrm{Y = PHT(W)}$.

$$\begin{pmatrix} Y1 \\ Y2 \\ Y3 \\ Y4 \\ Y5 \\ Y6 \\ Y7 \\ Y8 \\ Y9 \\ Y10 \\ Y11 \\ Y12 \\ Y13 \\ Y14 \\ Y15 \\ Y16 \end{pmatrix} = \begin{pmatrix} 16 & 1 & 4 & 2 & 2 & 2 & 4 & 1 & 2 & 1 & 4 & 4 & 1 & 2 & 1 & 8 \\ 8 & 1 & 2 & 2 & 2 & 1 & 2 & 1 & 1 & 1 & 4 & 2 & 1 & 1 & 1 & 4 \\ 1 & 2 & 1 & 2 & 4 & 1 & 4 & 4 & 16 & 1 & 4 & 2 & 2 & 8 & 2 & 1 \\ 1 & 1 & 1 & 1 & 2 & 1 & 4 & 2 & 8 & 1 & 2 & 2 & 2 & 4 & 1 & 1 \\ 4 & 2 & 2 & 2 & 16 & 1 & 1 & 8 & 1 & 4 & 2 & 1 & 4 & 1 & 4 & 2 \\ 4 & 1 & 2 & 1 & 8 & 1 & 1 & 4 & 1 & 2 & 1 & 1 & 2 & 1 & 2 & 2 \\ 4 & 1 & 2 & 1 & 1 & 2 & 16 & 1 & 4 & 2 & 2 & 8 & 4 & 2 & 1 & 4 \\ 2 & 1 & 1 & 1 & 1 & 1 & 8 & 1 & 4 & 1 & 2 & 4 & 2 & 2 & 1 & 2 \\ 2 & 4 & 4 & 8 & 4 & 2 & 1 & 2 & 1 & 2 & 16 & 1 & 2 & 1 & 4 & 1 \\ 2 & 2 & 4 & 4 & 2 & 2 & 1 & 1 & 1 & 1 & 8 & 1 & 1 & 1 & 2 & 1 \\ 4 & 1 & 16 & 1 & 4 & 8 & 2 & 2 & 4 & 1 & 1 & 2 & 1 & 4 & 2 & 2 \\ 2 & 1 & 8 & 1 & 4 & 4 & 1 & 2 & 2 & 1 & 1 & 1 & 1 & 2 & 2 & 1 \\ 2 & 2 & 4 & 1 & 1 & 4 & 4 & 1 & 2 & 8 & 1 & 2 & 16 & 1 & 4 & 2 \\ 1 & 2 & 2 & 1 & 1 & 2 & 2 & 1 & 2 & 4 & 1 & 1 & 8 & 1 & 4 & 1 \\ 1 & 8 & 1 & 4 & 2 & 1 & 2 & 2 & 4 & 2 & 4 & 1 & 4 & 2 & 16 & 1 \\ 1 & 4 & 1 & 2 & 1 & 1 & 2 & 1 & 2 & 2 & 2 & 1 & 4 & 1 & 8 & 1 \end{pmatrix} \cdot \begin{pmatrix} W1 \\ W2 \\ W3 \\ W4 \\ W5 \\ W6 \\ W7 \\ W8 \\ W9 \\ W10 \\ W11 \\ W12 \\ W13 \\ W14 \\ W15 \\ W16 \end{pmatrix} (\mathrm{mod}\ 256):$$

Նախ և առաջ $\mathrm{ADD/XOR}$ ու $\mathrm{XOR/ADD}$ խմբային գործողությունների համար գտնենք բոլոր բինար հոմոմորֆիզմները: 8 բիթերի XOR գործողության համար գոյություն ունի $2^8 - 1$ բինար հոմոմորֆիզմ, որը որոշվում է այսպես. $l_{a2}(V2) := a2 \circ V2$, որտեղ $a2$ վեկտորը 8 երկարությամբ բինար վեկտոր է, " $\circ$ " ըստ մոդուլ 2-ի սկալյար արտադրյալ է: Իսկ ըստ մոդուլ 256-ի գումար գործողության համար գոյություն ունի միայն մեկ բինար հոմոմորֆիզմ՝ l_{a1}, $a1 = 0000\ 0001 = 01_{\mathrm{hex}} = 01$ (hex notation of byte):

Այսպիսով, $\mathrm{ADD/XOR}$ խմբային գործողության համար գոյություն կունենա $2^{72} - 1$ բալանսավորված հոմոմորֆիզմ, որը որոշվում է այսպես. $l_a(V) := a \circ V$, որտեղ a-ն 128 երկարության կամայական վեկտոր է

$$\mathcal{A} = \{a : a \in \{0,1\}^{128} \backslash \{00\}; a1, a2, a3, a4, a9, a10, a11, a12\} \in \{00, 01\}\}$$

բազմությունից: Իսկ $\mathrm{XOR/ADD}$ խմբային գործողության համար գոյություն կունենա $2^{72} - 1$ բալանսավորված հոմոմորֆիզմ. $l_b(V) := b * V$, որտեղ b-ն

$$\mathcal{B} = \{b : b \in \{0,1\}^{128} \backslash \{00\}; b5, b6, b7, b8, b13, b14, b15, b16\} \in \{00, 01\}\}$$

բազմության կամայական վեկտոր է:

Հարկ է նշել, որ $\mathrm{XOR/ADD}$ և $\mathrm{ADD/XOR}$ գործողությունների բոլոր հոմոմորֆ ֆունկցիաները գծային բինար ֆունկցիաների բազմության ենթաբազմություններ են:

Հետևաբար մեր կողմից իրականացվող մեթոդով հարձակումը ավելի սահմանափակ է, քան բոլոր գծային I/O գումարների վերլուծությամբ հարձակումը, որը գործնականորեն համարյա անիրագործելի խնդիր է: Սակայն փորձը ցույց է տալիս, որ եթե այն լիներ իրագործելի դժվար թե նկատելիորեն ավելի էֆեկտիվ լիներ, քան մեր կողմից իրականացվող մեթոդը:

Այժմ դիտարկենք ռաունդի այն կեսը, որը պարունակում է PHT ֆունկցան (PHT-half round): Ներքևում բերված լեմմայի միջոցով կարելի որոշել ոչ զրոյական չբալանսավորվածությամբ բոլոր հոմոմորֆ I/O գումարները: Մուտքային ֆունկցիան պետք է լինի բալանսավորված և հոմոմորֆ ADD/XOR գործողության համար, իսկ ելքային ֆունկցիան պետք է լինի բալանսավորված և հոմոմորֆ XOR/ADD գործողության համար: Այսպիսով գոյություն ունեն այդպիսի $(2^{72}-1)^2$ I/O գումար.

$$\mathrm{S}_{a,b}^{\mathrm{PHT-hr}} := l_a(\mathrm{V}) \oplus l_b(\mathrm{Y}), \qquad a \in \mathcal{A}, b \in \mathcal{B}.$$

Լեմմա 3.2: PHT-կիսառաունդի (PHT-half-round(PHT-hr)) հոմոմորֆ I/O գումարները, որոնք ունեն ոչ զրոյական չբալանսավորվածություն, $2^{16}-1$ երաշխավորված I/O գումարներն են, որոնք ստացվում են Աղյուսակ 4-ում բերված 16 երաշխավորված I/O գումարներից՝ վերջավոր թվով գումարներ XOR-անելով:

Ապացույց: Քանի որ

$$I\left(S_{a,b}^{\mathrm{PHT-hr}} | k_{2i}\right) = I\left(S_{a,b}^{\mathrm{PHT}}\right)$$

ցանկացած $k_{2i} \in B^{128}$, որտեղ

$$S_{a,b}^{\mathrm{PHT}} := l_a(W) \oplus l_b(Y), \qquad a \in \mathcal{A}, b \in \mathcal{B}$$

PHT ֆունկցիայի I/O գումարն է, հետևաբար բանալուց կախված ոչ զրոյական չբալանսավորվածության $S_{a,b}^{\mathrm{PHT-hr}}$ I/O գումար գտնելը նույնն է գտնել ոչ զրոյական չբալանսավորվածության $S_{a,b}^{\mathrm{PHT}}$ I/O գումար: Այսպիսով բավական է գտնել այնպիսի $a \in \mathcal{A}$ և $b \in \mathcal{B}$, որունց համար $S_{a,b}^{\mathrm{PHT}} := l_a(W) \oplus l_b(\mathrm{PHT}(W))$ գումարը կունենա $W_\alpha \oplus \phi(W_{127}, \dots, W_{\alpha+1}, W_{\alpha-1}, \dots, W_0)$ տեսքը կամայական W_α մուտքային բիթի համար, քանի որ այդ դեպքում I/O գումարի չբալանսավորվածությունը հավասար է 0:

Աղյուսակ 4: Ձևափոխած SAFER+ ալգորիթմի PHT ֆունկցիայի էֆեկտիվ I/O գումարները:

(a,b)	$l_b(Y)$	$l_a(W)$	$I(S_{a,b}^{\mathrm{PHT}})$
$(0100000101001010, 1000000000000000)$	$Y1_0$	$W2_0 \oplus W8_0 \oplus W10_0 \oplus W13_0 \oplus W15_0$	1
$(0100010111001110, 0100000000000000)$	$Y2_0$	$W2_0 \oplus W6_0 \oplus W8_0 \oplus W9_0 \oplus W10_0 \oplus W11_0 \oplus W13_0 \oplus W14_0 \oplus W15_0$	1
$(1010010001000001, 0010000000000000)$	$Y3_0$	$W1_0 \oplus W3_0 \oplus W6_0 \oplus W10_0 \oplus W16_0$	1
$(1111010001000011, 0001000000000000)$	$Y4_0$	$W1_0 \oplus W2_0 \oplus W3_0 \oplus W4_0 \oplus W6_0 \oplus W10_0 \oplus W15_0 \oplus W16_0$	1
$(0000011010010100, 0000100000000000)$	$Y5_0$	$W6_0 \oplus W7_0 \oplus W9_0 \oplus W12_0 \oplus W14_0$	1
$(0101011010110100, 0000010000000000)$	$Y6_0$	$W2_0 \oplus W4_0 \oplus W6_0 \oplus W7_0 \oplus W9_0 \oplus W11_0 \oplus W12_0 \oplus W14_0$	1
$(0101100100000010, 0000001000000000)$	$Y7_0$	$W2_0 \oplus W4_0 \oplus W5_0 \oplus W8_0 \oplus W15_0$	1
$(0111110101000010, 0000000100000000)$	$Y8_0$	$W2_0 \oplus W3_0 \oplus W4_0 \oplus W5_0 \oplus W6_0 \oplus W8_0 \oplus W10_0 \oplus W15_0$	1
$(0000001010010101, 0000000010000000)$	$Y9_0$	$W7_0 \oplus W9_0 \oplus W12_0 \oplus W14_0 \oplus W16_0$	1
$(0000001111011101, 0000000001000000)$	$Y10_0$	$W7_0 \oplus W8_0 \oplus W9_0 \oplus W10_0 \oplus W12_0 \oplus W13_0 \oplus W14_0 \oplus W16_0$	1
$(0101000001101000, 0000000000100000)$	$Y11_0$	$W2_0 \oplus W4_0 \oplus W10_0 \oplus W11_0 \oplus W13_0$	1
$(0101001001111001, 0000000000010000)$	$Y12_0$	$W2_0 \oplus W4_0 \oplus W7_0 \oplus W10_0 \oplus W11_0 \oplus W12_0 \oplus W13_0 \oplus W16_0$	1
$(0001100100100100, 0000000000001000)$	$Y13_0$	$W4_0 \oplus W5_0 \oplus W8_0 \oplus W11_0 \oplus W14_0$	1
$(1001100100110101, 0000000000000100)$	$Y14_0$	$W1_0 \oplus W4_0 \oplus W5_0 \oplus W8_0 \oplus W11_0 \oplus W12_0 \oplus W14_0 \oplus W16_0$	1
$(1010010000010001, 0000000000000010)$	$Y15_0$	$W1_0 \oplus W3_0 \oplus W6_0 \oplus W6_0 \oplus W12_0 \oplus W16_0$	1
$(1010110100010101, 0000000000000001)$	$Y16_0$	$W1_0 \oplus W3_0 \oplus W5_0 \oplus W6_0 \oplus W8_0 \oplus W12_0 \oplus W14_0 \oplus W16_0$	1

Սկզբում դիտարկենք PHT ֆունկցիան: Աղյուսակ 4-ում բերված են մուտքային և ելքային որոշ բիթերի կախվածությունը ("0" նշանակում է կախվածություն չկա, "1" նշանակում է բինար գծային կախվածություն, "n" նշանակում է բինար ոչ գծային կախվածություն):

Օրինակ կամայական ϕ ֆունկցիայի համար աղյուսակ 5-ի $Y10_2$ ելքային բիթի համար կունենանք.

$$Y10_2 = W1_1 \oplus W2_2 \oplus W9_2 \oplus W10_2 \oplus W11_1 \oplus W12_1 \oplus \phi(W2_1, W9_1, W10_1; W1_0, W2_0,$$

$$W3_0, W4_0, W9_0, W10_0, W11_0, W12_0; W5, W6, W7, W8, W13, W14, W15, W16):$$

Աղյուսակ 5: Ձևափոխած SAFER+ համակարգի PHT-ի որոշ մուտքային W և ելքային Y բիթերի կախվածությունը:

Մուտքային բիթ / Ելքային բիթ		$W1$	$W2$	$W3$	$W4$	$W9$	$W10$	$W11$	$W12$
		7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1
$Y1$	7	0 0 0 0 1 n n	1 n n n n n n	0 0 1 n n n n	0 1 n n n n n	0 1 n n n n n	1 n n n n n n	0 0 1 n n n n	0 0 1 n n n n
	6	0 0 0 0 0 1 n	0 1 n n n n n	0 0 0 1 n n n	0 0 1 n n n n	0 0 0 1 n n n	0 1 n n n n n	0 0 0 1 n n n	0 0 0 1 n n n
	5	0 0 0 0 0 0 1	0 0 1 n n n n	0 0 0 0 1 n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 1 n n n n	0 0 0 0 1 n n	0 0 0 0 1 n n
	4	0 0 0 0 0 0 0	0 0 0 1 n n n	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 1 n n n	0 0 0 0 0 1 n	0 0 0 0 0 1 n
	3	0 0 0 0 0 0 0	0 0 0 0 1 n n	0 0 0 0 0 0 1	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 1 n n	0 0 0 0 0 0 1	0 0 0 0 0 0 1
	2	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 1 n n	0 0 0 0 0 0 0	0 0 0 0 0 0 0
	1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 0 0 0
	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y2$	7	0 0 0 1 n n n	1 n n n n n n	0 1 n n n n n	0 1 n n n n n	1 n n n n n n	1 n n n n n n	0 0 1 n n n n	0 1 n n n n n
	6	0 0 0 0 1 n n	0 1 n n n n n	0 0 1 n n n n	0 0 1 n n n n	0 1 n n n n n	0 1 n n n n n	0 0 0 1 n n n	0 0 1 n n n n
	5	0 0 0 0 0 1 n	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 0 0 1 n n	0 0 0 1 n n n
	4	0 0 0 0 0 0 1	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 0 1 n	0 0 0 0 1 n n
	3	0 0 0 0 0 0 0	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 0 1	0 0 0 0 0 1 n
	2	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 0 0 0
	1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y3$	7	1 n n n n n n	0 1 n n n n n	1 n n n n n n	0 1 n n n n n	0 0 0 0 1 n n	1 n n n n n n	0 0 1 n n n n	0 1 n n n n n
	6	0 1 n n n n n	0 0 1 n n n n	0 1 n n n n n	0 0 1 n n n n	0 0 0 0 0 1 n	0 1 n n n n n	0 0 0 1 n n n	0 0 1 n n n n
	5	0 0 1 n n n n	0 0 0 1 n n n	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 0 0 0 1	0 0 1 n n n n	0 0 0 0 1 n n	0 0 0 1 n n n
	4	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 0 0 0	0 0 0 1 n n n	0 0 0 0 0 1 n	0 0 0 0 1 n n
	3	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 1 n n	0 0 0 0 0 0 1	0 0 0 0 0 1 n
	2	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 0 0 0
	1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0
	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y4$	7	1 n n n n n n	1 n n n n n n	1 n n n n n n	1 n n n n n n	0 0 0 1 n n n	1 n n n n n n	0 1 n n n n n	0 1 n n n n n
	6	0 1 n n n n n	0 1 n n n n n	0 1 n n n n n	0 1 n n n n n	0 0 0 0 1 n n	0 1 n n n n n	0 0 1 n n n n	0 0 1 n n n n
	5	0 0 1 n n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 0 0 0 1 n	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 1 n n n
	4	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 0 0 1	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 1 n n
	3	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 0 0	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 1 n
	2	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 1
	1	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0
	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y5$	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y6$	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y7$	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y8$	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y9$	7	0 1 n n n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 0 1 n n n	1 n n n n n n	0 1 n n n n n	0 0 0 0 1 n n	1 n n n n n n
	6	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 1 n n n n n	0 0 1 n n n n	0 0 0 0 0 1 n	0 1 n n n n n
	5	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 0 0 0 1	0 0 1 n n n n
	4	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 0 0 0	0 0 0 1 n n n
	3	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 1 n n
	2	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 1 n
	1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1
	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y10$	7	0 1 n n n n n	0 1 n n n n n	0 0 1 n n n n	0 0 1 n n n n	1 n n n n n n	1 n n n n n n	0 0 0 1 n n n	1 n n n n n n
	6	0 0 1 n n n n	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 1 n n n n n	0 1 n n n n n	0 0 0 0 1 n n	0 1 n n n n n
	5	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 1 n n n n	0 0 1 n n n n	0 0 0 0 0 1 n	0 0 1 n n n n
	4	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 0 0 1	0 0 0 1 n n n
	3	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 0 0	0 0 0 0 1 n n
	2	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 1 n
	1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1
	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y11$	7	0 0 1 n n n n	1 n n n n n n	0 0 0 1 n n	1 n n n n n n	0 0 1 n n n n	1 n n n n n n	1 n n n n n n	0 1 n n n n n
	6	0 0 0 1 n n n	0 1 n n n n n	0 0 0 0 1 n n	0 1 n n n n n	0 0 0 1 n n n	0 1 n n n n n	0 1 n n n n n	0 0 1 n n n n
	5	0 0 0 0 1 n n	0 0 1 n n n n	0 0 0 0 0 1 n	0 0 1 n n n n	0 0 0 0 1 n n	0 0 1 n n n n	0 0 1 n n n n	0 0 0 1 n n n
	4	0 0 0 0 0 1 n	0 0 0 1 n n n	0 0 0 0 0 0 1	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 1 n n
	3	0 0 0 0 0 0 1	0 0 0 0 1 n n	0 0 0 0 0 0 0	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 1 n
	2	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 0 1
	1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 0
	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y12$	7	0 1 n n n n n	1 n n n n n n	0 0 0 1 n n n	1 n n n n n n	0 1 n n n n n	1 n n n n n n	1 n n n n n n	1 n n n n n n
	6	0 0 1 n n n n	0 1 n n n n n	0 0 1 n n n n	0 1 n n n n n	0 0 1 n n n n	0 1 n n n n n	0 1 n n n n n	0 1 n n n n n
	5	0 0 0 1 n n n	0 0 1 n n n n	0 0 0 1 n n n	0 0 1 n n n n	0 0 0 1 n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 1 n n n n
	4	0 0 0 0 1 n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 1 n n n
	3	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n
	2	0 0 0 0 0 0 1	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n
	1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 1 n
	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y13$	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y14$	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y15$	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y16$	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0

$S_{a,b}^{\mathrm{PHT}}$ գումարը գծորեն կախված է W_α-ից, եթե $l_b(Y)$-ը գծորեն կախված է W_α-ից, իսկ $a_\alpha = 0$: Քանի որ $a \in \mathcal{A}$, հետևաբար աղյուսակ 5-ի տողերը պարունակում են մուտքային բիթեր, որոնք չեն մասնակցում $l_a(W)$ գումարի մեջ: Եթե $l_b(Y)$-ը որպես գումարելի պարունակում է ելքային բիթ, որը գծորեն կախված է այդպիսի W_α-ից և չի պարունակում ելքային բիթ, որը կախված է W_α-ից, ապա $I\left(S_{a,b}^{\mathrm{PHT}}\right) = 0$:

Հետևաբար հիմնվելով աղյուսակ 5-ի վրա մենք կարող ենք իտերատիվ մեթոդով ցույց տալ, որ զրոյից տարբեր չբալանսավորվածությամբ I/O $S_{a,b}^{\mathrm{PHT}}$ գումարները չեն պարունակում ելքային $Yi_j, j = 1,2,3,4,5,6,7;\ i = 1,2,3,4,9,10,11,12$ բիթերը:

Վերջում դիտարկենք $2^{19} - 1$ բալանսավորված ելքային ֆունկցիաները, որոնք իրանցից ներկայացնում են մնացած 19 ելքային բիթերի գծային կոմբինացիաները (գոնե մեկ ոչ զրոյական անդամով), երբ $b \in \mathcal{B}$: Այդ ելքային ֆունկցիաներից յուրաքանչյուրի համար պետք է գտնել բոլոր մուտքային ֆունկցիաները, պայմանով որ $S_{a,b}^{\mathrm{PHT}}$ գումարը գծային չլինի ցանկացած մի մուտքային բիթի դեպքում: Հեշտ է ցույց տալ, որ $\{(a,b); a,b \in B^{128}, I\left(\mathrm{S}_{a,b}^{\mathrm{PHT}}\right) = 1\}$ բազմությունը $(\mathcal{A} \times \mathcal{B}, \oplus_{256\mathrm{bits}})$-ի ենթախումբն է:

(Իրոք, եթե $I\left(\mathrm{S}_{a,b}^{\mathrm{PHT}}\right) = 1$ և $I\left(\mathrm{S}_{a',b'}^{\mathrm{PHT}}\right) = 1$ կամայական a' և b' 128 երկարության վեկտորների համար, ապա

$$I\left(\mathrm{S}_{a\oplus a',b\oplus b'}^{\mathrm{PHT}}\right) = I\left(\mathrm{S}_{a,b}^{\mathrm{PHT}} \oplus \mathrm{S}_{a',b'}^{\mathrm{PHT}}\right) = I\left(\mathrm{S}_{a,b}^{\mathrm{PHT}}\right) \cdot I\left(\mathrm{S}_{a',b'}^{\mathrm{PHT}}\right) = 1,$$

հետևաբար $\mathcal{A} \times \mathcal{B}$ խումբը փակ է "$\oplus$" գործողության նկատմամբ:) Այսպիսով կստանանք ոչ զրոյական չբալանսավորվածությամբ բոլոր I/O գումարները, ինչպես նշված է լեմմա 3.2-ում:

Հիմա դիտարկենք ռաունդի այն կեսը, որը պարունակում է ոչ գծային (NL) մասը և ոչ գծային մասի համար գտնենք ոչ զրոյական չբալանսավորվածությամբ բոլոր հոմոմորֆ I/O գումարները: Այս դեպքում մուտքային ֆունկցիան հոմոմորֆ է XOR/ADD-ի համար, իսկ ելքային ֆունկցիան ADD/XOR-ի համար (տե՛ս Սահմանում 3.1): Այսպիսի I/O գումարներ կարելի է ստանալ EXP և LOG բլոկերի I/O գումարների գումարի արդյունքում: U1 մուտքային և V1 ելքային բայթերի դեպքում EXP ֆունկցիայի հոմոմորֆ I/O գումարները կորոշվեն այսպես.

$$S_{a1,b1}^{\mathrm{EXP}} = l_{a1}(U1) \oplus l_{b1}(V1), \quad a1 \in B^8 \backslash \{00\}, b1 = 01:$$

Ամենաէֆեկտիվ I/O գումարները կստացվեն, եթե $(a1, b1)$ հավասար է $(cd, 01)$ կամ $(ff, 01)$, (չբալանսավորվածությունը այս դեպքերում $\frac{28}{128}$ է) կամ $(86{,}01), (bf, 01), (c0, 01)$ կամ $(f7{,}01)$ (չբալանսավորվածությունը այս դեպքերում $\frac{24}{128}$ է):

Դիտողություն 3.2: $I(S_{01,01}^{\mathrm{EXP}}) = I(S_{02,01}^{\mathrm{EXP}}) = I(S_{03,01}^{\mathrm{EXP}}) = 0$: Ավելին բոլոր $a1$,$b1 \in B^8$ համար, եթե $a1_7 = 0$, ապա $I(S_{a1,b1}^{\mathrm{EXP}}) = 0$:

LOG ֆունկցիայի հոմոմորֆ I/O գումարները մուտքային U2 և ելքային V2 բայթերի համար որոշվում է այսպես.

$$S_{a2,b2}^{\mathrm{LOG}} = l_{a2}(\mathrm{U2}) \oplus l_{b2}(\mathrm{V2}); \ \ a2 = 01; \ b2 \in B^8 \backslash \{00\}:$$

Հարկ է նկատել, որ $I(S_{a1,b1}^{\mathrm{EXP}}) = I(S_{b1,a1}^{\mathrm{LOG}})$.

Դիտողություն 3.3: Ցանկացած $a1, b1 \in B^8$, եթե $b1_7 = 0$, ապա $\mathrm{I}(\mathrm{S}_{a1,b1}^{\mathrm{LOG}}) = 0$:

Վերջում կապակցենք հաջորդական կիսառաունդների I/O գումարները:

Թեորեմ 3.3: Էֆեկտիվ հոմոմորֆ I/O գումարներ գտնելու մեթոդի միջոցով հնարավոր չի գտնել ոչ զրոյական չբալանսավորվածությամբ I/O գումար ձևափոխած SAFER+ համակարգի կիսառունդների հաջորդականության համար, որը պարունակում է երկու PHT շերտ:

Ապացույց: Դիցուք $T_{a,b}^{\mathrm{PHT-hr1}}$, $T_{b,c}^{\mathrm{NL}}$ և $T_{c,d}^{\mathrm{PHT-hr2}}$ կիսառաունդների հոմոմորֆ կապակցված եռակի գումարներ են՝ մաքսիմալացնող առանցքային ֆունկցիաներով: Եթե $T_{a,b}^{\mathrm{PHT-hr}}$ և $T_{c,d}^{\mathrm{PHT-hr}}$ եռակի գումարների չբալանսավորվածությունը զրոյից տարբեր է, 128 բիթ ոչ զրոյական a, b, c, d դիմակների բայթերի միայն վերջին երկու բիթերը կարող են լինել 1 (տե՛ս Լեմմա 3.2): Հետևաբար $I(T_{b,c}^{\mathrm{NL}}) = 0$, քանի որ I/O գումարի բանալուց կախված միջին չբալանսավորվածությունը նույնպես 0 է (տե՛ս Դիտողություն 3.2): Այսպիսով այս երեք կիսառաունդների եռակի գումարների գումարի չբալանսավորվածությունը 0 է:

Ամենաէֆեկտիվ I/O գումարներից մեկը $S_{a,b}^{\mathrm{NL-PHT-NL}}$ գումարն է, որտեղ a և b վեկտորների $a2, a4, a11$ և $b5, b6$ բայթերը cd է կամ ff, իսկ մյուս բայթերը 0 են: Նրանց

չբալանսավորվածությունը $\left(\frac{28}{128}\right)^5$է, քանի որ $\left(S^{\mathrm{EXP}}_{ff,01}\right) = I\left(S^{\mathrm{EXP}}_{cd,01}\right) = I\left(S^{\mathrm{LOG}}_{01,ff}\right) = I\left(S^{\mathrm{LOG}}_{01,cd}\right) = \frac{28}{128}$, իսկ $I\left(S^{\mathrm{PHT}}_{a,b}\right) = I\left(S^{\mathrm{PHT}}_{a1\oplus a2,b1\oplus b2}\right) = I\left(S^{\mathrm{PHT}}_{a1,b1}\right)\cdot I\left(S^{\mathrm{PHT}}_{a2,b2}\right) = 1$, եթե

$$a1\oplus a2 = (00\ 00\ 00\ 00\ 00\ 01\ 01\ 00\ 01\ 00\ 00\ 01\ 00\ 01\ 00\ 00)\oplus$$

$$(00\ 01\ 00\ 01\ 00\ 01\ 01\ 00\ 01\ 00\ 01\ 01\ 00\ 01\ 00\ 00) =$$

$$(00\ 01\ 00\ 01\ 00\ 00\ 00\ 00\ 00\ 00\ 01\ 00\ 00\ 00\ 00\ 00) = a,$$

$$b1\oplus b2 = (\ 00\ 00\ 00\ 00\ 01\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00)\oplus$$

$$(00\ 00\ 00\ 00\ 00\ 01\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00) =$$

$$(00\ 00\ 00\ 00\ 01\ 01\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00) = b:$$

Այսպիսով, էֆեկտիվ հոմոմորֆ I/O գումար գտնելու մեթոդի միջոցով հնարավոր չի գտնել ոչ զրոյական չբալանսավորվածությամբ I/O գումար ձևափոխած SAFER+ համակարգի 2 ռաունդի համար, հետևաբար գոյություն չունի այնպիսի հոմոմորֆ I/O գումար, որի շնորհիվ գծային վերլուծության ընդհանրացված այս մեթոդը լինի էֆեկտիվ: Այսինքն SAFER+ համակարգի արագագործությունը բարելավող ձևափոխություններից հետո գծային վերլուծության նկատմամբ կայունությունը պահպանվում է:

3.3. SAFER-256 համակարգի գծային վերլուծությունը

SAFER-256 համակարգի գծային վերլուծության նպատակով համակարգի նկար 11–ում պատկերված ռաունդը բաժանենք երկու կիսառունդների այնպես ինչպես դա արվեց ձևափոխած SAFER+ համակարգի դեպքում.

1. Ոչ գծային կիսառաունդ (NL half-round).

   1.1. $\mathrm{XOR/ADD} - \mathrm{U} = \mathrm{XOR/ADD}(\mathrm{X}, \mathrm{K}_{2\mathrm{i}-1})$:

   1.2. $\mathrm{NL\ (Non\ Linear)} - \mathrm{V} = \mathrm{NL}(\mathrm{U})$:

2. PHT կիսառաունդ (PHT half-round).

   2.1. $\mathrm{ADD/XOR} - \mathrm{W} = \mathrm{ADD/XOR}(\mathrm{V}, \mathrm{K}_{2\mathrm{i}})$:

   2.2. $\mathrm{PHT(Pseudo\ Hadamard\ Transform)} -$ հակադարձելի գծային ձևափոխության մատրիցը նկար 14–ում պատկերված մատրիցն է. $\mathrm{Y} = \mathrm{PHT}(\mathrm{W})$:

Ինչպես նախորդ համակարգի գծային վերլուծության դեպքում գտնենք SAFER-256 համակարգի կիսառաունդների բոլոր էֆեկտիվ հոմոմորֆ I/O գումարները: Սկզբում գտնենք բոլոր բինար հոմոմորֆիզմները $\mathrm{ADD/XOR}$ և $\mathrm{XOR/ADD}$ գործողությունների համար: Քանի որ գոյություն ունեն $2^8 - 1$ բինար հոմոմորֆիզմ 8 բիթերի XOR գործողության համար և միայն մեկ բինար հոմոմորֆիզմ ըստ մոդուլ 256-ի գումար գործողության համար, հետևաբար գոյություն կունենա $2^{144} - 1$ բալանսավորված հոմոմորֆիզմ $\mathrm{ADD/XOR}$ գործողության համար, որոնք որոշվում են այսպես. $l_a(V) = a \circ V$, որտեղ a-ն

$$\mathcal{A} = \{a: a \in \{0,1\}^{256} \backslash \{00\}; a1, a2, a3, a4, a9, a10, a11, a12, a17, a18, a19, a20, a25, a26, a27, a28\} \in \{00, 01\}\}$$

բազմության վեկտոր է:

Նույն տրամաբանությամբ գոյություն ունի $2^{144} - 1$ բալանսավորված հոմոմորֆիզմ $\mathrm{XOR/ADD}$ գործողության համար. $l_b(V) = b * V$, որտեղ b-ն պատկանում է

$$\mathcal{B} = \{b: b \in \{0,1\}^{256} \backslash \{00\}; b5, b6, b7, b8, b13, b14, b15, b16, b21, b22, b23, b24,$$
$$b29, b30, b31, b32\} \in \{00, 01\}\}$$

բազմությանը:

Դիտարկենք համակարգի ռաունդի այն կեսը, որը պարունակում է PHT ֆունկցիան: Գոյություն ունի $(2^{144} - 1)^2$ I/O գումար, որոնք որոշվում են հետևյալ բանաձևով.

$$\mathrm{S}_{a,b}^{\mathrm{PHT-hr}} := l_a(\mathrm{V}) \oplus l_b(\mathrm{Y}) \quad a \in \mathcal{A}\,, b \ \in \mathcal{B},$$

և որոնց համար մուտքային ֆունկցիան բալանսավորված և հոմոմորֆ է ADD/XOR-ի համար, ելքային ֆունկցիան բալանսավորված և հոմոմորֆ է XOR/ADD-ի համար:

Լեմմա 3.3: SAFER-256 համակարգի PHT կիսառաունդի (PHT-hr) հոմոմորֆ I/O գումարները, որոնք ունեն ոչ զրոյական չբալանսավորվածություն դրանք $2^{32} - 1$ երաշխավորված I/O գումարներն են, որոնք ստացվում են աղյուսակ 6-ում բերված 32 երաշխավորված I/O գումարներից հաշվելի թվով գումարներ XOR-անելով:

Իսկ հիմա դիտարկենք PHT ֆունկցիան: Աղյուսակ 7-ում բերված են որոշ մուտքային և ելքային բիթերի կախվածությունը ("0" նշանակում է կախվածություն չկա, "1" նշանակում է բինար գծային կախվածություն, "n" նշանակում է բինար ոչ գծային կախվածություն): Աղյուսակ 7-ից ելնելով իտերատիվ մեթոդով կարելի է ցույց տալ, որ ոչ զրոյական չբալանսավորվածությամբ $S_{a,b}^{\mathrm{PHT}}$ I/O գումարները չեն պարունակում Yi_j, $i = 1,2,3,4,9,10,11,12,17,18,19,20,25,26,27,28$, $j = 1,2,3,4,5,6,7$ ելքային բիթերը:

$2^{40} - 1$ բալանսավորված ելքային ֆունկցիաների համար, որոնք մնացած 40 ելքային բիթերի գծային կոմբինացիաներն են (գոնե մեկ ոչ զրոյական անդամով), երբ $b \in \mathcal{B}$, պետք է գտնել բոլոր մուտքային ֆունկցիաները, պայմանով որ $S_{a,b}^{\mathrm{PHT}}$ գումարը գծային չլինի ցանկացած մի մուտքային բիթի դեպքում: $\{(a, b); a, b \in B^{256}, I(\mathrm{S}_{a,b}^{\mathrm{PHT}}) = 1\}$ բազմությունը $(\mathcal{A} \times \mathcal{B}, \oplus_{256\mathrm{bits}})$ -ի ենթախումբն է: (Իրոք, եթե $I(\mathrm{S}_{a,b}^{\mathrm{PHT}}) = 1$ և $I(\mathrm{S}_{a',b'}^{\mathrm{PHT}}) = 1$ կամայական a' և b' 256 երկարության վեկտորների համար, ապա $I(\mathrm{S}_{a\oplus a',b\oplus b'}^{\mathrm{PHT}}) = I(\mathrm{S}_{a,b}^{\mathrm{PHT}} \oplus \mathrm{S}_{a',b'}^{\mathrm{PHT}}) = I(\mathrm{S}_{a,b}^{\mathrm{PHT}}) \cdot I(\mathrm{S}_{a',b'}^{\mathrm{PHT}}) = 1$, հետևաբար $\mathcal{A} \times \mathcal{B}$ խումբը փակ է "⊕" գործողության նկատմամբ:) Այսպիսով կստացվի զրոյից տարբեր չբալանսավորվածությամբ բոլոր I/O գումարները, ինչպես նշված է լեմմա 3.3-ում:

Աղյուսակ 6: SAFER-256 ալգորիթմի PHT ֆունկցիայի էֆեկտիվ I/O գումարները:

(a,b)	$l_b(Y)$	$l_a(W)$	$I(S_{a,b}^{\mathrm{PHT}})$
(00000000000110011001000010010001, 10000000000000000000000000000000)	$Y1_0$	$W12_0 \oplus W13_0 \oplus W16_0 \oplus W17_0 \oplus W20_0 \oplus W25_0 \oplus W28_0 \oplus W32_0$	1
(00000000000110011111000011110101, 01000000000000000000000000000000)	$Y2_0$	$W12_0 \oplus W13_0 \oplus W16_0 \oplus W17_0 \oplus W18_0 \oplus W19_0 \oplus W20_0 \oplus W25_0 \oplus W26_0 \oplus W27_0 \oplus W28_0 \oplus W30_0 \oplus W32_0$	1
(10010001100100000001100100000000, 00100000000000000000000000000000)	$Y3_0$	$W1_0 \oplus W4_0 \oplus W8_0 \oplus W9_0 \oplus W12_0 \oplus W20_0 \oplus W21_0 \oplus W24_0$	1
(10010001100100000001111101000110, 00010000000000000000000000000000)	$Y4_0$	$W1_0 \oplus W4_0 \oplus W8_0 \oplus W9_0 \oplus W12_0 \oplus W20_0 \oplus W21_0 \oplus W22_0 \oplus W23_0 \oplus W24_0 \oplus W26_0 \oplus W30_0 \oplus W31_0$	1
(00011001000000001001000110010000, 00001000000000000000000000000000)	$Y5_0$	$W4_0 \oplus W5_0 \oplus W8_0 \oplus W17_0 \oplus W20_0 \oplus W24_0 \oplus W25_0 \oplus W28_0$	1
(00011001000000001111010111110000, 00000100000000000000000000000000)	$Y6_0$	$W4_0 \oplus W5_0 \oplus W8_0 \oplus W17_0 \oplus W18_0 \oplus W19_0 \oplus W20_0 \oplus W22_0 \oplus W24_0 \oplus W25_0 \oplus W26_0 \oplus W27_0 \oplus W28_0$	1
(00000110010001100100011000000000, 00000010000000000000000000000000)	$Y7_0$	$W6_0 \oplus W7_0 \oplus W10_0 \oplus W14_0 \oplus W15_0 \oplus W18_0 \oplus W22_0 \oplus W23_0$	1
(00000110010001100100011110011001, 00000001000000000000000000000000)	$Y8_0$	$W6_0 \oplus W7_0 \oplus W10_0 \oplus W14_0 \oplus W15_0 \oplus W18_0 \oplus W21_0 \oplus W22_0 \oplus W23_0 \oplus W24_0 \oplus W28_0 \oplus W29_0 \oplus W32_0$	1
(01100000110010001100100000000000, 00000000100000000000000000000000)	$Y9_0$	$W2_0 \oplus W3_0 \oplus W10_0 \oplus W11_0 \oplus W14_0 \oplus W18_0 \oplus W19_0 \oplus W22_0$	1
(01111001011011010110010000000000, 00000000010000000000000000000000)	$Y10_0$	$W2_0 \oplus W3_0 \oplus W4_0 \oplus W5_0 \oplus W8_0 \oplus W10_0 \oplus W11_0 \oplus W13_0 \oplus W14_0 \oplus W16_0 \oplus W18_0 \oplus W19_0 \oplus W22_0$	1
(10010000100100010000000000011001, 00000000001000000000000000000000)	$Y11_0$	$W1_0 \oplus W4_0 \oplus W9_0 \oplus W12_0 \oplus W16_0 \oplus W28_0 \oplus W29_0 \oplus W32_0$	1
(10010000100100010100011000011111, 00000000000100000000000000000000)	$Y12_0$	$W1_0 \oplus W4_0 \oplus W9_0 \oplus W12_0 \oplus W16_0 \oplus W18_0 \oplus W22_0 \oplus W23_0 \oplus W28_0 \oplus W29_0 \oplus W30_0 \oplus W31_0 \oplus W32_0$	1
(10010001000000000001100100001001, 00000000000010000000000000000000)	$Y13_0$	$W1_0 \oplus W4_0 \oplus W8_0 \oplus W20_0 \oplus W21_0 \oplus W24_0 \oplus W29_0 \oplus W32_0$	1
(11010111000001100001100100001001, 00000000000001000000000000000000)	$Y14_0$	$W1_0 \oplus W2_0 \oplus W4_0 \oplus W6_0 \oplus W7_0 \oplus W8_0 \oplus W14_0 \oplus W15_0 \oplus W20_0 \oplus W21_0 \oplus W24_0 \oplus W29_0 \oplus W32_0$	1
(00001001000110010000000010010001, 00000000000000100000000000000000)	$Y15_0$	$W5_0 \oplus W8_0 \oplus W12_0 \oplus W13_0 \oplus W16_0 \oplus W25_0 \oplus W28_0 \oplus W32_0$	1
(01101101011110010000000010010001, 00000000000000010000000000000000)	$Y16_0$	$W2_0 \oplus W3_0 \oplus W5_0 \oplus W6_0 \oplus W8_0 \oplus W10_0 \oplus W11_0 \oplus W12_0 \oplus W13_0 \oplus W16_0 \oplus W25_0 \oplus W28_0 \oplus W32_0$	1
(01100100011000000000000001100100, 00000000000000001000000000000000)	$Y17_0$	$W2_0 \oplus W3_0 \oplus W6_0 \oplus W10_0 \oplus W11_0 \oplus W26_0 \oplus W27_0 \oplus W30_0$	1
(01101101011110010000000001100100, 00000000000000000100000000000000)	$Y18_0$	$W2_0 \oplus W3_0 \oplus W5_0 \oplus W6_0 \oplus W8_0 \oplus W10_0 \oplus W11_0 \oplus W12_0 \oplus W13_0 \oplus W16_0 \oplus W26_0 \oplus W27_0 \oplus W30_0$	1
(01000110000001100000000001000110, 00000000000000000010000000000000)	$Y19_0$	$W2_0 \oplus W6_0 \oplus W7_0 \oplus W14_0 \oplus W15_0 \oplus W26_0 \oplus W30_0 \oplus W31_0$	1
(01000110000001100001100101001111, 00000000000000000001000000000000)	$Y20_0$	$W2_0 \oplus W6_0 \oplus W7_0 \oplus W14_0 \oplus W15_0 \oplus W20_0 \oplus W21_0 \oplus W24_0 \oplus W26_0 \oplus W29_0 \oplus W30_0 \oplus W31_0 \oplus W32_0$	1
(00000000010001100100011000000110, 00000000000000000000100000000000)	$Y21_0$	$W10_0 \oplus W14_0 \oplus W15_0 \oplus W18_0 \oplus W22_0 \oplus W23_0 \oplus W30_0 \oplus W31_0$	1
(10010000110101110100011000000110, 00000000000000000000010000000000)	$Y22_0$	$W1_0 \oplus W4_0 \oplus W9_0 \oplus W10_0 \oplus W12_0 \oplus W14_0 \oplus W15_0 \oplus W16_0 \oplus W18_0 \oplus W22_0 \oplus W23_0 \oplus W30_0 \oplus W31_0$	1
(00011001000010011001000100000000, 00000000000000000000001000000000)	$Y23_0$	$W4_0 \oplus W5_0 \oplus W8_0 \oplus W13_0 \oplus W16_0 \oplus W17_0 \oplus W20_0 \oplus W24_0$	1
(01111001011011011001000100000000, 00000000000000000000000100000000)	$Y24_0$	$W2_0 \oplus W3_0 \oplus W4_0 \oplus W5_0 \oplus W8_0 \oplus W10_0 \oplus W11_0 \oplus W13_0 \oplus W14_0 \oplus W16_0 \oplus W17_0 \oplus W20_0 \oplus W24_0$	1
(01100100000000000110000001100100, 00000000000000000000000010000000)	$Y25_0$	$W2_0 \oplus W3_0 \oplus W6_0 \oplus W18_0 \oplus W19_0 \oplus W26_0 \oplus W27_0 \oplus W30_0$	1
(01100100000000000111100001110101, 00000000000000000000000001000000)	$Y26_0$	$W2_0 \oplus W3_0 \oplus W6_0 \oplus W17_0 \oplus W18_0 \oplus W19_0 \oplus W20_0 \oplus W25_0 \oplus W26_0 \oplus W27_0 \oplus W28_0 \oplus W30_0 \oplus W32_0$	1
(01000110000000000000011001000110, 00000000000000000000000000100000)	$Y27_0$	$W2_0 \oplus W6_0 \oplus W7_0 \oplus W22_0 \oplus W23_0 \oplus W26_0 \oplus W30_0 \oplus W31_0$	1
(11010111100100000000011001000110, 00000000000000000000000000010000)	$Y28_0$	$W1_0 \oplus W2_0 \oplus W4_0 \oplus W6_0 \oplus W7_0 \oplus W8_0 \oplus W9_0 \oplus W12_0 \oplus W22_0 \oplus W23_0 \oplus W26_0 \oplus W30_0 \oplus W31_0$	1
(00000000011001000110010001100000, 00000000000000000000000000001000)	$Y29_0$	$W10_0 \oplus W11_0 \oplus W14_0 \oplus W18_0 \oplus W19_0 \oplus W22_0 \oplus W26_0 \oplus W27_0$	1
(00000000011001001111010111110000, 00000000000000000000000000000100)	$Y30_0$	$W10_0 \oplus W11_0 \oplus W14_0 \oplus W17_0 \oplus W18_0 \oplus W19_0 \oplus W20_0 \oplus W22\_0 \oplus W24\_0 \oplus W25_0 \oplus W26_0 \oplus W27_0 \oplus W28_0$	1
(00000000100100010000100100011001, 00000000000000000000000000000010)	$Y31_0$	$W9_0 \oplus W12_0 \oplus W16_0 \oplus W21_0 \oplus W24_0 \oplus W28_0 \oplus W29_0 \oplus W32_0$	1
(00000110110101110000100100011001, 00000000000000000000000000000001)	$Y32_0$	$W6_0 \oplus W7_0 \oplus W9_0 \oplus W10_0 \oplus W12_0 \oplus W14_0 \oplus W15_0 \oplus W16_0 \oplus W21_0 \oplus W24_0 \oplus W28_0 \oplus W29_0 \oplus W32_0$	1

Աղյուսակ 7: SAFER-256 համակարգի PHT-ի որոշ մուտքային W և ելքային Y բիթերի կախվածությունը:

Մուտքային բիթ / Ելքային բիթ		$W1$	$W2$	$W3$	$W4$	$W9$	$W10$	$W11$	$W12$	$W17$	$W18$	$W19$	$W20$	$W25$	$W26$	$W27$	$W28$
		7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1	7 6 5 4 3 2 1
$Y1$	7	0 0 0 1 n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 1 n n n n	0 1 n n n n n	0 0 0 1 n n n	0 0 0 1 n n n	1 n n n n n n	1 n n n n n n	0 1 n n n n n	0 1 n n n n n	1 n n n n n n	1 n n n n n n	0 1 n n n n n	0 1 n n n n n	1 n n n n n n
	6	0 0 0 0 1 n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 1 n n n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 1 n n n n n	0 1 n n n n n	0 0 1 n n n n	0 0 1 n n n n	0 1 n n n n n	0 1 n n n n n	0 0 1 n n n n	0 0 1 n n n n	0 1 n n n n n
	5	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 1 n n n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 1 n n n n	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 1 n n n n
	4	0 0 0 0 0 0 1	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 1 n n n
	3	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 1 n n
	2	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 1 n
	1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1
	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y2$	7	0 0 0 1 n n n	0 1 n n n n n	0 1 n n n n n	0 0 1 n n n n	0 1 n n n n n	0 0 1 n n n n	0 0 1 n n n n	1 n n n n n n	1 n n n n n n	1 n n n n n n	1 n n n n n n	1 n n n n n n	1 n n n n n n	1 n n n n n n	1 n n n n n n	1 n n n n n n
	6	0 0 0 0 1 n n	0 0 1 n n n n	0 0 1 n n n n	0 0 0 1 n n n	0 0 1 n n n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 1 n n n n n	0 1 n n n n n	0 1 n n n n n	0 1 n n n n n	0 1 n n n n n	0 1 n n n n n	0 1 n n n n n	0 1 n n n n n	0 1 n n n n n
	5	0 0 0 0 0 1 n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 1 n n n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 1 n n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 1 n n n n
	4	0 0 0 0 0 0 1	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 1 n n n
	3	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n
	2	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 1 n
	1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 1
	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y3$	7	1 n n n n n n	0 0 1 n n n n	0 0 0 1 n n n	1 n n n n n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 0 1	1 n n n n n n	0 1 n n n n n	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 1 n n n n	0 0 1 n n n n	0 0 1 n n n n
	6	0 1 n n n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 1 n n n n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 1 n n n n n	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 1 n n n
	5	0 0 1 n n n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 1 n n n n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 1 n n
	4	0 0 0 1 n n n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 1 n n n	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 1 n
	3	0 0 0 0 1 n n	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 1 n n	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
	2	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
	1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y4$	7	1 n n n n n n	0 1 n n n n n	1 n n n n n n	1 n n n n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 1 n n n n n	0 1 n n n n n	0 1 n n n n n	0 0 1 n n n n	1 n n n n n n	0 0 0 1 n n n	1 n n n n n n	0 1 n n n n n	0 0 1 n n n n
	6	0 1 n n n n n	0 0 1 n n n n	0 1 n n n n n	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 1 n n n n n	0 0 1 n n n n	0 0 1 n n n n	0 0 0 1 n n n	0 1 n n n n n	0 0 0 0 1 n n	0 1 n n n n n	0 0 1 n n n n	0 0 0 1 n n n
	5	0 0 1 n n n n	0 0 0 1 n n n	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 1 n n n n	0 0 0 0 0 1 n	0 0 1 n n n n	0 0 0 1 n n n	0 0 0 0 1 n n
	4	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 1 n n n	0 0 0 1 n n n	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 1 n n n	0 0 0 0 0 0 1	0 0 0 1 n n n	0 0 0 0 1 n n	0 0 0 0 0 1 n
	3	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 1 n n	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 1 n n	0 0 0 0 0 1 n	0 0 0 0 0 0 1
	2	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 1 n	0 0 0 0 0 1 n	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 1 n	0 0 0 0 0 0 1	0 0 0 0 0 0 0
	1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 0
	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y5$	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y6$	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y7$	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
$Y8$	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0

Y9	7	0011nnnn	1nnnnnnn	01nnnnnn	0001nnnn	1nnnnnnn	0011nnnn	0001nnnn	1nnnnnnn	0011nnnn	0001nnnn	1nnnnnnn	0011nnnn	0001nnnn	1nnnnnnn	0011nnnn	0001nnnn
	6	0001nnnn	01nnnnnn	001nnnnn	0001nnnn	01nnnnnn	0001nnnn	0001nnnn	01nnnnnn	0001nnnn	0001nnnn	01nnnnnn	0001nnnn	0001nnnn	01nnnnnn	0001nnnn	0001nnnn
	5	0000nnnn	001nnnnn	0001nnnn	0000nnnn	001nnnnn	0000nnnn	0000nnnn	001nnnnn	0000nnnn	0000nnnn	001nnnnn	0000nnnn	0000nnnn	001nnnnn	0000nnnn	0000nnnn
	4	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn
	3	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n
	2	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
	1	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
	0	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
Y10	7	01nnnnnn	1nnnnnnn	01nnnnnn	001nnnnn	1nnnnnnn	01nnnnnn	001nnnnn	1nnnnnnn	01nnnnnn	001nnnnn	1nnnnnnn	01nnnnnn	001nnnnn	1nnnnnnn	01nnnnnn	001nnnnn
	6	001nnnnn	01nnnnnn	001nnnnn	0001nnnn	01nnnnnn	001nnnnn	0001nnnn	01nnnnnn	001nnnnn	0001nnnn	01nnnnnn	001nnnnn	0001nnnn	01nnnnnn	001nnnnn	0001nnnn
	5	0001nnnn	001nnnnn	0001nnnn	0000nnnn	001nnnnn	0001nnnn	0000nnnn	001nnnnn	0001nnnn	0000nnnn	001nnnnn	0001nnnn	0000nnnn	001nnnnn	0001nnnn	0000nnnn
	4	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn
	3	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n
	2	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
	1	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
	0	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
Y11	7	1nnnnnnn	0000001n	1nnnnnnn	0001nnnn	1nnnnnnn	0001nnnn	0001nnnn	1nnnnnnn	0001nnnn	0001nnnn	1nnnnnnn	0001nnnn	0001nnnn	1nnnnnnn	0001nnnn	0001nnnn
	6	01nnnnnn	0000001n	01nnnnnn	0001nnnn	01nnnnnn	0001nnnn	0001nnnn	01nnnnnn	0001nnnn	0001nnnn	01nnnnnn	0001nnnn	0001nnnn	01nnnnnn	0001nnnn	0001nnnn
	5	001nnnnn	0000001n	001nnnnn	0000nnnn	001nnnnn	0000nnnn	0000nnnn	001nnnnn	0000nnnn	0000nnnn	001nnnnn	0000nnnn	0000nnnn	001nnnnn	0000nnnn	0000nnnn
	4	0001nnnn	0000001n	0001nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn
	3	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n
	2	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
	1	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
	0	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
Y12	7	1nnnnnnn	0000001n	1nnnnnnn	0001nnnn	1nnnnnnn	0001nnnn	0001nnnn	1nnnnnnn	0001nnnn	0001nnnn	1nnnnnnn	0001nnnn	0001nnnn	1nnnnnnn	0001nnnn	0001nnnn
	6	01nnnnnn	0000001n	01nnnnnn	0001nnnn	01nnnnnn	0001nnnn	0001nnnn	01nnnnnn	0001nnnn	0001nnnn	01nnnnnn	0001nnnn	0001nnnn	01nnnnnn	0001nnnn	0001nnnn
	5	001nnnnn	0000001n	001nnnnn	0000nnnn	001nnnnn	0000nnnn	0000nnnn	001nnnnn	0000nnnn	0000nnnn	001nnnnn	0000nnnn	0000nnnn	001nnnnn	0000nnnn	0000nnnn
	4	0001nnnn	0000001n	0001nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn	0001nnnn	0000nnnn	0000nnnn
	3	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n
	2	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
	1	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
	0	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
Y13	0	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
Y14	0	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
Y15	0	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
Y16	0	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n

$Y17$	7	0001nnn	1nnnnnn	001nnnn	01nnnnn	001nnnn	1nnnnnn	01nnnnn	0001nnn	01nnnnn	0001nnn	001nnnn	1nnnnnn	001nnnn	1nnnnnn	001nnnn	1nnnnnn	001nnnn	001nnnn
	6	0000001	01nnnnn	0001nnn	001nnnn	0000001	01nnnnn	001nnnn	0000001	001nnnn	0000001	0001nnn	01nnnnn	0001nnn	01nnnnn	0001nnn	01nnnnn	0001nnn	0000001
	5	000001n	001nnnn	0000001	0001nnn	0000001	001nnnn	0001nnn	0000001	0001nnn	0000001	0000001	001nnnn	0000001	001nnnn	0000001	001nnnn	0000001	000001n
	4	0000001	0001nnn	0000001	0000001	0000001	0001nnn	0000001	0000001	0000001	0000001	0000001	0001nnn	0000001	0001nnn	0000001	0001nnn	0000001	0000001
	3	0000000	000001n	0000000	0000001	0000000	000001n	0000000	0000000	0000000	0000000	0000000	000001n	0000000	000001n	0000000	000001n	0000000	0000000
	2	0000000	0000001	0000000	0000000	0000000	0000001	0000000	0000000	0000000	0000000	0000000	0000001	0000000	0000001	0000000	0000001	0000000	0000000
	1	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$Y18$	7	001nnnn	1nnnnnn	01nnnnn	1nnnnnn	01nnnnn	1nnnnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	01nnnnn	1nnnnnn	01nnnnn	1nnnnnn	01nnnnn	1nnnnnn	01nnnnn	01nnnnn
	6	0001nnn	01nnnnn	001nnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	0001nnn	001nnnn	0001nnn	001nnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	001nnnn
	5	0000001	001nnnn	0001nnn	001nnnn	0001nnn	001nnnn	0001nnn	0000001	0001nnn	0000001	0001nnn	001nnnn	0001nnn	001nnnn	0001nnn	001nnnn	0001nnn	0001nnn
	4	0000001	0001nnn	0000001	0001nnn	0000001	0001nnn	0000001	0000001	0000001	0000001	0000001	0001nnn	0000001	0001nnn	0000001	0001nnn	0000001	0000001
	3	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001
	2	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
	1	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$Y19$	7	001nnnn	1nnnnnn	01nnnnn	1nnnnnn	01nnnnn	1nnnnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	01nnnnn	1nnnnnn	01nnnnn	1nnnnnn	01nnnnn	1nnnnnn	01nnnnn	01nnnnn
	6	0001nnn	01nnnnn	001nnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	0001nnn	001nnnn	0001nnn	001nnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	001nnnn
	5	0000001	001nnnn	0001nnn	001nnnn	0001nnn	001nnnn	0001nnn	0000001	0001nnn	0000001	0001nnn	001nnnn	0001nnn	001nnnn	0001nnn	001nnnn	0001nnn	0001nnn
	4	0000001	0001nnn	0000001	0001nnn	0000001	0001nnn	0000001	0000001	0000001	0000001	0000001	0001nnn	0000001	0001nnn	0000001	0001nnn	0000001	0000001
	3	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001
	2	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
	1	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$Y20$	7	01nnnnn	1nnnnnn	01nnnnn	1nnnnnn	01nnnnn	1nnnnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	01nnnnn	1nnnnnn	01nnnnn	1nnnnnn	01nnnnn	1nnnnnn	01nnnnn	01nnnnn
	6	001nnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	0001nnn	001nnnn	0001nnn	001nnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	01nnnnn	001nnnn	001nnnn
	5	0001nnn	001nnnn	0001nnn	001nnnn	0001nnn	001nnnn	0001nnn	0000001	0001nnn	0000001	0001nnn	001nnnn	0001nnn	001nnnn	0001nnn	001nnnn	0001nnn	0001nnn
	4	0000001	0001nnn	0000001	0001nnn	0000001	0001nnn	0000001	0000001	0000001	0000001	0000001	0001nnn	0000001	0001nnn	0000001	0001nnn	0000001	0000001
	3	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001	0000001
	2	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
	1	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$Y21$	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$Y22$	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$Y23$	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$Y24$	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000

$Y25$	7	0000001n	1nnnnnnn	0000001nn	0001nnnn	01nnnnnn	001nnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn
	6	0000001	01nnnnnn	0000001n	0001nnnn	01nnnnnn	001nnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn
	5	0000000	001nnnnn	0000001	0001nnnn	01nnnnnn	001nnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn
	4	0000000	0001nnnn	0000000	0001nnnn	01nnnnnn	001nnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn
	3	0000000	0001nnnn	0000000	0001nnnn	01nnnnnn	001nnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn
	2	0000000	0000001n	0000000	0000001n	00000001	000001nn	00000001	0000001n	00000001	0000001n	00000001	0000001n	00000001	0000001n	00000001	0000001n
	1	0000000	0000001	0000000	0000001	00000000	000000nn	00000000	0000001	00000000	0000001	00000000	0000001	00000000	0000001	00000000	0000001
	0	0000000	0000000	0000000	0000000	00000000	000000nn	00000000	0000000	00000000	0000000	00000000	0000000	00000000	0000000	00000000	0000000
$Y26$	7	0000001n	1nnnnnnn	0001nnnn	001nnnnn	01nnnnnn	001nnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn
	6	0000001	01nnnnnn	0001nnnn	001nnnnn	01nnnnnn	001nnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn
	5	0000000	001nnnnn	0000001	0001nnnn	01nnnnnn	001nnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn
	4	0000000	0001nnnn	0000000	0001nnnn	01nnnnnn	001nnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn
	3	0000000	0001nnnn	0000000	0001nnnn	01nnnnnn	001nnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	1nnnnnnn
	2	0000000	0000001n	0000000	0000001n	00000001	000001nn	00000001	0000001n	00000001	0000001n	00000001	0000001n	00000001	0000001n	00000001	0000001n
	1	0000000	0000001	0000000	0000001	00000000	000000nn	00000000	0000001	00000000	0000001	00000000	0000001	00000000	0000001	00000000	0000001
	0	0000000	0000000	0000000	0000000	00000000	000000nn	00000000	0000000	00000000	0000000	00000000	0000000	00000000	0000000	00000000	0000000
$Y27$	7	01nnnnnn	1nnnnnnn	01nnnnnn	001nnnnn	01nnnnnn	0001nnnn	01nnnnnn	001nnnnn	01nnnnnn	001nnnnn	01nnnnnn	001nnnnn	01nnnnnn	1nnnnnnn	01nnnnnn	0001nnnn
	6	001nnnnn	01nnnnnn	001nnnnn	0001nnnn	001nnnnn	0001nnnn	001nnnnn	0001nnnn	001nnnnn	0001nnnn	001nnnnn	0001nnnn	001nnnnn	01nnnnnn	001nnnnn	0000001n
	5	0001nnnn	001nnnnn	0001nnnn	0000001n	0001nnnn	0000001n	0001nnnn	0000001n	0001nnnn	0000001n	0001nnnn	0000001n	0001nnnn	001nnnnn	0001nnnn	0000001n
	4	0001nnnn	0001nnnn	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0001nnnn	0000001n	0000000n
	3	0000001n	0000001n	0000001n	0000000n	0000001n	0000000n	0000001n	0000000n	0000001n	0000000n	0000001n	0000000n	0000001n	0000001n	0000000n	0000000n
	2	0000001	0000001n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n	0000000n
	1	0000000	0000001	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$Y28$	7	1nnnnnnn	1nnnnnnn	001nnnnn	1nnnnnnn	001nnnnn	0001nnnn	001nnnnn	1nnnnnnn	001nnnnn	1nnnnnnn	001nnnnn	1nnnnnnn	001nnnnn	1nnnnnnn	001nnnnn	0001nnnn
	6	01nnnnnn	01nnnnnn	0001nnnn	01nnnnnn	0001nnnn	0000001n	0001nnnn	01nnnnnn	0001nnnn	01nnnnnn	0001nnnn	01nnnnnn	0001nnnn	01nnnnnn	0001nnnn	0000001n
	5	001nnnnn	001nnnnn	0001nnnn	001nnnnn	0000001n	0000001n	0000001n	001nnnnn	0000001n	001nnnnn	0000001n	001nnnnn	0000001n	001nnnnn	0000001n	0000001n
	4	0001nnnn	0001nnnn	0000001n	0001nnnn	0000001n	0000001n	0000001n	0001nnnn	0000001n	0001nnnn	0000001n	0001nnnn	0000001n	0001nnnn	0000001n	0000001n
	3	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000001n	0000000n
	2	0000001	0000001	0000000n	0000001	0000000n	0000000n	0000000n	0000001	0000000n	0000001	0000000n	0000001	0000000n	0000001	0000000n	0000000n
	1	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$Y29$	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$Y30$	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$Y31$	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000
$Y32$	0	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000	0000000

Այժմ դիտարկենք ռաունդի այն մասը, որը պարունակում է ոչ գծային ֆունկցիաները (NL) և նրանց համար գտնենք ոչ զրոյական չբալանսավորվածությամբ հոմոմորֆ I/O գումարները: Մուտքային ֆունկցիան հոմոմորֆ է XOR/ADD համար, իսկ ելքային ֆունկցիան ADD/XOR համար: Այսպիսով I/O գումարները կարելի է ստանալ գումարելով նրանց EXP և LOG բլոկների I/O գումարները:

Այսպիսով մնում է կապակցել հաջորդական կիսառաունդների I/O գումարները:

Թեորեմ 3.4: Էֆեկտիվ հոմոմորֆ I/O գումարներ գտնելու մեթոդի միջոցով հնարավոր չի գտնել ոչ զրոյական չբալանսավորվածությամբ I/O գումար SAFER-256 համակարգի կիսառունդների հաջորդականության համար, որը պարունակում է երկու PHT շերտ:

Ապացույց: Դիցուք $T_{a,b}^{\mathrm{PHT-hr}}$, $T_{b,c}^{\mathrm{NL}}$ և $T_{c,d}^{\mathrm{PHT-hr}}$ եռակի գումարները կիսառաունդների հոմոմորֆ կապակցված եռակի գումարներ են՝ մաքսիմալացնող առանցքային ֆունկցիաներով: Եթե $T_{a,b}^{\mathrm{PHT-hr}}$ և $T_{c,d}^{\mathrm{PHT-hr}}$ եռակի գումարների չբալանսավորվածությունը զրոյից տարբեր է, 256 բիթ ոչ զրոյական $a,b,c,d,$ դիմակների միայն վերջին երկու բիթերը կարող են լինել 1 (տե՛ս Լեմմա 3.3): Հետևաբար $I\left(T_{b,c}^{\mathrm{NL}}\right)=0,$ քանի որ I/O գումարի բանալուց կախված միջին չբալանսավորվածությունը նույնպես 0 է (տե՛ս Դիտողություն 3.2): Այսպիսով՝ այս երեք կիսառաունդների եռակի գումարների գումարի չբալանսավորվածությունը 0 է:

Նկատենք որ գոյություն ունեն NL, PHT, NL հաջորդականության ոչ զրոյական չբալանսավորվածությամբ հոմոմորֆ I/O գումարներ: Ամենաէֆեկտիվ I/O գումարներից մեկը $S_{a,b}^{\mathrm{NL-PHT-NL}}$ գումարն է, որտեղ $a2,a10$ և $b13,b14,b31,b32$ բայթերը կամ cd են կամ ff, իսկ մնացած բայթերը 0 են: Նրանց չբալանսավորվածությունը $\left(\frac{28}{128}\right)^6$ է, քանի որ $I\left(S_{ff,01}^{\mathrm{EXP}}\right)=I\left(S_{cd,01}^{\mathrm{EXP}}\right)=I\left(S_{01,ff}^{\mathrm{LOG}}\right)=I\left(S_{01,cd}^{\mathrm{LOG}}\right)=\frac{28}{128}$, իսկ

$$I\left(S_{a,b}^{\mathrm{PHT}}\right)=I\left(S_{a1\oplus a2\oplus a3\oplus a4,b1\oplus b2\oplus b3\oplus b4}^{\mathrm{PHT}}\right)=\ I\left(S_{a1,b1}^{\mathrm{PHT}}\right)\cdot I\left(S_{a2,b2}^{\mathrm{PHT}}\right)=I\left(S_{a3,b3}^{\mathrm{PHT}}\right)=I\left(S_{a4,b4}^{\mathrm{PHT}}\right)=1,$$

եթե

$a1\oplus a2\oplus a3\oplus a4=$

(01 00 00 01 00 00 00 01 00 00 00 00 00 00 00 00 00 00 01 01 00 00 01 00 00 00 00

01 00 00 01)⊕

(01 01 00 01 00 01 01 01 00 00 00 00 00 01 01 00 00 00 00 01 01 00 00 01 00 00 00 00

01 00 00 01) ⊕

(00 00 00 00 00 00 00 00 01 00 00 01 00 00 00 01 00 00 00 00 01 00 00 01 00 00 00 01

01 00 00 01) ⊕

(00 00 00 00 00 01 01 00 01 01 00 01 00 01 01 01 00 00 00 00 01 00 00 01 00 00 00 01

01 00 00 01) =

(00 01 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00) = a,

$b1 \oplus b2 \oplus b3 \oplus b4 =$

(00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00)⊕

(00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00)⊕

(00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 01 00) ⊕

(00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 01

(00 00 00 00 00 00 00 00 00 00 00 00 00 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 01 01) = b:

Այսպիսով՝ SAFER-256 համակարգը անվտանգ է գծային վերլուծության նկատմամբ 3 ռաունդի դեպքում:

Եզրակացություն

Ատենախոսությունը նվիրված է սիմետրիկ ծածկագրաբանության անվտանգ բլոկային ծածկագրական համակարգերի SAFER ընտանիքի 128 բիթային և 256 բիթային արագ և արդյունավետ իրագործման ծածկագրման ալգորիթմների մշակմանը:

Ստորև բերված են աշխատանքում ստացված արդյունքների համառոտ նկարագրությունները:

1. SAFER ընտանիքի 128 բիթ երկարության SAFER+ բլոկային ծածկագրական համակարգի հիման վրա մշակվել է ավելի մեծ արագագործությամբ 128 բիթային ծածկագրական համակարգ (Ձևափոխած SAFER+): Մշակվել է SAFER+ համակարգի այնպիսի ձևափոխություններ, որոնք մեծացրել են համակարգի արագագործությունը ARM պլատֆորմի 32-բիթ պրոցեսորի վրա 1,7 անգամ՝ չդարձնելով համակարգի կրիպտոկայունությունը խոցելի դիֆերենցիալ և գծային վերլուծությունների նկատմամբ Դիֆերենցիալ և գծային վերլուծությունների նկատմամբ ձևափոխած SAFER+ համակարգի կայունությունը ուսումնասիրվել և հիմնավորվել է ատենախոսության շրջանակներում ստեղծված ծրագրային փաթեթների միջոցով (տե՛ս Հավելված 3) [10],[15]:

2. SAFER ընտանիքի բլոկային ծածկագրական համակարգերի, մասնավորապես ատենախոսության շրջանակում մշակված ձևափոխած SAFER+ համակարգի հիման վրա մշակվել է 256 բիթ բլոկի և բանալու երկարության նոր բլոկային ծածկագրական համակարգ՝ հաշվի առնելով դիֆերենցիալ վերլուծության նկատմամբ ձևափոխած SAFER+ համակարգի կայունության բարձր մակարդակը: Նոր համակարգը անվանել ենք SAFER-256: Համակարգը կայուն է դիֆերնցիալ վերլուծության նկատմամբ նվազագույնը 5 ռաունդի դեպքում (SAFER+ ալգորիթմի հիման վրա մշակված 256-բիթային համակարգերը դիֆերենցիալ վերլուծության նկատմամբ կրպտոկայուն են նվազագույնը 6 ռաունդի դեպքում), իսկ գծային վերլուծության նկատմամբ 3 ռաունդի դեպքում (Դիֆերենցիալ և գծային վերլուծությունների նկատմամբ SAFER-256 համակարգի

կայունությունը ուսումնասիրելու և հիմնավորելու համար ատենախոսության շրջանակներում ստեղծվել են ծրագրային փաթեթներ (տե՛ս Հավելված 2)): SAFER-256 ապահովում է 256 բիթ անվտանգություն բախում հարձակման նկատմամբ և 256 բիթ երկարության բլոկը 1,5 անգամ ավելի արագ է ծածկագրում, քան SAFER+ համակարգը [11],[14]:

3. SAFER ընտանիքի SAFER+, ձևափոխած SAFER+ և SAFER-256 բլոկային ծածկագրական համակարգերից յուրաքանչյուրը դիֆերենցիալ վերլուծության նկատմամբ կայուն են նվազագույնը 5 ռաունդի դեպքում (Հետազոտությունները կատարվել են դիֆերենցիալ վերլուծության միջոցով (տե՛ս Հավելված 2 և 3)): Այն բայթային տեղադրությունները, որոնց դեպքում համակարգերը կայուն են դիֆերենցիալ վերլուծության նկատմամբ նվազագույնը 5 ռաունդի դեպքում անվանել ենք "*Armenian Shuffle*" և կասենք, որ նրանք ապահովում են օպտիմալ դիֆուզիա: Իսկ ծածկագրական համակարգերը, որոնք իրարից տարբերվում են միայն բայթային տեղադրությամբ և ունեն համարժեք կրիպտոկայունություն ու նույն արագագործությունը կհամարենք համարժեք բլոկային ծածկագրական համակարգեր [12],[13]:

Ատենախոսության շրջանակներում ստացված արդյունքները ունեն և՛ տեսական, և՛ կիրառական նշանակություն:

Օգտագործված գրականության ցանկը

1. Biham E., Shamir A.: Differential cryptanalysis of DES-like cryptosystem. Advances in Cryptology-CRYPTO'90 (Ed. A. J. Menezes and S. A. Vanstone), Lecture Notes in Computer Science, Heidelberg and New York, Springer, No. 537, pp. 212-241, 1990.
2. Diffie W., Hellman M.: New directions in cryptography". IEEE Transactions on Information Theory. 22 (6): 644–654, 1976.
3. ElGamal T.: A public key cryptosystem and a signature scheme based on discrete logarithms. Advances in cryptology: Proceedings of CRYPTO 84. Lecture Notes in Computer Science. Santa Barbara, California, United States: Springer-Verlag. pp. 10–18, 1985.
4. Harpes C.: Cryptanalysis of Iterated Block Ciphers. ETH Series in Inform. Processing Konstanz: Hartung-Gorre Verlag, 1996.
5. Harpes C., Kramer G. C. and Massey J. L.: A Generalization of Linear Cryptanalysis and the Applicability of Matsiu's Pilling-up Lemma. Advances in Cryptology-CRYPTO'95 (Ed. L.C. Guilou and J. J. Quisquater), Lecture Notes in Computer Science, Heidelberg and New York, Springer, No. 921, 1995.
6. Harpes C.: A Generalization of Linear Cryptanalysis Applied to SAFER. Tech. Rpt. Signal and Information Proc. Lav. Swiss Federal Institute of Technology, Zurich, March 1995.
7. Kelsey J., Scheier B., Wagner D.: Key Schedule Weaknesses in SAFER+. The Second Encryption Standard Candidate Conference, Rome, ITALY, March 22-23, pp. 155-167, 1999.
8. Knudsen L. R.: A Key-Schedule Weakness in SAFER K-64. Advances in Cryptology-CRYPTO'95 (Ed. D. Coppersmith), Lecture Notes in Computer Science, Heidelberg and New York, Springer, No. 963, 1995.
9. Knudsen L. R., Berson T. A.: Truncated Differentials of SAFER. Fast Software Encryption (Ed. D. Gollmann), Lecture Notes in Computer Science, New York, Springer, No. 1039, pp. 15-26, 1996.
10. Kyuregyan K.: Some modifications of SAFER+. In Reports of NAS RA, v. 115, No 1, pp. 33-39, Yerevan, Armenia, 2015.
11. Khachatrian G. H., Kyureghyan M. K., Kyuregyan K. M.: Design and Cryptanalysis of a New Encryption Algorithm SAFER-256. Transactions of IIAP NAS RA, Mathematical Problems of Computer Science Vol. 42, pp. 97-106, 2014.

12. Kyuregyan K. M.: On Optimality of Regular SAFER+ and Modified SAFER+ Diffusion. Transactions of IIAP NAS RA, Mathematical Problems of Computer Science Vol. 44, pp. 109-115, 2015.
13. Kyuregyan K. M.: On Optimality of SAFER-256 Diffusion. Transactions of IIAP NAS RA, Mathematical Problems of Computer Science Vol. 44, pp. 133-137, 2015.
14. Kyureghyan M. K., Kyuregyan K. M.: Linear Cryptanalysis of SAFER-256. Transactions of IIAP NAS RA, Mathematical Problems of Computer Science Vol. 45, pp. 111-121, 2016.
15. Kyuregyan K. M.: Linear Cryptanalysis of 128-bit Modified SAFER+. Transactions of IIAP NAS RA, Mathematical Problems of Computer Science Vol. 45, pp. 127-137, 2016.
16. Lai X., Massey J. L., Murphy S.: Markov Ciphers and Differential Cryptanalysis. Advances in Cryptology-CRYPTO'91 (Ed. D. W. Davies), Lecture Notes in Computer Science, Heidelberg and New York, Springer, No. 547, pp. 17-38, 1991.
17. Lai X., Massey J. L.: Hash Function Based on Block Ciphers. Advances in Cryptology-EUROCRYPT'92 (Ed. R. A. Rueppel), Lecture Notes in Computer Science, Heidelberg and New York, Springer, No. 658, pp. 55-70, 1993.
18. Matsui M.: Linear cryptanalysis Method for DES Cipher. Advances in Cryptology-EUROCRYPT'93 (Ed. T. Helleseth), Lecture Notes in Computer Science New York, Springer, No. 765, 1994.
19. Massey J. L.: SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm. Fast Software Encryption (Ed. R. Anderson), Lecture Notes in Computer Science, New York, Springer, No. 809, pp. 1-17, 1994.
20. Massey J. L.: SAFER K-64: One Year Later. Fast Software Encryption II (Ed. B. Preneel) Lecture Notes in Computer Science, New York, Springer, No. 1008, pp. 212-241, 1995.
21. Massey J. L., Khachatrian G., Kyuregian M.: Nomination of SAFER+ as candidate algorithm for the advanced encryption standard (AES). NIST AES Proposal, 1998.
22. Massey J. L.: On the Optimality of SAFER+ diffusion. The Second Encryption Standard Candidate Conference, Rome, ITALY, March 22-23, pp. 215-229, 1999.
23. Massey J. L., Khachatrian G., Kyuregian M.: Nomination of SAFER++ as Candidate Algorithm for the New European Schemes for Signatures, Integrity and Encryption (NESSIE). Submission document from Cylink Corporation, 2000.
24. Menezes A., Oorschot P. van, Vanstone S.: Handbook of Applied Cryptography, by CRC Press, 1996.
http://cacr.uwaterloo.ca/hac/

25. Murphy S.: An analysis of SAFER, J. Cryptology, Vol. 11 No. 4, pp. 235-251. Autumn 1998.
26. Nakahara J., Prenel B., Vandewalle J.: Linear Cryptanalysis of Reduced-Round Versions of the SAFER Block Cipher Family. Fast Software Encryption Workshop 2000, New York, 10-12 April 2000.
27. Perkins S.R.: Linear cryptanalysis of the SAFER K-64 Block Cipher. Diploma Thesis, Signal & Info. Proc. Lab., Swiss Fed. Inst. of Tech., Zurich, 15 July 1994.
28. Rivest R.L., Shamir A., Adleman L.: "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM, Vol. 21, No. 2, February 1978, p. 120-26.
29. Shannon C. E.: Communication Theory of Secrecy Systems. Bell System Tech. J., Vol. 28, pp.656-715, Oct., 1949.
30. Shibutani K., Bogdanov A.: Towards the optimality of Feistel ciphers with substitution-permutation functions. Des. Codes Cryptography 73(2), 667–682 (2014). http://dx.doi.org/10.1007/s10623-014-9970-4
31. US Department of Commerce/National Bureau of Standards. FIPS Pub 46, Data Encryption Standard Candidate Conference, March 22-23, 1999, Rome, ITALY.
32. Vaudenay S.: One the Need for Multipermutations Cryptanalysis of MD4 and SAFER. Fast Software Encryption II, Lecture Notes in Computer Science, New York, Springer, No. 1008, pp. 286-297, 1995.
33. Microsoft: Distributed Security/Cryptography for Network and Information Security/Basic Components of Modern Cryptography. https://technet.microsoft.com/en-us/library/cc962028.aspx
34. BLUETOOTH SPECIFICATION Version 1.0B, 28 Nov. 1999. http://grouper.ieee.org/groups/802/15/Bluetooth/core_10_b.pdf
35. Migrating a software application from ARMv5 to ARMv7-A/R. Application Note 425. 2014. http://arm.com (ARM infocenter).
36. Feng K., Xu. C., Wang W., Yang Z., Tian Z.: An Optimized Matrix Multiplication on ARMv7 Architecture. Journal of Convergence Information Technology (JCIT), Volume7, Number10, pp. 41-48, June 2012.
37. Sharmila D., Neelaveni R.: A Proposed SAFER Plus Security algorithm using Fast Walsh Hadamard transform for Bluetooth Technology. International Journal of Wireless & Mobile Networks (IJWMN), Vol 1, No 2, pp. 80-88, November 2009.

38. Chaudhari P., Diwanji H.: Enhanced SAFER+ Algorithm for Bluetooth to Withstand Against Key Pairing Attack. Advances in Computing and Information Technology. AISC 176, pp. 651–660, Springer-Verlag Berlin Heidelberg 2012.
39. Zhao J. Wang M., Chen J., Zheng Y.: New Impossible Differential Attack on SAFER Block Cipher Family. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences Vol. E98.A (2015) No. 3 pp. 843-852.
40. Zhao J., Wang M., Chen J., Zheng Y.: New Impossible Differential Attack on SAFER+ and SAFER++. LECTURE NOTES IN COMPUTER SCIENCE, 7839; 170-183; Information security and cryptology von Springer, Heidelberg; 2013.
41. Hu Y., Zhang Y., Xiao. G.: Integral Cryptanalysis of SAFER+. Electronic Letters, 35(17):1458–1459, Aug 1999.
42. Bogdanov A. and Rijmen V.: Linear hulls with correlation zero and linear cryptanalysis of block ciphers. Designs, Codes and Cryptography, vol. 70, no. 3, pp. 369-383, March (2014).
43. Choy J., Yap H.: Impossible Boomerang Attack for Block Cipher Structures. In Tsuyoshi Takagi and Masahiro Mambo, editors, IWSEC'09, volume 5824 of LNCS, pages 22–37. Springer-Verlag, 2009.
44. Khachatrian G., Karapetyan M.: White-box Encryption Algorithm Based on SAFER+. In Proceedings of International Workshop on Information Theory and Data Science, "Forum Information Age to Big Data Era", Yerevan, Armenia, October 3-5, pp. 77-88, 2016.
45. Chow S., Eisen P., Johnson H., P.C. van Oorschot: White-Box Cryptography and an AES Implementation. In "Selected Areas in Cryptography: 9th Annual International Workshop, SAC 2002", Lecture Notes in Computer Science 2595 (2003), 250–270.

SAFER-256 համակարգի ծրագրային կոդը (C++ լեզվով)

```
#include "stdafx.h"
#include <iostream>
#include <cstring>
#include <memory.h>
#include <ctime>
#include <fstream>

using namespace std;

#define FALSE 0
#define TRUE 1
#define ROUND_COUNT 6
#define BLOCK_SIZE 32
#define KEY_SIZE 32
#define KEY_COUNT 2 * ROUND_COUNT + 1
#define VALUES_COUNT 256

#define Sub(v1,v2,start_from_xor) \
{ \
if( start_from_xor ) { \
    v1[0] ^= v2[0]; \
    v1[1] ^= v2[1]; \
    v1[2] ^= v2[2]; \
    v1[3] ^= v2[3]; \
    v1[4] -= v2[4]; \
    v1[5] -= v2[5]; \
    v1[6] -= v2[6]; \
    v1[7] -= v2[7]; \
    v1[8] ^= v2[8]; \
    v1[9] ^= v2[9]; \
    v1[10] ^= v2[10]; \
    v1[11] ^= v2[11]; \
    v1[12] -= v2[12]; \
    v1[13] -= v2[13]; \
    v1[14] -= v2[14]; \
    v1[15] -= v2[15]; \
      v1[16] ^= v2[16]; \
    v1[17] ^= v2[17]; \
    v1[18] ^= v2[18]; \
    v1[19] ^= v2[19]; \
    v1[20] -= v2[20]; \
    v1[21] -= v2[21]; \
    v1[22] -= v2[22]; \
    v1[23] -= v2[23]; \
    v1[24] ^= v2[24]; \
    v1[25] ^= v2[25]; \
    v1[26] ^= v2[26]; \
    v1[27] ^= v2[27]; \
    v1[28] -= v2[28]; \
    v1[29] -= v2[29]; \
    v1[30] -= v2[30]; \
    v1[31] -= v2[31]; \
} else { \
    v1[0] -= v2[0]; \
    v1[1] -= v2[1]; \
    v1[2] -= v2[2]; \
    v1[3] -= v2[3]; \
    v1[4] ^= v2[4]; \
    v1[5] ^= v2[5]; \
    v1[6] ^= v2[6]; \
    v1[7] ^= v2[7]; \
    v1[8] -= v2[8]; \
    v1[9] -= v2[9]; \
    v1[10] -= v2[10]; \
    v1[11] -= v2[11]; \
    v1[12] ^= v2[12]; \
    v1[13] ^= v2[13]; \
    v1[14] ^= v2[14]; \
    v1[15] ^= v2[15]; \
      v1[16] -= v2[16]; \
```

```
    v1[17] -= v2[17]; \
    v1[18] -= v2[18]; \
    v1[19] -= v2[19]; \
    v1[20] ^= v2[20]; \
    v1[21] ^= v2[21]; \
    v1[22] ^= v2[22]; \
    v1[23] ^= v2[23]; \
    v1[24] -= v2[24]; \
    v1[25] -= v2[25]; \
    v1[26] -= v2[26]; \
    v1[27] -= v2[27]; \
    v1[28] ^= v2[28]; \
    v1[29] ^= v2[29]; \
    v1[30] ^= v2[30]; \
    v1[31] ^= v2[31]; \
} \
}

#define Sum(v1,v2,start_from_xor) \
{ \
if( start_from_xor ) { \
    v1[0] ^= v2[0]; \
    v1[1] ^= v2[1]; \
    v1[2] ^= v2[2]; \
    v1[3] ^= v2[3]; \
    v1[4] += v2[4]; \
    v1[5] += v2[5]; \
    v1[6] += v2[6]; \
    v1[7] += v2[7]; \
    v1[8] ^= v2[8]; \
    v1[9] ^= v2[9]; \
    v1[10] ^= v2[10]; \
    v1[11] ^= v2[11]; \
    v1[12] += v2[12]; \
    v1[13] += v2[13]; \
    v1[14] += v2[14]; \
    v1[15] += v2[15]; \
      v1[16] ^= v2[16]; \
    v1[17] ^= v2[17]; \
    v1[18] ^= v2[18]; \
    v1[19] ^= v2[19]; \
    v1[20] += v2[20]; \
    v1[21] += v2[21]; \
    v1[22] += v2[22]; \
    v1[23] += v2[23]; \
    v1[24] ^= v2[24]; \
    v1[25] ^= v2[25]; \
    v1[26] ^= v2[26]; \
    v1[27] ^= v2[27]; \
    v1[28] += v2[28]; \
    v1[29] += v2[29]; \
    v1[30] += v2[30]; \
    v1[31] += v2[31]; \
} else { \
    v1[0] += v2[0]; \
    v1[1] += v2[1]; \
    v1[2] += v2[2]; \
    v1[3] += v2[3]; \
    v1[4] ^= v2[4]; \
    v1[5] ^= v2[5]; \
    v1[6] ^= v2[6]; \
    v1[7] ^= v2[7]; \
    v1[8] += v2[8]; \
    v1[9] += v2[9]; \
    v1[10] += v2[10]; \
    v1[11] += v2[11]; \
    v1[12] ^= v2[12]; \
    v1[13] ^= v2[13]; \
    v1[14] ^= v2[14]; \
    v1[15] ^= v2[15]; \
      v1[16] += v2[16]; \
    v1[17] += v2[17]; \
    v1[18] += v2[18]; \
    v1[19] += v2[19]; \
    v1[20] ^= v2[20]; \
    v1[21] ^= v2[21]; \
```

```
    v1[22] ^= v2[22]; \
    v1[23] ^= v2[23]; \
    v1[24] += v2[24]; \
    v1[25] += v2[25]; \
    v1[26] += v2[26]; \
    v1[27] += v2[27]; \
    v1[28] ^= v2[28]; \
    v1[29] ^= v2[29]; \
    v1[30] ^= v2[30]; \
    v1[31] ^= v2[31]; \
} \
}

#define ExpLog(exponent,logarithm,v,start_from_exp) \
{ \
    if( start_from_exp ) { \
        v[0] = exponent[v[0]]; \
        v[1] = exponent[v[1]]; \
        v[2] = exponent[v[2]]; \
        v[3] = exponent[v[3]]; \
        v[4] = logarithm[v[4]]; \
        v[5] = logarithm[v[5]]; \
        v[6] = logarithm[v[6]]; \
        v[7] = logarithm[v[7]]; \
        v[8] = exponent[v[8]]; \
        v[9] = exponent[v[9]]; \
        v[10] = exponent[v[10]]; \
        v[11] = exponent[v[11]]; \
        v[12] = logarithm[v[12]]; \
        v[13] = logarithm[v[13]]; \
        v[14] = logarithm[v[14]]; \
        v[15] = logarithm[v[15]]; \
             v[16] = exponent[v[16]]; \
        v[17] = exponent[v[17]]; \
        v[18] = exponent[v[18]]; \
        v[19] = exponent[v[19]]; \
        v[20] = logarithm[v[20]]; \
        v[21] = logarithm[v[21]]; \
        v[22] = logarithm[v[22]]; \
        v[23] = logarithm[v[23]]; \
        v[24] = exponent[v[24]]; \
        v[25] = exponent[v[25]]; \
        v[26] = exponent[v[26]]; \
        v[27] = exponent[v[27]]; \
        v[28] = logarithm[v[28]]; \
        v[29] = logarithm[v[29]]; \
        v[30] = logarithm[v[30]]; \
        v[31] = logarithm[v[31]]; \
    } else { \
        v[0] = logarithm[v[0]]; \
        v[1] = logarithm[v[1]]; \
        v[2] = logarithm[v[2]]; \
        v[3] = logarithm[v[3]]; \
        v[4] = exponent[v[4]]; \
        v[5] = exponent[v[5]]; \
        v[6] = exponent[v[6]]; \
        v[7] = exponent[v[7]]; \
        v[8] = logarithm[v[8]]; \
        v[9] = logarithm[v[9]]; \
        v[10] = logarithm[v[10]]; \
        v[11] = logarithm[v[11]]; \
        v[12] = exponent[v[12]]; \
        v[13] = exponent[v[13]]; \
        v[14] = exponent[v[14]]; \
        v[15] = exponent[v[15]]; \
             v[16] = logarithm[v[16]]; \
        v[17] = logarithm[v[17]]; \
        v[18] = logarithm[v[18]]; \
        v[19] = logarithm[v[19]]; \
        v[20] = exponent[v[20]]; \
        v[21] = exponent[v[21]]; \
        v[22] = exponent[v[22]]; \
        v[23] = exponent[v[23]]; \
        v[24] = logarithm[v[24]]; \
        v[25] = logarithm[v[25]]; \
        v[26] = logarithm[v[26]]; \
```

```
        v[27] = logarithm[v[27]]; \
        v[28] = exponent[v[28]]; \
        v[29] = exponent[v[29]]; \
        v[30] = exponent[v[30]]; \
        v[31] = exponent[v[31]]; \
} \
}

static const unsigned char bias[384] =
{70, 151, 177, 186, 163, 183, 16, 10, 197, 55, 179, 201, 90, 40, 172, 100, 165, 236, 171, 170, 198, 103, 149,
88, 13, 248, 154, 246, 110, 102, 220, 5,
236, 171, 170, 198, 103, 149, 88, 13, 248, 154, 246, 110, 102, 220, 5, 61, 211, 138, 195, 216, 137, 106, 233,
54, 73, 67, 191, 235, 212, 150, 155, 104,
138, 195, 216, 137, 106, 233, 54, 73, 67, 191, 235, 212, 150, 155, 104, 160, 101, 93, 87, 146, 31, 213, 113,
92, 187, 34, 193, 190, 123, 188, 153, 99,
93, 87, 146, 31, 213, 113, 92, 187, 34, 193, 190, 123, 188, 153, 99, 148, 95, 42, 97, 184, 52, 50, 25, 253,
251, 23, 64, 230, 81, 29, 65, 68,
42, 97, 184, 52, 50, 25, 253, 251, 23, 64, 230, 81, 29, 65, 68, 143, 41, 221, 4,  128, 222, 231, 49, 214,
127, 1, 162, 247, 57, 218, 111, 35,
221, 4, 128, 222, 231, 49, 214, 127, 1, 162, 247, 57, 218, 111, 35, 202, 254, 58, 208, 28, 209, 48, 62, 18,
161, 205, 15, 224, 168, 175, 130, 89,
58, 208, 28, 209, 48, 62, 18, 161, 205, 15, 224, 168, 175, 130, 89, 44, 245, 125, 173, 178, 239, 194, 135,
206, 117, 6, 19, 2, 144, 79, 46, 114,
125, 173, 178, 239, 194, 135, 206, 117, 6, 19, 2, 144, 79, 46, 114, 51, 133, 192, 141, 207, 169, 129, 226,
196, 39, 47, 108, 122, 159, 82, 225, 21,
192, 141, 207, 169, 129, 226, 196, 39, 47, 108, 122, 159, 82, 225, 21, 56, 43, 252, 32, 66, 199, 8, 228, 9,
85, 94, 140, 20, 118, 96, 255, 223,
252, 32, 66, 199, 8, 228, 9, 85, 94, 140, 20, 118, 96, 255, 223, 215, 152, 250, 11, 33, 0, 26, 249, 166, 185,
232, 158, 98, 76, 217, 145, 80,
250, 11, 33, 0, 26, 249, 166, 185, 232, 158, 98, 76, 217, 145, 80, 210, 238, 24, 180, 7, 132, 234, 91, 164,
200, 14, 203, 72, 105, 75, 78, 156,
24, 180, 7, 132, 234, 91, 164, 200, 14, 203, 72, 105, 75, 78, 156, 53, 121, 69, 77, 84, 229, 37, 60, 12, 74,
139, 63, 204, 167, 219, 107, 174};

unsigned char shuffle[BLOCK_SIZE] = {25, 28, 29, 32, 17, 20, 21, 24, 13, 16, 9, 12, 5, 8, 1, 4, 3, 2, 7, 6,
11, 10, 15, 14, 27, 26, 31, 30, 19, 18, 23, 22};

unsigned char key[KEY_COUNT][BLOCK_SIZE];
unsigned char exponent[VALUES_COUNT];
unsigned char logarithm[VALUES_COUNT];

void GenerateKeySchedule(unsigned char key[KEY_COUNT][BLOCK_SIZE], unsigned char* Key);
void LinearTransformDecrypt( unsigned char* input, const unsigned char* shuffle);
void LinearTransformEncrypt( unsigned char* input, const unsigned char* shuffle);
void InitializeExpLog( unsigned char  exponent[VALUES_COUNT], unsigned char logarithm[VALUES_COUNT]);
void DecryptBlock( unsigned char input[BLOCK_SIZE]);
void EncryptBlock( unsigned char input[BLOCK_SIZE]);

int main()
{
      int i, j;

unsigned char Key[KEY_SIZE] = {75, 199, 59, 17, 253, 138, 14, 173, 241, 147, 48, 71, 201, 91, 65, 171, 161,
108, 193, 20, 110, 7, 168, 207, 238, 56, 97, 81, 199, 41, 162, 235};

unsigned char input[BLOCK_SIZE] = {188, 214, 84, 60, 8, 113, 124, 14, 252, 207, 88, 192, 187, 17, 28, 161,
254, 223, 102, 41, 167, 154, 202, 168, 186, 100, 61, 44, 171, 199, 55, 157};

      InitializeExpLog(exponent,logarithm);
      GenerateKeySchedule(key, Key);

      cout << endl << endl << "plaintext:" << "\t";

      for(i = 0; i < BLOCK_SIZE; i++)
      cout << (int)input[i] << "\t";

      cout << endl << endl << "ciphertext:" << "\t";

      EncryptBlock(input);

      for(i = 0; i < BLOCK_SIZE; i++)
      cout << (int)input[i] << "\t";

      cout << endl << endl << "Recoverd plaintext:  ";

      DecryptBlock(input);
```

```
      for(i = 0; i < BLOCK_SIZE; i++)
            cout << (int)input[i] << "\t";

/*
      clock_t start = clock();

      for(j = 0; j < 1000000; j++)
            EncryptBlock(input);

      clock_t end = clock();
      clock_t duration = end - start;
      double timeInSeconds = duration / (double) CLOCKS_PER_SEC;

      cout << endl << "Time:   " << timeInSeconds;
*/
      cin >> i;
      return 0;
}

void GenerateKeySchedule(unsigned char key[KEY_COUNT][BLOCK_SIZE], unsigned char* Key)
{
    int i, j;
    int threeBits = 224;
    unsigned char parity_byte = 0;
    unsigned char KeyRegister[KEY_SIZE+1];

    for(i = 0; i < KEY_SIZE; i++)
    {
        KeyRegister[i] = Key[i];
        parity_byte ^= Key[i];
    }

      KeyRegister[KEY_SIZE] = parity_byte;
      cout << "   Subkeys: " << endl << endl;

      for(i = 0; i < BLOCK_SIZE; i++)
      {
            key[0][i] = Key[i];
            cout << (int)key[0][i] << "   ";
      }
      cout << endl << endl;

      for(i = 1; i < KEY_COUNT; i++)
      {
       for(j = 0; j < KEY_SIZE + 1; j++)
       {
            threeBits = 224;
            threeBits = threeBits & KeyRegister[j];
            threeBits = threeBits >> 5;
            KeyRegister[j] = KeyRegister[j] << 3;
            KeyRegister[j] = KeyRegister[j] | threeBits;
       }

       for( j = 0; j < KEY_SIZE; j++ )
           swap(KeyRegister[j], KeyRegister[j + 1]);

       for( j = 0; j < BLOCK_SIZE; j++)
       {
           key[i][j] = KeyRegister[j] + bias[(i - 1) * BLOCK_SIZE + j];
                  cout << (int)key[i][j] << "   ";
       }
            cout << endl << endl;
    }
}

void InitializeExpLog( unsigned char  exponent[VALUES_COUNT], unsigned char logarithm[VALUES_COUNT])
{
    int i, iNum = 45;

    exponent[0] = 1;
    for(i = 1; i < VALUES_COUNT;  i++)
         exponent[i] = (iNum * exponent[i-1]) % 257;

      exponent[128] = 0;
        exponent[129] = (iNum * VALUES_COUNT) % 257;
```

```
        for(i = 130; i < VALUES_COUNT; i++)
           exponent[i] = (iNum * exponent[i-1]) % 257;

        for(i = 0; i < VALUES_COUNT; i++)
          logarithm[exponent[i]] = i;
}

void DecryptBlock( unsigned char input[BLOCK_SIZE])
{
   int round;

   int  i = KEY_COUNT - 1;
   Sub(input, key[KEY_COUNT - 1], TRUE);

    for(round = 1; round <= ROUND_COUNT; round++)
    {
       LinearTransformDecrypt( input, shuffle);
      Sub(input, key[KEY_COUNT - 2 * round], FALSE);
       ExpLog(exponent,logarithm, input, FALSE);
       Sub(input, key[KEY_COUNT - 2 * round - 1], TRUE);
    }
}

void EncryptBlock(unsigned char input[BLOCK_SIZE])
{
    int round;

    for(round = 1; round <= ROUND_COUNT; round++)
    {
       Sum(input, key[ 2*round-2], TRUE);
       ExpLog(exponent,logarithm, input, TRUE);
       Sum(input, key[2* round-1], FALSE);
       LinearTransformEncrypt(input, shuffle);
    }

Sum(input, key[KEY_COUNT - 1], TRUE);
}

void LinearTransformEncrypt( unsigned char* input, const unsigned char* shuffle)
{
    int i, j;
    unsigned char output[BLOCK_SIZE];

    for (i = 0; i < 5; i++)
    {
       for (j = 0; j < BLOCK_SIZE; j++)
             output[j] = input[shuffle[j] - 1];

        memcpy(input, output, BLOCK_SIZE);

        for (j = 0; j < BLOCK_SIZE; j += 2)
        {
            input[j + 1] += input[j];
            input[j] += input[j + 1];
        }
    }
}

void LinearTransformDecrypt( unsigned char* input, const unsigned char* shuffle)
{
    int i, j;
    unsigned char output[BLOCK_SIZE];

    for (i = 0; i < 5; i++)
    {
       for (j = 0; j < BLOCK_SIZE; j += 2)
       {
             input[j] -= input[j + 1];
             input[j + 1] -= input[j];
       }
       for (j = 0; j < BLOCK_SIZE; j++)
             output[shuffle[j] - 1] = input[j];

        memcpy(input, output, BLOCK_SIZE);
    }
}
```

SAFER-256 համակարգի դիֆերենցիալ վերլուծության ծրագրային կոդը (C++ լեզվով)

```
//PHTSolver.h

#ifndef _COMP_PHTSOLVER_
#define _COMP_PHTSOLVER_
#define MAX_LEN     32
#define OP_XOR      0
#define OP_ADD      1
#define OP_EXP      2
#define OP_LOG      3

extern int DefaultRoundsCount ;
extern int Default_EXP_LOG_Transform_Root ;
extern int DefaultPHTTransformDepth ;
extern int Default_XOR_ADD_Define_Vector[2][MAX_LEN] ;
extern int Default_EXP_LOG_Define_Vector[MAX_LEN] ;
extern int DefaultPHTLines[MAX_LEN] ;
extern int DefaultPHT2Lines[MAX_LEN];

void SubVectors(const int* pVectorA,const int* pVectorB,int* pVectorC,int iVectorLength);
void AddVectors(const int* pVectorA,const int* pVectorB,int* pVectorC,int iVectorLength);
void MulNumToVector(int iNum,const int* pVectorA,int* pVectorB,int iVectorLength);

#define EC_NON_ZERO 1
#define EC_ZERO 2
#define EC_GREATE_THAN_ZERO 3
#define EC_LESS_THAN_ZERO 4

int GetVectorElementCount(int iCountParam,const int* pVector,int iVectorLength);
int CalcCoefMatrix(const int* pVectorPHTLines,int iPHTDepth,int* pMatrix);
int CalcCoefInvMatrix(const int* pVectorPHTLines,int iPHTDepth,int* pMatrix);
int GetNextOdd(unsigned char* v, int start,int end);

#endif _COMP_PHTSOLVER_

//PHTSolver.cpp

#include "stdafx.h"
#include "stdio.h"
#include "PHTSolver.h"
#include <memory.h>

int DefaultRoundsCount = 5 ;
int Default_EXP_LOG_Transform_Root = 45 ;
int DefaultPHTTransformDepth = 4 ;
int Default_XOR_ADD_Define_Vector[2][MAX_LEN] =
      {
                {OP_XOR,OP_XOR,OP_XOR,OP_XOR,OP_ADD,OP_ADD,OP_ADD,OP_ADD,OP_XOR,OP_XOR,OP_XOR,OP_XOR,OP_ADD,O
                P_ADD,OP_ADD,OP_ADD,OP_XOR,OP_XOR,OP_XOR,OP_XOR,OP_ADD,OP_ADD,OP_ADD,OP_ADD,OP_XOR,OP_XOR,OP_
                XOR,OP_XOR,OP_ADD,OP_ADD,OP_ADD,OP_ADD},
                {OP_ADD,OP_ADD,OP_ADD,OP_ADD,OP_XOR,OP_XOR,OP_XOR,OP_XOR,OP_ADD,OP_ADD,OP_ADD,OP_ADD,OP_XOR,O
                P_XOR,OP_XOR,OP_XOR,OP_ADD,OP_ADD,OP_ADD,OP_ADD,OP_XOR,OP_XOR,OP_XOR,OP_XOR,OP_ADD,OP_ADD,OP_
                ADD,OP_ADD,OP_XOR,OP_XOR,OP_XOR,OP_XOR}
        };
int Default_EXP_LOG_Define_Vector[MAX_LEN] =
      {
                OP_EXP,OP_LOG,OP_LOG,OP_EXP,OP_EXP,OP_LOG,OP_LOG,OP_EXP,OP_EXP,OP_LOG,OP_LOG,OP_EXP,OP_EXP,OP
                _LOG,OP_LOG,OP_EXP,OP_EXP,OP_LOG,OP_LOG,OP_EXP,OP_EXP,OP_LOG,OP_LOG,OP_EXP,OP_EXP,OP_LOG,OP_L
                OG,OP_EXP,OP_EXP,OP_LOG,OP_LOG,OP_EXP
      };

// SAFER-256 shuffling
int DefaultPHT2Lines[MAX_LEN] = {25, 28, 29, 32, 17, 20, 21, 24, 13, 16, 9, 12, 5, 8, 1, 4, 3, 2, 7, 6, 11,
10, 15, 14, 27, 26, 31, 30, 19, 18, 23, 22};

void SubVectors(const int* pVectorA,const int* pVectorB,int* pVectorC,int iVectorLength)
{
      for ( int i = 0; i < iVectorLength; i++)
            pVectorC[i] = ( pVectorA[i] - pVectorB[i] ) % 256 ;
}
void AddVectors(const int* pVectorA,const int* pVectorB,int* pVectorC,int iVectorLength)
```

```
{
	for ( int i = 0; i < iVectorLength; i++)
		pVectorC[i] = ( pVectorA[i] + pVectorB[i] ) % 256 ;
}
void MulNumToVector(int iNum,const int* pVectorA,int* pVectorB,int iVectorLength)
{
	for ( int i = 0; i < iVectorLength; i++)
		pVectorB[i] = ( iNum * pVectorA[i] ) % 256 ;
}

#define EC_NON_ZERO 1
#define EC_ZERO 2
#define EC_GREATE_THAN_ZERO 3
#define EC_LESS_THAN_ZERO 4

int GetVectorElementCount(int iCountParam,const int* pVector,int iVectorLength)
{
	int iRet = 0 ;
	int i = 0 ;
	switch ( iCountParam )
	{
		case EC_NON_ZERO :
			for ( i = 0; i < iVectorLength; i++)
			{
				if ( pVector[i] != 0 ) iRet++ ;
			}
		break;
		case EC_ZERO :
			for ( i = 0; i < iVectorLength; i++)
			{
				if ( pVector[i] == 0 ) iRet++ ;
			}
		break;
		case EC_GREATE_THAN_ZERO :
			for ( i = 0; i < iVectorLength; i++)
			{
				if ( pVector[i] > 0 ) iRet++ ;
			}
		break;
		case EC_LESS_THAN_ZERO :
			for ( i = 0; i < iVectorLength; i++)
			{
				if ( pVector[i] < 0 ) iRet++ ;
			}
		break;
		default:
		break;
	}
	return iRet ;
}

// calculate coefficients of matrix from given PHT(lines)
int CalcCoefMatrix(const int* pVectorPHTLines,int iPHTDepth,int* pMatrix)
{
	int Matrix0[MAX_LEN*MAX_LEN];
	int  Matrix1[MAX_LEN*MAX_LEN];
	int  Vector[MAX_LEN];
	int* pMatrixDest  = Matrix0 ;
	int* pMatrixSrc   = Matrix1 ;
	int* pMatrixTemp  = NULL ;
	int* pVectorTempA = NULL ;
	int* pVectorTempB = NULL ;
	int* pVectorTempC = NULL ;
	int* pVectorTempD = Vector ;
	int i = 0 ;
	int j = 0 ;
	int k = 0 ;

	if( pMatrix == NULL )
		return -1;
	if( pVectorPHTLines == NULL )
		return -1;
	if( 1 > iPHTDepth || iPHTDepth > 256 )
		return -1;

	memset(pVectorTempD,0,sizeof(int)*MAX_LEN);
```

```
	memset(pMatrixDest,0,sizeof(int)*MAX_LEN*MAX_LEN);
	memset(pMatrixSrc,0,sizeof(int)*MAX_LEN*MAX_LEN);
	pMatrixTemp = pMatrixSrc ;
	for ( i = 0; i < MAX_LEN; i++)
	{
		*pMatrixTemp = 1 ;
		pMatrixTemp += (MAX_LEN+1) ;
	}
	for ( k = 0; k <= iPHTDepth; k++)
	{
		for ( i = 0, j = 0; i < (int)(MAX_LEN/2); i++, j+=2)
		{
			pVectorTempA = (pMatrixSrc + (*(pVectorPHTLines+j)-1)*MAX_LEN) ;
			pVectorTempB = (pMatrixSrc + (*(pVectorPHTLines+j+1)-1)*MAX_LEN) ;

			pVectorTempC = (pMatrixDest + j*MAX_LEN) ;
			MulNumToVector(2,pVectorTempA,pVectorTempD,MAX_LEN);
			AddVectors(pVectorTempD,pVectorTempB,pVectorTempC,MAX_LEN);

			pVectorTempC = (pMatrixDest + (j+1)*MAX_LEN) ;
			AddVectors(pVectorTempA,pVectorTempB,pVectorTempC,MAX_LEN);
		}
		pMatrixTemp = pMatrixSrc ;
		pMatrixSrc = pMatrixDest ;
		pMatrixDest = pMatrixTemp ;
	}
	memcpy(pMatrix,pMatrixDest,sizeof(int)*MAX_LEN*MAX_LEN);

	return 1 ;
}

// calculate coefficients of inverse matrix from given PHT(lines)
int CalcCoefInvMatrix(const int* pVectorPHTLines,int iPHTDepth,int* pMatrix)
{
	int  Matrix0[MAX_LEN*MAX_LEN];
	int  Matrix1[MAX_LEN*MAX_LEN];
	int  Vector[MAX_LEN];
	int* pMatrixDest  = Matrix0 ;
	int* pMatrixSrc   = Matrix1 ;
	int* pMatrixTemp  = NULL ;
	int* pVectorTempA = NULL ;
	int* pVectorTempB = NULL ;
	int* pVectorTempC = NULL ;
	int* pVectorTempD = Vector ;
	int i = 0 ;
	int j = 0 ;
	int k = 0 ;

	if( pMatrix == NULL )
		return -1;
	if( pVectorPHTLines == NULL )
		return -1;
	if( 1 > iPHTDepth || iPHTDepth > 256 )
		return -1;

	memset(pVectorTempD,0,sizeof(int)*MAX_LEN);
	memset(pMatrixDest,0,sizeof(int)*MAX_LEN*MAX_LEN);
	memset(pMatrixSrc,0,sizeof(int)*MAX_LEN*MAX_LEN);
	pMatrixTemp = pMatrixSrc ;
	for ( i = 0; i < MAX_LEN; i++)
	{
		*pMatrixTemp = 1 ;
		pMatrixTemp += (MAX_LEN+1) ;
	}
	for ( k = 0; k <= iPHTDepth; k++)
	{
		for ( i = 0, j = 0; i < (int)(MAX_LEN/2); i++, j+=2)
		{
			pVectorTempA = (pMatrixSrc + (*(pVectorPHTLines+j)-1)*MAX_LEN) ;
			pVectorTempB = (pMatrixSrc + (*(pVectorPHTLines+j+1)-1)*MAX_LEN) ;

			pVectorTempC = (pMatrixDest + j*MAX_LEN) ;
			SubVectors(pVectorTempA,pVectorTempB,pVectorTempC,MAX_LEN);

			pVectorTempC = (pMatrixDest + (j+1)*MAX_LEN) ;
			MulNumToVector(2,pVectorTempB,pVectorTempD,MAX_LEN);
```

```
                  SubVectors(pVectorTempD,pVectorTempA,pVectorTempC,MAX_LEN);
            }
            pMatrixTemp = pMatrixSrc ;
            pMatrixSrc = pMatrixDest ;
            pMatrixDest = pMatrixTemp ;
      }
      memcpy(pMatrix,pMatrixDest,sizeof(int)*MAX_LEN*MAX_LEN);

      return 1 ;
}

//Solve.h

#if !defined(SOLVE_H)
#define SOLVE_H

extern unsigned short spp_shuffling0[32];
extern unsigned short spp_shuffling1[32];
extern uchar encPHT2Matrix[32][32];
extern uchar decPHT2Matrix[32][32];

int initCycle(unsigned short* curCycleState,
                     unsigned short minValue,
                     unsigned short maxValue,
                     unsigned short iCyclesCount,
                     unsigned short allVariants);
int stepCycle(unsigned short* curCycleState,
                     unsigned short minValue,
                     unsigned short maxValue,
                     unsigned short iCyclesCount,
                     unsigned short allVariants);
void decryption_matrix(unsigned char* in, unsigned char* out);
void encryption_matrix(unsigned char* in, unsigned char* out);
int SolveAnEquation(CResult* solution,
                                unsigned char*  matrix,
                                unsigned short* validVars,
                                unsigned char  validVarsCount,
                                unsigned short* validEquations,
                                unsigned char  validEquationsCount);
unsigned char GetNonzeroOutputs(CResult* sol, unsigned char*  matrix);
unsigned char VectorWeight( unsigned char* vector, int len);
BOOL isIn(unsigned char iNum,unsigned char* arrNum,int len);
void FillInvArray();

#endif // !defined(SOLVE_H)

//Solve.cpp

#include "stdafx.h"
#include "SaferCryptoAnalyzerIIDoc.h"
#include "solve.h"
#include "PHTSolver.h"

#define SPP_EXP 1
#define SPP_LOG 0

unsigned short spp_shuffling0[32] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
unsigned short spp_shuffling1[32] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
uchar encPHT2Matrix[32][32];
uchar decPHT2Matrix[32][32];

#define sh(i) spp_shuffling0[i]

unsigned short spp_explog[32] =
{SPP_EXP,SPP_EXP,SPP_EXP,SPP_EXP,SPP_LOG,SPP_LOG,SPP_LOG,SPP_LOG,SPP_EXP,SPP_EXP,SPP_EXP,SPP_EXP,SPP_LOG,SPP_
LOG,SPP_LOG,SPP_LOG,SPP_EXP,SPP_EXP,SPP_EXP,SPP_EXP,SPP_LOG,SPP_LOG,SPP_LOG,SPP_LOG,SPP_EXP,SPP_EXP,SPP_EXP,S
PP_EXP,SPP_LOG,SPP_LOG,SPP_LOG,SPP_LOG};

int initCycle(unsigned short* curCycleState,
                     unsigned short minValue,
                     unsigned short maxValue,
                     unsigned short iCyclesCount,
                     unsigned short allVariants)
{
      unsigned int i;
```

```
	ASSERT( minValue <= maxValue ) ;

	if ( !allVariants ){
		ASSERT( minValue + iCyclesCount - 1 <= maxValue);
		for ( i = 0; i < iCyclesCount; i++)
			curCycleState[i] = minValue + i;
	} else {
		for ( i = 0; i < iCyclesCount; i++)
			curCycleState[i] = minValue;
	}
	curCycleState[iCyclesCount-1]--;

	return 1;
}
int stepCycle(unsigned short* curCycleState,
		unsigned short minValue,
		unsigned short maxValue,
		unsigned short iCyclesCount,
		unsigned short allVariants)
{
	int currCycle = iCyclesCount - 1;

	ASSERT( minValue <= maxValue );

	if ( !allVariants ){
		ASSERT( minValue + iCyclesCount - 1 <= maxValue);
		while ( currCycle >= 0 ){
			curCycleState[currCycle]++;
			if ( curCycleState[currCycle] > (maxValue - (iCyclesCount-currCycle-1)) ){
				currCycle-- ;
			} else {
				currCycle++;
				while ( currCycle < iCyclesCount ){
					curCycleState[currCycle] = curCycleState[currCycle-1]+1;
					currCycle++;
				}
				return 1;
			}
		}
		return 0;
	} else {
		while ( currCycle >= 0 ){
			curCycleState[currCycle]++;
			if ( curCycleState[currCycle] > maxValue ){
				curCycleState[currCycle] = minValue;
				currCycle--;
			} else {
				return 1;
			}
		}
	}
	return 0;
}
#define	decPHT2_0(i1,i2,o1,o2)\
{\
	buff[o2] = 2*in[i2]-in[i1];\
	buff[o1] = in[i1]-in[i2];\
}
#define	decPHT2_1(i1,i2,o1,o2)\
{\
	out[o2] = 2*buff[i2]-buff[i1];\
	out[o1] = buff[i1]-buff[i2];\
}
#define	decPHT2_2(i1,i2,o1,o2)\
{\
	buff[o2] = 2*out[i2]-out[i1];\
	buff[o1] = out[i1]-out[i2];\
}
#define	decPHT2_3(i1,i2,o1,o2)\
{\
	out[o1] = buff[i1];\
	out[o2] = buff[i2];\
}
void decryption_matrix(unsigned char* in, unsigned char* out)
{
	unsigned char buff[32];
```

```
decPHT2_0(0, 1, sh( 0),sh( 1));
decPHT2_0(2, 3, sh( 2),sh( 3));
decPHT2_0(4, 5, sh( 4),sh( 5));
decPHT2_0(6, 7, sh( 6),sh( 7));
decPHT2_0(8, 9, sh( 8),sh( 9));
decPHT2_0(10, 11, sh(10),sh(11));
decPHT2_0(12, 13, sh(12),sh(13));
decPHT2_0(14, 15, sh(14),sh(15));
decPHT2_0(16, 17, sh(16),sh(17));
decPHT2_0(18, 19, sh(18),sh(19));
decPHT2_0(20, 21, sh(20),sh(21));
decPHT2_0(22, 23, sh(22),sh(23));
decPHT2_0(24, 25, sh(24),sh(25));
decPHT2_0(26, 27, sh(26),sh(27));
decPHT2_0(28, 29, sh(28),sh(29));
decPHT2_0(30, 31, sh(30),sh(31));

decPHT2_1(0, 1, sh( 0),sh( 1));
decPHT2_1(2, 3, sh( 2),sh( 3));
decPHT2_1(4, 5, sh( 4),sh( 5));
decPHT2_1(6, 7, sh( 6),sh( 7));
decPHT2_1(8, 9, sh( 8),sh( 9));
decPHT2_1(10, 11, sh(10),sh(11));
decPHT2_1(12, 13, sh(12),sh(13));
decPHT2_1(14, 15, sh(14),sh(15));
decPHT2_1(16, 17, sh(16),sh(17));
decPHT2_1(18, 19, sh(18),sh(19));
decPHT2_1(20, 21, sh(20),sh(21));
decPHT2_1(22, 23, sh(22),sh(23));
decPHT2_1(24, 25, sh(24),sh(25));
decPHT2_1(26, 27, sh(26),sh(27));
decPHT2_1(28, 29, sh(28),sh(29));
decPHT2_1(30, 31, sh(30),sh(31));

decPHT2_2(0, 1, sh( 0),sh( 1));
decPHT2_2(2, 3, sh( 2),sh( 3));
decPHT2_2(4, 5, sh( 4),sh( 5));
decPHT2_2(6, 7, sh( 6),sh( 7));
decPHT2_2(8, 9, sh( 8),sh( 9));
decPHT2_2(10, 11, sh(10),sh(11));
decPHT2_2(12, 13, sh(12),sh(13));
decPHT2_2(14, 15, sh(14),sh(15));
decPHT2_2(16, 17, sh(16),sh(17));
decPHT2_2(18, 19, sh(18),sh(19));
decPHT2_2(20, 21, sh(20),sh(21));
decPHT2_2(22, 23, sh(22),sh(23));
decPHT2_2(24, 25, sh(24),sh(25));
decPHT2_2(26, 27, sh(26),sh(27));
decPHT2_2(28, 29, sh(28),sh(29));
decPHT2_2(30, 31, sh(30),sh(31));

decPHT2_1(0, 1, sh( 0),sh( 1));
decPHT2_1(2, 3, sh( 2),sh( 3));
decPHT2_1(4, 5, sh( 4),sh( 5));
decPHT2_1(6, 7, sh( 6),sh( 7));
decPHT2_1(8, 9, sh( 8),sh( 9));
decPHT2_1(10, 11, sh(10),sh(11));
decPHT2_1(12, 13, sh(12),sh(13));
decPHT2_1(14, 15, sh(14),sh(15));
decPHT2_1(16, 17, sh(16),sh(17));
decPHT2_1(18, 19, sh(18),sh(19));
decPHT2_1(20, 21, sh(20),sh(21));
decPHT2_1(22, 23, sh(22),sh(23));
decPHT2_1(24, 25, sh(24),sh(25));
decPHT2_1(26, 27, sh(26),sh(27));
decPHT2_1(28, 29, sh(28),sh(29));
decPHT2_1(30, 31, sh(30),sh(31));

decPHT2_2(0, 1, sh( 0),sh( 1));
decPHT2_2(2, 3, sh( 2),sh( 3));
decPHT2_2(4, 5, sh( 4),sh( 5));
decPHT2_2(6, 7, sh( 6),sh( 7));
decPHT2_2(8, 9, sh( 8),sh( 9));
decPHT2_2(10, 11, sh(10),sh(11));
decPHT2_2(12, 13, sh(12),sh(13));
```

```
		decPHT2_2(14, 15, sh(14),sh(15));
		decPHT2_2(16, 17, sh(16),sh(17));
		decPHT2_2(18, 19, sh(18),sh(19));
		decPHT2_2(20, 21, sh(20),sh(21));
		decPHT2_2(22, 23, sh(22),sh(23));
		decPHT2_2(24, 25, sh(24),sh(25));
		decPHT2_2(26, 27, sh(26),sh(27));
		decPHT2_2(28, 29, sh(28),sh(29));
		decPHT2_2(30, 31, sh(30),sh(31));

		decPHT2_3(0, 1, 0, 1);
		decPHT2_3(2, 3, 2, 3);
		decPHT2_3(4, 5, 4, 5);
		decPHT2_3(6, 7, 6, 7);
		decPHT2_3(8, 9, 8, 9);
		decPHT2_3(10,11,10,11);
		decPHT2_3(12,13,12,13);
		decPHT2_3(14,15,14,15);
		decPHT2_3(16,17,16,17);
		decPHT2_3(18,19,18,19);
		decPHT2_3(20,21,20,21);
		decPHT2_3(22,23,22,23);
		decPHT2_3(24,25,24,25);
		decPHT2_3(26,27,26,27);
		decPHT2_3(28,29,28,29);
		decPHT2_3(30,31,30,31);
}
#define	encPHT2_0(i1,i2,o1,o2)\
{\
	buff[o2] = in[i1]+in[i2];\
	buff[o1] = buff[o2]+in[i1];\
}
#define	encPHT2_1(i1,i2,o1,o2)\
{\
	out[o2] = buff[i1]+buff[i2];\
	out[o1] = out[o2]+buff[i1];\
}
#define	encPHT2_2(i1,i2,o1,o2)\
{\
	buff[o2] = out[i1]+out[i2];\
	buff[o1] = buff[o2]+out[i1];\
}
#define	encPHT2_3(i1,i2,o1,o2)\
{\
	out[o2] = buff[i2];\
	out[o1] = buff[i1];\
}
void encryption_matrix(unsigned char* in, unsigned char* out)
{
	unsigned char buff[32];

		encPHT2_0(sh( 0),sh( 1), 0, 1);
		encPHT2_0(sh( 2),sh( 3), 2, 3);
		encPHT2_0(sh( 4),sh( 5), 4, 5);
		encPHT2_0(sh( 6),sh( 7), 6, 7);
		encPHT2_0(sh( 8),sh( 9), 8, 9);
		encPHT2_0(sh(10),sh(11),10,11);
		encPHT2_0(sh(12),sh(13),12,13);
		encPHT2_0(sh(14),sh(15),14,15);
		encPHT2_0(sh(16),sh(17),16,17);
		encPHT2_0(sh(18),sh(19),18,19);
		encPHT2_0(sh(20),sh(21),20,21);
		encPHT2_0(sh(22),sh(23),22,23);
		encPHT2_0(sh(24),sh(25),24,25);
		encPHT2_0(sh(26),sh(27),26,27);
		encPHT2_0(sh(28),sh(29),28,29);
		encPHT2_0(sh(30),sh(31),30,31);

		encPHT2_1(sh( 0),sh( 1), 0, 1);
		encPHT2_1(sh( 2),sh( 3), 2, 3);
		encPHT2_1(sh( 4),sh( 5), 4, 5);
		encPHT2_1(sh( 6),sh( 7), 6, 7);
		encPHT2_1(sh( 8),sh( 9), 8, 9);
		encPHT2_1(sh(10),sh(11),10,11);
		encPHT2_1(sh(12),sh(13),12,13);
		encPHT2_1(sh(14),sh(15),14,15);
```

```
encPHT2_1(sh(16),sh(17),16,17);
encPHT2_1(sh(18),sh(19),18,19);
encPHT2_1(sh(20),sh(21),20,21);
encPHT2_1(sh(22),sh(23),22,23);
encPHT2_1(sh(24),sh(25),24,25);
encPHT2_1(sh(26),sh(27),26,27);
encPHT2_1(sh(28),sh(29),28,29);
encPHT2_1(sh(30),sh(31),30,31);

encPHT2_2(sh( 0),sh( 1), 0, 1);
encPHT2_2(sh( 2),sh( 3), 2, 3);
encPHT2_2(sh( 4),sh( 5), 4, 5);
encPHT2_2(sh( 6),sh( 7), 6, 7);
encPHT2_2(sh( 8),sh( 9), 8, 9);
encPHT2_2(sh(10),sh(11),10,11);
encPHT2_2(sh(12),sh(13),12,13);
encPHT2_2(sh(14),sh(15),14,15);
encPHT2_2(sh(16),sh(17),16,17);
encPHT2_2(sh(18),sh(19),18,19);
encPHT2_2(sh(20),sh(21),20,21);
encPHT2_2(sh(22),sh(23),22,23);
encPHT2_2(sh(24),sh(25),24,25);
encPHT2_2(sh(26),sh(27),26,27);
encPHT2_2(sh(28),sh(29),28,29);
encPHT2_2(sh(30),sh(31),30,31);

encPHT2_1(sh( 0),sh( 1), 0, 1);
encPHT2_1(sh( 2),sh( 3), 2, 3);
encPHT2_1(sh( 4),sh( 5), 4, 5);
encPHT2_1(sh( 6),sh( 7), 6, 7);
encPHT2_1(sh( 8),sh( 9), 8, 9);
encPHT2_1(sh(10),sh(11),10,11);
encPHT2_1(sh(12),sh(13),12,13);
encPHT2_1(sh(14),sh(15),14,15);
encPHT2_1(sh(16),sh(17),16,17);
encPHT2_1(sh(18),sh(19),18,19);
encPHT2_1(sh(20),sh(21),20,21);
encPHT2_1(sh(22),sh(23),22,23);
encPHT2_1(sh(24),sh(25),24,25);
encPHT2_1(sh(26),sh(27),26,27);
encPHT2_1(sh(28),sh(29),28,29);
encPHT2_1(sh(30),sh(31),30,31);

encPHT2_2(sh( 0),sh( 1), 0, 1);
encPHT2_2(sh( 2),sh( 3), 2, 3);
encPHT2_2(sh( 4),sh( 5), 4, 5);
encPHT2_2(sh( 6),sh( 7), 6, 7);
encPHT2_2(sh( 8),sh( 9), 8, 9);
encPHT2_2(sh(10),sh(11),10,11);
encPHT2_2(sh(12),sh(13),12,13);
encPHT2_2(sh(14),sh(15),14,15);
encPHT2_2(sh(16),sh(17),16,17);
encPHT2_2(sh(18),sh(19),18,19);
encPHT2_2(sh(20),sh(21),20,21);
encPHT2_2(sh(22),sh(23),22,23);
encPHT2_2(sh(24),sh(25),24,25);
encPHT2_2(sh(26),sh(27),26,27);
encPHT2_2(sh(28),sh(29),28,29);
encPHT2_2(sh(30),sh(31),30,31);

encPHT2_3(0, 1, 0, 1);
encPHT2_3(2, 3, 2, 3);
encPHT2_3(4, 5, 4, 5);
encPHT2_3(6, 7, 6, 7);
encPHT2_3(8, 9, 8, 9);
encPHT2_3(10,11,10,11);
encPHT2_3(12,13,12,13);
encPHT2_3(14,15,14,15);
encPHT2_3(16,17,16,17);
encPHT2_3(18,19,18,19);
encPHT2_3(20,21,20,21);
encPHT2_3(22,23,22,23);
encPHT2_3(24,25,24,25);
encPHT2_3(26,27,26,27);
encPHT2_3(28,29,28,29);
encPHT2_3(30,31,30,31);
```

```
}
BOOL isIn(unsigned char iNum,unsigned char* arrNum,int len)
{
	for ( int i = 0; i < len; i++)
		if ( iNum == arrNum[i] )
			return TRUE;
	return FALSE;
}
int GetNextOdd(unsigned char* v, int start,int end)
{
	for ( int i = start; i < end; i++ )
		if ( v[i] & 1 )
			return i;
	return -1;
}
unsigned char Inv[256];
unsigned char GetInverse(unsigned char ch)
{
	for ( int i = 0; i < 256; i++)
		if ( (unsigned char)(i * ch) == 1 )
			return i;
	return 0;
}
void FillInvArray()
{
	int i ;
	Inv[0] = 0;
	for ( i = 1; i < 256; i++ ) Inv[i] = GetInverse(i);
	Inv[128] = 0;
}
void AddVectors(unsigned char* vres, unsigned char* v1, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++)  vres[i] = v1[i] + v2[i];
}
void SubVectors(unsigned char* vres, unsigned char* v1, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++)  vres[i] = v1[i] - v2[i];
}
void AddVectorsX(unsigned char* v1, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++)  v1[i] += v2[i];
}
void SubVectorsX(unsigned char* v1, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++)  v2[i] =v1[i] - v2[i];
}
void MulNumToVector(unsigned char* vres, unsigned char num, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++) vres[i] = num * v2[i];
}
void MulNumToVectorX(unsigned char num, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++) v2[i] = num * v2[i];
}
void DivVectorToNumX(unsigned char num, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++) v2[i] = v2[i]/num;
}
void InsertValue(unsigned char* vres, unsigned char* v1, unsigned char* v2, unsigned char pos, unsigned char
len)
{
	unsigned char koef = v1[pos];
	v1[pos] = 0;
	for ( int i = 0; i < len; i++) vres[i] = koef*v2[i] + v1[i];
}
void InsertValueQ(unsigned char* validInputs,unsigned char* vres, unsigned char* v1, unsigned char* v2,
unsigned char pos, unsigned char len)
{
	unsigned char koef = v1[validInputs[pos]];
	v1[pos] = 0;
	for ( int i = 0; i < len; i++) vres[validInputs[i]] = koef*v2[i] + v1[validInputs[i]];
}
void InsertValueX(unsigned char* v1, unsigned char* v2, unsigned char pos, unsigned char len)
{
	unsigned char koef = v1[pos];
	v1[pos] = 0;
```

```
	for ( int i = 0; i < len; i++) v1[i] += koef*v2[i];
}
void InsertValueQX(unsigned char* validInputs, unsigned char* v1, unsigned char* v2, unsigned char pos,
unsigned char len)
{
	unsigned char koef = v1[validInputs[pos]];
	v1[pos] = 0;
	for ( int i = 0; i < len; i++) v1[validInputs[i]] += koef*v2[i];
}
int GetNextNonZero(unsigned char* v, int start,int end)
{
	for ( int i = start; i < end; i++ )
		if ( v[i] ) return i;
	return -1;
}
void NegVector( unsigned char* vector, int start, int end)
{
	for ( int i = start; i < end; i++) vector[i] = -vector[i];
}
unsigned char VectorWeight( unsigned char* vector, int len)
{
	unsigned char w = 0;
	for ( int i = 0; i < len; i++)
		if ( vector[i] ) w++;
	return w;
}
int SolveAnEquation(CResult* solution,
				unsigned char*  matrix,
				unsigned short* validVars,
				unsigned char  validVarsCount,
				unsigned short* validEquations,
				unsigned char  validEquationsCount)
{
	unsigned char modifmatrix[32][32];
	unsigned char *pEq[32];
	unsigned char *pTmp;
	unsigned char vtemp1[32];
	unsigned char w, minw, iminwEq;
	int i, j, iOddPos, iOddsCnt;
	int iLastOddLine, iLastOddLine2, iLastOddLine4, iLastOddLine8, iLastOddLine10, iLastOddLine12,
	iLastOddLine14, iLastOddLine16;
	BOOL b128;

	ASSERT( validVarsCount > validEquationsCount );

	for ( i = 0; i < validEquationsCount; i++){
		pEq[i] = modifmatrix[i];
		for ( j = 0; j < validVarsCount; j++)
			modifmatrix[i][j] = matrix[validVars[j]*32+validEquations[i]];
	}
	for ( i = 0; i < validEquationsCount; i++){
		if ( VectorWeight(modifmatrix[i], validVarsCount) < validVarsCount-validEquationsCount){
			// no solution
			return -1;
		}
	}
	minw = 100;
	iOddsCnt = 0;
	iLastOddLine = -1;
	for ( i = 0; i < validEquationsCount; i++){
		iOddPos = -1;
		iOddPos = GetNextOdd( pEq[i], iOddPos+1, validVarsCount);
		if (iOddPos != -1){
			if ( i-1 != iLastOddLine ){
				pTmp = pEq[iLastOddLine+1];
				pEq[iLastOddLine+1] = pEq[i];
				pEq[i] = pTmp;
				i = iLastOddLine+1;
			}
			for ( j = i+1; j < validEquationsCount; j++){
				MulNumToVector(vtemp1, Inv[pEq[i][iOddPos]]*pEq[j][iOddPos], pEq[i],
validVarsCount);
				SubVectorsX(vtemp1, pEq[j], validVarsCount);
				if ( (w = VectorWeight(pEq[j], validVarsCount)) < validVarsCount-
validEquationsCount+1){
					// no solution
```

```
                                return -1;
                                } else {
                                      if ( w < minw ){
                                            minw = w;
                                      // ;) allways iminwEq == validEquationsCount-1
                                            iminwEq = i;
                                      }
                                }
                          }
                          iOddsCnt++;
                          iLastOddLine = i;
                          if ( iLastOddLine == validEquationsCount-2 )
                                break;
                    }
              }
              iLastOddLine2 = validEquationsCount;
              if ( iOddsCnt < validEquationsCount-1 ){
                    // find non odd vectors and divide on 2 ??????
                    iLastOddLine2 = iLastOddLine+1;
                    while ( GetNextOdd( pEq[iLastOddLine2], 0, validVarsCount) == -1 ){
                          for ( i = iLastOddLine2; i < validEquationsCount; i++){
                                DivVectorToNumX(2, pEq[i], validVarsCount);
                          }
                    }
                    // solve equations
                    iLastOddLine = iLastOddLine2-1;
                    for ( i = iLastOddLine2; i < validEquationsCount; i++){
                          iOddPos = -1;
                          iOddPos = GetNextOdd( pEq[i], iOddPos+1, validVarsCount);
                          if (iOddPos != -1){
                                if ( i-1 != iLastOddLine ){
                                      pTmp = pEq[iLastOddLine+1];
                                      pEq[iLastOddLine+1] = pEq[i];
                                      pEq[i] = pTmp;
                                      i = iLastOddLine+1;
                                }
                                for ( j = i+1; j < validEquationsCount; j++){
                                      MulNumToVector(vtemp1, Inv[pEq[i][iOddPos]]*pEq[j][iOddPos], pEq[i],
validVarsCount);
                                      SubVectorsX(vtemp1, pEq[j], validVarsCount);
                                      if ( VectorWeight(pEq[j], validVarsCount) < validVarsCount-
validEquationsCount+1){
                                            // no solution
                                            return -1;
                                      } else {
                                            if ( w < minw ){
                                                  minw = w;
                                            // ;) allways iminwEq == validEquationsCount-1
                                                  iminwEq = i;
                                            }
                                      }
                                }
                                iOddsCnt++;
                                iLastOddLine = i;
                                if ( iLastOddLine == validEquationsCount-2 )
                                      break;
                          }
                    }
              }
              iLastOddLine4 = validEquationsCount;
              if ( iOddsCnt < validEquationsCount-1 ){
                    // find non odd vectors and divide on 2 ??????
                    iLastOddLine4 = iLastOddLine+1;
                    while ( GetNextOdd( pEq[iLastOddLine4], 0, validVarsCount) == -1 ){
                          for ( i = iLastOddLine4; i < validEquationsCount; i++){
                                DivVectorToNumX(2, pEq[i], validVarsCount);
                          }
                    }
                    // solve equations
                    iLastOddLine = iLastOddLine4-1;
                    for ( i = iLastOddLine4; i < validEquationsCount; i++){
                          iOddPos = -1;
                          iOddPos = GetNextOdd( pEq[i], iOddPos+1, validVarsCount);
                          if (iOddPos != -1){
                                if ( i-1 != iLastOddLine ){
                                      pTmp = pEq[iLastOddLine+1];
```

```
                                pEq[iLastOddLine+1] = pEq[i];
                                pEq[i] = pTmp;
                                i = iLastOddLine+1;
                        }
                        for ( j = i+1; j < validEquationsCount; j++){
                                MulNumToVector(vtemp1, Inv[pEq[i][iOddPos]]*pEq[j][iOddPos], pEq[i],
validVarsCount);
                                SubVectorsX(vtemp1, pEq[j], validVarsCount);
                                if ( VectorWeight(pEq[j], validVarsCount) < validVarsCount-
validEquationsCount+1){
                                        // no solution
                                        return -1;
                                } else {
                                        if ( w < minw ){
                                                minw = w;
                                        // ;) allways iminwEq == validEquationsCount-1
                                                iminwEq = i;
                                        }
                                }
                        }
                        iOddsCnt++;
                        iLastOddLine = i;
                        if ( iLastOddLine == validEquationsCount-2 )
                                break;
                }
        }
    }
    iLastOddLine8 = validEquationsCount;
    if ( iOddsCnt < validEquationsCount-1 ){
        // find non odd vectors and divide on 2 ??????
        iLastOddLine8 = iLastOddLine+1;
        while ( GetNextOdd( pEq[iLastOddLine8], 0, validVarsCount) == -1 ){
                for ( i = iLastOddLine8; i < validEquationsCount; i++)
                        DivVectorToNumX(2, pEq[i], validVarsCount);
        }
        // solve equations
        iLastOddLine = iLastOddLine8-1;
        for ( i = iLastOddLine8; i < validEquationsCount; i++){
                iOddPos = -1;
                iOddPos = GetNextOdd( pEq[i], iOddPos+1, validVarsCount);
                if (iOddPos != -1){
                        if ( i-1 != iLastOddLine ){
                                pTmp = pEq[iLastOddLine+1];
                                pEq[iLastOddLine+1] = pEq[i];
                                pEq[i] = pTmp;
                                i = iLastOddLine+1;
                        }
                        for ( j = i+1; j < validEquationsCount; j++){
                                MulNumToVector(vtemp1, Inv[pEq[i][iOddPos]]*pEq[j][iOddPos], pEq[i],
validVarsCount);
                                SubVectorsX(vtemp1, pEq[j], validVarsCount);
                                if ( VectorWeight(pEq[j], validVarsCount) < validVarsCount-
validEquationsCount+1){
                                        // no solution
                                        return -1;
                                } else {
                                        if ( w < minw ){
                                                minw = w;
                                        // ;) allways iminwEq == validEquationsCount-1
                                                iminwEq = i;
                                        }
                                }
                        }
                        iOddsCnt++;
                        iLastOddLine = i;
                        if ( iLastOddLine == validEquationsCount-2 )
                                break;
                }
        }
    }
    iLastOddLine10 = validEquationsCount;
    if ( iOddsCnt < validEquationsCount-1 ){
        // find non odd vectors and divide on 2 ??????
        iLastOddLine10 = iLastOddLine+1;
        while ( GetNextOdd( pEq[iLastOddLine10], 0, validVarsCount) == -1 ){
                for ( i = iLastOddLine10; i < validEquationsCount; i++)
```

```
                        DivVectorToNumX(2, pEq[i], validVarsCount);
                }
                // solve equations
                iLastOddLine = iLastOddLine10-1;
                for ( i = iLastOddLine10; i < validEquationsCount; i++){
                        iOddPos = -1;
                        iOddPos = GetNextOdd( pEq[i], iOddPos+1, validVarsCount);
                        if (iOddPos != -1){
                                if ( i-1 != iLastOddLine ){
                                        pTmp = pEq[iLastOddLine+1];
                                        pEq[iLastOddLine+1] = pEq[i];
                                        pEq[i] = pTmp;
                                        i = iLastOddLine+1;
                                }
                                for ( j = i+1; j < validEquationsCount; j++){
                                        MulNumToVector(vtemp1, Inv[pEq[i][iOddPos]]*pEq[j][iOddPos], pEq[i],
validVarsCount);
                                        SubVectorsX(vtemp1, pEq[j], validVarsCount);
                                        if ( VectorWeight(pEq[j], validVarsCount) < validVarsCount-
validEquationsCount+1){
                                                // no solution
                                                return -1;
                                        } else {
                                                if ( w < minw ){
                                                        minw = w;
                                                // ;) allways iminwEq == validEquationsCount-1
                                                        iminwEq = i;
                                                }
                                        }
                                }
                                iOddsCnt++;
                                iLastOddLine = i;
                                if ( iLastOddLine == validEquationsCount-2 )
                                        break;
                        }
                }
        }
        iLastOddLine12 = validEquationsCount;
        if ( iOddsCnt < validEquationsCount-1 ){
                // find non odd vectors and divide on 2 ??????
                iLastOddLine12 = iLastOddLine+1;
                while ( GetNextOdd( pEq[iLastOddLine12], 0, validVarsCount) == -1 ){
                        for ( i = iLastOddLine12; i < validEquationsCount; i++)
                                DivVectorToNumX(2, pEq[i], validVarsCount);
                }
                // solve equations
                iLastOddLine = iLastOddLine12-1;
                for ( i = iLastOddLine12; i < validEquationsCount; i++){
                        iOddPos = -1;
                        iOddPos = GetNextOdd( pEq[i], iOddPos+1, validVarsCount);
                        if (iOddPos != -1){
                                if ( i-1 != iLastOddLine ){
                                        pTmp = pEq[iLastOddLine+1];
                                        pEq[iLastOddLine+1] = pEq[i];
                                        pEq[i] = pTmp;
                                        i = iLastOddLine+1;
                                }
                                for ( j = i+1; j < validEquationsCount; j++){
                                        MulNumToVector(vtemp1, Inv[pEq[i][iOddPos]]*pEq[j][iOddPos], pEq[i],
validVarsCount);
                                        SubVectorsX(vtemp1, pEq[j], validVarsCount);
                                        if ( VectorWeight(pEq[j], validVarsCount) < validVarsCount-
validEquationsCount+1){
                                                // no solution
                                                return -1;
                                        } else {
                                                if ( w < minw ){
                                                        minw = w;
                                                // ;) allways iminwEq == validEquationsCount-1
                                                        iminwEq = i;
                                                }
                                        }
                                }
                                iOddsCnt++;
                                iLastOddLine = i;
                                if ( iLastOddLine == validEquationsCount-2 )
```

```
                                    break;
                        }
                  }
            }
            iLastOddLine14 = validEquationsCount;
            if ( iOddsCnt < validEquationsCount-1 ){
                  // find non odd vectors and divide on 2 ??????
                  iLastOddLine14 = iLastOddLine+1;
                  while ( GetNextOdd( pEq[iLastOddLine14], 0, validVarsCount) == -1 ){
                        for ( i = iLastOddLine14; i < validEquationsCount; i++)
                              DivVectorToNumX(2, pEq[i], validVarsCount);
                  }
                  // solve equations
                  iLastOddLine = iLastOddLine14-1;
                  for ( i = iLastOddLine14; i < validEquationsCount; i++){
                        iOddPos = -1;
                        iOddPos = GetNextOdd( pEq[i], iOddPos+1, validVarsCount);
                        if (iOddPos != -1){
                              if ( i-1 != iLastOddLine ){
                                    pTmp = pEq[iLastOddLine+1];
                                    pEq[iLastOddLine+1] = pEq[i];
                                    pEq[i] = pTmp;
                                    i = iLastOddLine+1;
                              }
                              for ( j = i+1; j < validEquationsCount; j++){
                                    MulNumToVector(vtemp1, Inv[pEq[i][iOddPos]]*pEq[j][iOddPos], pEq[i],
validVarsCount);
                                    SubVectorsX(vtemp1, pEq[j], validVarsCount);
                                    if ( VectorWeight(pEq[j], validVarsCount) < validVarsCount-
validEquationsCount+1){
                                          // no solution
                                          return -1;
                                    } else {
                                          if ( w < minw ){
                                                minw = w;
                                          // ;) allways iminwEq == validEquationsCount-1
                                                iminwEq = i;
                                          }
                                    }
                              }
                              iOddsCnt++;
                              iLastOddLine = i;
                              if ( iLastOddLine == validEquationsCount-2 )
                                    break;
                        }
                  }
            }
            iLastOddLine16 = validEquationsCount;
            if ( iOddsCnt < validEquationsCount-1 ){
                  // find non odd vectors and divide on 2 ??????
                  iLastOddLine16 = iLastOddLine+1;
                  while ( GetNextOdd( pEq[iLastOddLine16], 0, validVarsCount) == -1 ){
                        for ( i = iLastOddLine16; i < validEquationsCount; i++)
                              DivVectorToNumX(2, pEq[i], validVarsCount);
                  }
                  // solve equations
                  iLastOddLine = iLastOddLine16-1;
                  for ( i = iLastOddLine16; i < validEquationsCount; i++){
                        iOddPos = -1;
                        iOddPos = GetNextOdd( pEq[i], iOddPos+1, validVarsCount);
                        if (iOddPos != -1){
                              if ( i-1 != iLastOddLine ){
                                    pTmp = pEq[iLastOddLine+1];
                                    pEq[iLastOddLine+1] = pEq[i];
                                    pEq[i] = pTmp;
                                    i = iLastOddLine+1;
                              }
                              for ( j = i+1; j < validEquationsCount; j++){
                                    MulNumToVector(vtemp1, Inv[pEq[i][iOddPos]]*pEq[j][iOddPos], pEq[i],
validVarsCount);
                                    SubVectorsX(vtemp1, pEq[j], validVarsCount);
                                    if ( VectorWeight(pEq[j], validVarsCount) < validVarsCount-
validEquationsCount+1){
                                          // no solution
                                          return -1;
                                    } else {
```

```
                                    if ( w < minw ){
                                          minw = w;
                                    // ;) allways iminwEq == validEquationsCount-1
                                          iminwEq = i;
                                    }
                              }
                        }
                        iOddsCnt++;
                        iLastOddLine = i;
                        if ( iLastOddLine == validEquationsCount-2 )
                              break;
                  }
            }
      }
      b128 = FALSE;
      for ( i = validEquationsCount-1; i >= 0; i--){
            iOddPos = GetNextOdd( pEq[i], 0, validVarsCount);
            if ( iOddPos != -1 ){
                  MulNumToVector(&solution->m_varInput[i][2], Inv[pEq[i][iOddPos]], pEq[i], validVarsCount);
                  solution->m_varInput[i][0] = iOddPos;//(unsigned char)validVars[iOddPos];
                  solution->m_varInput[i][iOddPos+2] = 0;
                  if ( i >= iLastOddLine2 || b128 ){
                        solution->m_varInput[i][1] = 128;
                        b128 = FALSE;
                  }
                  else
                        solution->m_varInput[i][1] = 0;
                  NegVector( &solution->m_varInput[i][2], 0, validVarsCount);
                  for ( j = i-1; j >= 0; j--){
                        InsertValueX( pEq[j], &solution->m_varInput[i][2], iOddPos, validVarsCount);
                  }
                  if ( VectorWeight(&solution->m_varInput[i][2], validVarsCount) < validVarsCount-
validEquationsCount){
                        return -1; // no solution
                  }
            } else {
                  DivVectorToNumX(2, pEq[i], validVarsCount);
                  b128 = TRUE;
                  i++;
            }
      }
      solution->m_iInputWeight = validVarsCount;
      solution->m_ivarCount = validVarsCount - validEquationsCount;
      for ( i = 0; i < validVarsCount; i++){
            solution->m_nzInputBytes[i] = (unsigned char)validVars[i];
      }

      return 0;
}
unsigned char GetNonzeroOutputs(CResult* sol, unsigned char*  matrix)
{
      int i, j;
      int w = 0;

      for ( i = 0; i < 32; i++){
            for ( j = 0; j < sol->m_iInputWeight; j++)
                  sol->m_varOutput[i][j] = matrix[sol->m_nzInputBytes[j]*32+i];
      }

      for ( i = 0; i < sol->m_iInputWeight-sol->m_ivarCount; i++){
            for ( j = 0; j < 32; j++)
            InsertValueX(sol->m_varOutput[j], &sol->m_varInput[i][2], sol->m_varInput[i][0], sol-
>m_iInputWeight);
      }
      for ( j = 0; j < 32; j++){

            if ( VectorWeight(sol->m_varOutput[j], sol->m_iInputWeight) ){
                  sol->m_nzOutputBytes[w] = j;
                  w++;
            }
      }
      sol->m_iOutputWeight = w;
      return w;
}
```

Սա կոդի մի մասն է, ամբողջական կոդը և “exe” ֆայլը կոշտ սկավառակի վրա են:

Ձևափոխած SAFER+ համակարգի դիֆերենցիալ վերլուծության ծրագրային կոդը (C++ լեզվով)

```
//PHTSolver.h

#ifndef _COMP_PHTSOLVER_
#define _COMP_PHTSOLVER_

#define MAX_LEN    16
#define OP_XOR     0
#define OP_ADD     1
#define OP_EXP     2
#define OP_LOG     3

extern int DefaultRoundsCount ;
extern int Default_EXP_LOG_Transform_Root ;
extern int DefaultPHTTransformDepth ;
extern int Default_XOR_ADD_Define_Vector[2][MAX_LEN] ;
extern int Default_EXP_LOG_Define_Vector[MAX_LEN] ;
extern int DefaultPHTLines[MAX_LEN] ;
extern int DefaultPHT2Lines[MAX_LEN];

void SubVectors(const int* pVectorA,const int* pVectorB,int* pVectorC,int iVectorLength);
void AddVectors(const int* pVectorA,const int* pVectorB,int* pVectorC,int iVectorLength);
void MulNumToVector(int iNum,const int* pVectorA,int* pVectorB,int iVectorLength);

#define EC_NON_ZERO 1
#define EC_ZERO 2
#define EC_GREATE_THAN_ZERO 3
#define EC_LESS_THAN_ZERO 4

int GetVectorElementCount(int iCountParam,const int* pVector,int iVectorLength);
int CalcCoefMatrix(const int* pVectorPHTLines,int iPHTDepth,int* pMatrix);
int CalcInvCoefMatrix(const int* pVectorPHTLines,const int* pVectorPHTLines2, const int* pVectorPHTLines3,
const int* pVectorPHTLines4,int iPHTDepth,int* pMatrix);
int GetNextOdd(unsigned char* v, int start,int end);

#endif _COMP_PHTSOLVER_

//PHTSolver.cpp

#include "stdafx.h"
#include "stdio.h"
#include "PHTSolver.h"
#include <memory.h>

int DefaultRoundsCount = 5 ;
int Default_EXP_LOG_Transform_Root = 45 ;
int DefaultPHTTransformDepth = 4 ;
int Default_XOR_ADD_Define_Vector[2][MAX_LEN] =
      {
            {OP_XOR,OP_XOR,OP_XOR,OP_XOR,OP_ADD,OP_ADD,OP_ADD,OP_ADD,OP_XOR,OP_XOR,OP_XOR,OP_XOR,OP_ADD,OP_A
            DD,OP_ADD,OP_ADD},
            {OP_ADD,OP_ADD,OP_ADD,OP_ADD,OP_XOR,OP_XOR,OP_XOR,OP_XOR,OP_ADD,OP_ADD,OP_ADD,OP_ADD,OP_XOR,OP_X
            OR,OP_XOR,OP_XOR}
      };

int Default_EXP_LOG_Define_Vector[MAX_LEN] =
      {
            OP_EXP,OP_EXP,OP_EXP,OP_EXP,OP_LOG,OP_LOG,OP_LOG,OP_LOG,OP_EXP,OP_EXP,OP_EXP,OP_EXP,OP_LOG,OP_LO
            G,OP_LOG,OP_LOG
      };

//Modified SAFER+ Shuffling
int DefaultPHT2Lines[MAX_LEN] =  {7,  12,  9,  14,  5,  8,  13,  10,  11,  4,  3,  6,  15,  2,  1,  16};

void SubVectors(const int* pVectorA,const int* pVectorB,int* pVectorC,int iVectorLength)
{
      int i ;
      for ( i = 0; i < iVectorLength; i++)
            pVectorC[i] = ( pVectorA[i] - pVectorB[i] ) % 256 ;
```

```
}
void AddVectors(const int* pVectorA,const int* pVectorB,int* pVectorC,int iVectorLength)
{
	int i ;
	for ( i = 0; i < iVectorLength; i++)
		pVectorC[i] = ( pVectorA[i] + pVectorB[i] ) % 256 ;
}
void MulNumToVector(int iNum,const int* pVectorA,int* pVectorB,int iVectorLength)
{
	int i ;
	for ( i = 0; i < iVectorLength; i++)
		pVectorB[i] = ( iNum * pVectorA[i] ) % 256 ;
}
#define EC_NON_ZERO 1
#define EC_ZERO 2
#define EC_GREATE_THAN_ZERO 3
#define EC_LESS_THAN_ZERO 4

int	GetVectorElementCount(int iCountParam,const int* pVector,int iVectorLength)
{
	int iRet = 0 ;
	int i = 0 ;
	switch ( iCountParam )
	{
		case EC_NON_ZERO :
			for ( i = 0; i < iVectorLength; i++)
			{
				if ( pVector[i] != 0 ) iRet++ ;
			}
		break;
		case EC_ZERO :
			for ( i = 0; i < iVectorLength; i++)
			{
				if ( pVector[i] == 0 ) iRet++ ;
			}
		break;
		case EC_GREATE_THAN_ZERO :
			for ( i = 0; i < iVectorLength; i++)
			{
				if ( pVector[i] > 0 ) iRet++ ;
			}
		break;
		case EC_LESS_THAN_ZERO :
			for ( i = 0; i < iVectorLength; i++)
			{
				if ( pVector[i] < 0 ) iRet++ ;
			}
		break;
		default:
		break;
	}
	return iRet ;
}
// calculate coefficients matrix from given PHT(lines)
int CalcCoefMatrix(const int* pVectorPHTLines,int iPHTDepth,int* pMatrix)
{
	int  Matrix0[MAX_LEN*MAX_LEN];
	int  Matrix1[MAX_LEN*MAX_LEN];
	int  Vector[MAX_LEN];
	int* pMatrixDest  = Matrix0 ;
	int* pMatrixSrc   = Matrix1 ;
	int* pMatrixTemp  = NULL ;
	int* pVectorTempA = NULL ;
	int* pVectorTempB = NULL ;
	int* pVectorTempC = NULL ;
	int* pVectorTempD = Vector ;
	int i = 0 ;
	int j = 0 ;
	int k = 0 ;

	if( pMatrix == NULL )
		return -1;
	if( pVectorPHTLines == NULL )
		return -1;
	if( 1 > iPHTDepth || iPHTDepth > 256 )
		return -1;
```

```
	memset(pVectorTempD,0,sizeof(int)*MAX_LEN);
	memset(pMatrixDest,0,sizeof(int)*MAX_LEN*MAX_LEN);
	memset(pMatrixSrc,0,sizeof(int)*MAX_LEN*MAX_LEN);
	pMatrixTemp = pMatrixSrc ;
	for ( i = 0; i < MAX_LEN; i++)
	{
		*pMatrixTemp = 1 ;
		pMatrixTemp += (MAX_LEN+1) ;
	}
	for ( k = 0; k < iPHTDepth; k++)
	{
		for ( i = 0, j = 0; i < (int)(MAX_LEN/2); i++, j+=2)
		{
			pVectorTempA = (pMatrixSrc + (*(pVectorPHTLines+j)-1)*MAX_LEN) ;
			pVectorTempB = (pMatrixSrc + (*(pVectorPHTLines+j+1)-1)*MAX_LEN) ;

			pVectorTempC = (pMatrixDest + j*MAX_LEN) ;
			MulNumToVector(2,pVectorTempA,pVectorTempD,MAX_LEN);
			AddVectors(pVectorTempD,pVectorTempB,pVectorTempC,MAX_LEN);

			pVectorTempC = (pMatrixDest + (j+1)*MAX_LEN) ;
			AddVectors(pVectorTempA,pVectorTempB,pVectorTempC,MAX_LEN);
		}
		pMatrixTemp = pMatrixSrc ;
		pMatrixSrc = pMatrixDest ;
		pMatrixDest = pMatrixTemp ;
	}
	memcpy(pMatrix,pMatrixSrc,sizeof(int)*MAX_LEN*MAX_LEN);
	return 1 ;
}
// calculate coefficients of inverse matrix from given PHT(lines)
int CalcInvCoefMatrix(const int* pVectorPHTLines,const int* pVectorPHTLines2, const int* pVectorPHTLines3,
const int* pVectorPHTLines4,int iPHTDepth,int* pMatrix)
{
	int Matrix0[MAX_LEN*MAX_LEN];
	int Matrix1[MAX_LEN*MAX_LEN];
	int Vector[MAX_LEN];
	int* pMatrixDest  = Matrix0 ;
	int* pMatrixSrc   = Matrix1 ;
	int* pMatrixTemp  = NULL ;
	int* pVectorTempA = NULL ;
	int* pVectorTempB = NULL ;
	int* pVectorTempC = NULL ;
	int* pVectorTempD = Vector ;
	int* pVectorTempi1 = NULL ;
	int* pVectorTempi2 = NULL ;
	int i = 0 ;
	int j = 0 ;
	int k = 0 ;

	if( pMatrix == NULL )
		return -1;
	if( pVectorPHTLines == NULL )
		return -1;
	if( 1 > iPHTDepth || iPHTDepth > 256 )
		return -1;

	memset(pVectorTempD,0,sizeof(int)*MAX_LEN);
	memset(pMatrixDest,0,sizeof(int)*MAX_LEN*MAX_LEN);
	memset(pMatrixSrc,0,sizeof(int)*MAX_LEN*MAX_LEN);
	pMatrixTemp = pMatrixSrc ;
	for ( i = 0; i < MAX_LEN; i++)
	{
		*pMatrixTemp = 1 ;
		pMatrixTemp += (MAX_LEN+1) ;
	}
	for ( i = 0, j = 0; i < (int)(MAX_LEN/2); i++, j+=2)
	{
		pVectorTempi1 = (pMatrixSrc + j*MAX_LEN) ;
		pVectorTempi2 = (pMatrixSrc + (j+1)*MAX_LEN) ;
		pVectorTempC = (pMatrixDest + (*(pVectorPHTLines2+j+1)-1)*MAX_LEN) ;
		MulNumToVector(2,pVectorTempi2,pVectorTempD,MAX_LEN);
		SubVectors(pVectorTempD,pVectorTempi1,pVectorTempD,MAX_LEN);
		memcpy(pVectorTempC, pVectorTempD, sizeof(int)*MAX_LEN);
		pVectorTempC = (pMatrixDest + (*(pVectorPHTLines2+j)-1)*MAX_LEN) ;
```

```
            SubVectors(pVectorTempi1,pVectorTempi2,pVectorTempC,MAX_LEN);

        }
        pMatrixTemp = pMatrixSrc ;
        pMatrixSrc = pMatrixDest ;
        pMatrixDest = pMatrixTemp ;
        for ( i = 0, j = 0; i < (int)(MAX_LEN/2); i++, j+=2)
        {
            pVectorTempi1 = (pMatrixSrc + j*MAX_LEN) ;
            pVectorTempi2 = (pMatrixSrc + (j+1)*MAX_LEN) ;

            pVectorTempC = (pMatrixDest + (*(pVectorPHTLines+j+1)-1)*MAX_LEN) ;
            MulNumToVector(2,pVectorTempi2,pVectorTempD,MAX_LEN);
            SubVectors(pVectorTempD,pVectorTempi1,pVectorTempD,MAX_LEN);
            memcpy(pVectorTempC, pVectorTempD, sizeof(int)*MAX_LEN);

            pVectorTempC = (pMatrixDest + (*(pVectorPHTLines+j)-1)*MAX_LEN) ;
            SubVectors(pVectorTempi1,pVectorTempi2,pVectorTempC,MAX_LEN);

        }
        pMatrixTemp = pMatrixSrc ;
        pMatrixSrc = pMatrixDest ;
        pMatrixDest = pMatrixTemp ;

        for ( i = 0, j = 0; i < (int)(MAX_LEN/2); i++, j+=2)
        {
            pVectorTempi1 = (pMatrixSrc + j*MAX_LEN) ;
            pVectorTempi2 = (pMatrixSrc + (j+1)*MAX_LEN) ;

            pVectorTempC = (pMatrixDest + (*(pVectorPHTLines3+j+1)-1)*MAX_LEN) ;
            MulNumToVector(2,pVectorTempi2,pVectorTempD,MAX_LEN);
            SubVectors(pVectorTempD,pVectorTempi1,pVectorTempD,MAX_LEN);
            memcpy(pVectorTempC, pVectorTempD, sizeof(int)*MAX_LEN);

            pVectorTempC = (pMatrixDest + (*(pVectorPHTLines3+j)-1)*MAX_LEN) ;
            SubVectors(pVectorTempi1,pVectorTempi2,pVectorTempC,MAX_LEN);

        }
        pMatrixTemp = pMatrixSrc ;
        pMatrixSrc = pMatrixDest ;
        pMatrixDest = pMatrixTemp ;

        for ( i = 0, j = 0; i < (int)(MAX_LEN/2); i++, j+=2)
        {
            pVectorTempi1 = (pMatrixSrc + j*MAX_LEN) ;
            pVectorTempi2 = (pMatrixSrc + (j+1)*MAX_LEN) ;

            pVectorTempC = (pMatrixDest + (*(pVectorPHTLines4+j+1)-1)*MAX_LEN) ;
            MulNumToVector(2,pVectorTempi2,pVectorTempD,MAX_LEN);
            SubVectors(pVectorTempD,pVectorTempi1,pVectorTempD,MAX_LEN);
            memcpy(pVectorTempC, pVectorTempD, sizeof(int)*MAX_LEN);

            pVectorTempC = (pMatrixDest + (*(pVectorPHTLines4+j)-1)*MAX_LEN) ;
            SubVectors(pVectorTempi1,pVectorTempi2,pVectorTempC,MAX_LEN);
        }
        pMatrixTemp = pMatrixSrc ;
        pMatrixSrc = pMatrixDest ;
        pMatrixDest = pMatrixTemp ;
        memcpy(pMatrix,pMatrixSrc,sizeof(int)*MAX_LEN*MAX_LEN);
        return 1 ;
}

// Solve.cpp

#include "stdafx.h"
#include "SaferCryptoAnalyzerIIDoc.h"
#include "solve.h"
#include "PHTSolver.h"

#define SPP_EXP 1
#define SPP_LOG 0

unsigned short spp_shuffling0[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
unsigned short spp_shuffling1[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
uchar encPHT2Matrix[16][16];
uchar decPHT2Matrix[16][16];
```

```
#define sh(i) spp_shuffling0[i]
unsigned short spp_explog[16] = {SPP_EXP,SPP_EXP,SPP_EXP,SPP_EXP,SPP_LOG,SPP_LOG,SPP_LOG,SPP_LOG,
                                 SPP_EXP,SPP_EXP,SPP_EXP,SPP_EXP,SPP_LOG,SPP_LOG,SPP_LOG,SPP_LOG};

int initCycle(unsigned short* curCycleState,
                     unsigned short minValue,
                     unsigned short maxValue,
                     unsigned short iCyclesCount,
                     unsigned short allVariants)
{
      unsigned int i;

      ASSERT( minValue <= maxValue ) ;
      if ( !allVariants ){
            ASSERT( minValue + iCyclesCount - 1 <= maxValue);
            for ( i = 0; i < iCyclesCount; i++)
                  curCycleState[i] = minValue + i;
      } else {
            for ( i = 0; i < iCyclesCount; i++)
                  curCycleState[i] = minValue;
      }
      curCycleState[iCyclesCount-1]--;

      return 1;
}
int stepCycle(unsigned short* curCycleState,
                     unsigned short minValue,
                     unsigned short maxValue,
                     unsigned short iCyclesCount,
                     unsigned short allVariants)
{
      int currCycle = iCyclesCount - 1;

      ASSERT( minValue <= maxValue );

      if ( !allVariants ){
            ASSERT( minValue + iCyclesCount - 1 <= maxValue);
            while ( currCycle >= 0 ){
                  curCycleState[currCycle]++;
                  if ( curCycleState[currCycle] > (maxValue - (iCyclesCount-currCycle-1)) ){
                        currCycle-- ;
                  } else {
                        currCycle++;
                        while ( currCycle < iCyclesCount ){
                              curCycleState[currCycle] = curCycleState[currCycle-1]+1;
                              currCycle++;
                        }
                        return 1;
                  }
            }
            return 0;
      } else {
            while ( currCycle >= 0 ){
                  curCycleState[currCycle]++;
                  if ( curCycleState[currCycle] > maxValue ){
                        curCycleState[currCycle] = minValue;
                        currCycle--;
                  } else {
                        return 1;
                  }
            }
      }
      return 0;
}
#define     decPHT2_0(i1,i2,o1,o2)\
{\
      buff[o2] = 2*in[i2]-in[i1];\
      buff[o1] = in[i1]-in[i2];\
}
#define     decPHT2_1(i1,i2,o1,o2)\
{\
      out[o2] = 2*buff[i2]-buff[i1];\
      out[o1] = buff[i1]-buff[i2];\
}
#define     decPHT2_2(i1,i2,o1,o2)\
```

```
{\
	buff[o2] = 2*out[i2]-out[i1];\
	buff[o1] = out[i1]-out[i2];\
}
#define	decPHT2_3(i1,i2,o1,o2)\
{\
	out[o2] = 2*buff[i2]-buff[i1];\
	out[o1] = buff[i1]-buff[i2];\
}
void decryption_matrix(unsigned char* in, unsigned char* out)
{
	unsigned char buff[16];

		decPHT2_0(0, 1, sh( 0),sh( 1));
		decPHT2_0(2, 3, sh( 2),sh( 3));
		decPHT2_0(4, 5, sh( 4),sh( 5));
		decPHT2_0(6, 7, sh( 6),sh( 7));
		decPHT2_0(8, 9, sh( 8),sh( 9));
		decPHT2_0(10, 11, sh(10),sh(11));
		decPHT2_0(12, 13, sh(12),sh(13));
		decPHT2_0(14, 15, sh(14),sh(15));

		decPHT2_1(0, 1, sh( 0),sh( 1));
		decPHT2_1(2, 3, sh( 2),sh( 3));
		decPHT2_1(4, 5, sh( 4),sh( 5));
		decPHT2_1(6, 7, sh( 6),sh( 7));
		decPHT2_1(8, 9, sh( 8),sh( 9));
		decPHT2_1(10, 11, sh(10),sh(11));
		decPHT2_1(12, 13, sh(12),sh(13));
		decPHT2_1(14, 15, sh(14),sh(15));

		decPHT2_2(0, 1, sh( 0),sh( 1));
		decPHT2_2(2, 3, sh( 2),sh( 3));
		decPHT2_2(4, 5, sh( 4),sh( 5));
		decPHT2_2(6, 7, sh( 6),sh( 7));
		decPHT2_2(8, 9, sh( 8),sh( 9));
		decPHT2_2(10, 11, sh(10),sh(11));
		decPHT2_2(12, 13, sh(12),sh(13));
		decPHT2_2(14, 15, sh(14),sh(15));

		decPHT2_3(0, 1, sh( 0),sh( 1));
		decPHT2_3(2, 3, sh( 2),sh( 3));
		decPHT2_3(4, 5, sh( 4),sh( 5));
		decPHT2_3(6, 7, sh( 6),sh( 7));
		decPHT2_3(8, 9, sh( 8),sh( 9));
		decPHT2_3(10, 11, sh(10),sh(11));
		decPHT2_3(12, 13, sh(12),sh(13));
		decPHT2_3(14, 15, sh(14),sh(15));
}
#define	encPHT2_0(i1,i2,o1,o2)\
{\
	buff[o2] = in[i1]+in[i2];\
	buff[o1] = buff[o2]+in[i1];\
}
#define	encPHT2_1(i1,i2,o1,o2)\
{\
	out[o2] = buff[i1]+buff[i2];\
	out[o1] = out[o2]+buff[i1];\
}
#define	encPHT2_2(i1,i2,o1,o2)\
{\
	buff[o2] = out[i1]+out[i2];\
	buff[o1] = buff[o2]+out[i1];\
}
void encryption_matrix(unsigned char* in, unsigned char* out)
{
	unsigned char buff[16];

		encPHT2_0(sh( 0),sh( 1), 0, 1);
		encPHT2_0(sh( 2),sh( 3), 2, 3);
		encPHT2_0(sh( 4),sh( 5), 4, 5);
		encPHT2_0(sh( 6),sh( 7), 6, 7);
		encPHT2_0(sh( 8),sh( 9), 8, 9);
		encPHT2_0(sh(10),sh(11),10,11);
		encPHT2_0(sh(12),sh(13),12,13);
		encPHT2_0(sh(14),sh(15),14,15);
```

```
		encPHT2_1(sh( 0),sh( 1), 0, 1);
		encPHT2_1(sh( 2),sh( 3), 2, 3);
		encPHT2_1(sh( 4),sh( 5), 4, 5);
		encPHT2_1(sh( 6),sh( 7), 6, 7);
		encPHT2_1(sh( 8),sh( 9), 8, 9);
		encPHT2_1(sh(10),sh(11),10,11);
		encPHT2_1(sh(12),sh(13),12,13);
		encPHT2_1(sh(14),sh(15),14,15);

		encPHT2_2(sh( 0),sh( 1), 0, 1);
		encPHT2_2(sh( 2),sh( 3), 2, 3);
		encPHT2_2(sh( 4),sh( 5), 4, 5);
		encPHT2_2(sh( 6),sh( 7), 6, 7);
		encPHT2_2(sh( 8),sh( 9), 8, 9);
		encPHT2_2(sh(10),sh(11),10,11);
		encPHT2_2(sh(12),sh(13),12,13);
		encPHT2_2(sh(14),sh(15),14,15);

		encPHT2_1(sh( 0),sh( 1), 0, 1);
		encPHT2_1(sh( 2),sh( 3), 2, 3);
		encPHT2_1(sh( 4),sh( 5), 4, 5);
		encPHT2_1(sh( 6),sh( 7), 6, 7);
		encPHT2_1(sh( 8),sh( 9), 8, 9);
		encPHT2_1(sh(10),sh(11),10,11);
		encPHT2_1(sh(12),sh(13),12,13);
		encPHT2_1(sh(14),sh(15),14,15);
}
BOOL isIn(unsigned char iNum,unsigned char* arrNum,int len)
{
	for ( int i = 0; i < len; i++)
		if ( iNum == arrNum[i] )
			return TRUE;
	return FALSE;
}
int GetNextOdd(unsigned char* v, int start,int end)
{
	for ( int i = start; i < end; i++ )
		if ( v[i] & 1 )
			return i;
	return -1;
}
unsigned char Inv[256];
unsigned char GetInverse(unsigned char ch)
{
	for ( int i = 0; i < 256; i++)
		if ( (unsigned char)(i * ch) == 1 )
			return i;
	return 0;
}
void FillInvArray()
{
	int i ;
	Inv[0] = 0;
	for ( i = 1; i < 256; i++ ) Inv[i] = GetInverse(i);
	Inv[128] = 0;
}
void AddVectors(unsigned char* vres, unsigned char* v1, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++)  vres[i] = v1[i] + v2[i];
}
void SubVectors(unsigned char* vres, unsigned char* v1, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++)  vres[i] = v1[i] - v2[i];
}
void AddVectorsX(unsigned char* v1, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++)  v1[i] += v2[i];
}
void SubVectorsX(unsigned char* v1, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++)  v2[i] =v1[i] - v2[i];
}
void MulNumToVector(unsigned char* vres, unsigned char num, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++) vres[i] = num * v2[i];
```

```
}
void MulNumToVectorX(unsigned char num, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++) v2[i] = num * v2[i];
}
void DivVectorToNumX(unsigned char num, unsigned char* v2, unsigned char len)
{
	for ( int i = 0; i < len; i++) v2[i] = v2[i]/num;
}
void InsertValue(unsigned char* vres, unsigned char* v1, unsigned char* v2, unsigned char pos, unsigned char
len)
{
	unsigned char koef = v1[pos];
	v1[pos] = 0;
	for ( int i = 0; i < len; i++) vres[i] = koef*v2[i] + v1[i];
}
void InsertValueQ(unsigned char* validInputs,unsigned char* vres, unsigned char* v1, unsigned char* v2,
unsigned char pos, unsigned char len)
{
	unsigned char koef = v1[validInputs[pos]];
	v1[pos] = 0;
	for ( int i = 0; i < len; i++) vres[validInputs[i]] = koef*v2[i] + v1[validInputs[i]];
}
void InsertValueX(unsigned char* v1, unsigned char* v2, unsigned char pos, unsigned char len)
{
	unsigned char koef = v1[pos];
	v1[pos] = 0;
	for ( int i = 0; i < len; i++) v1[i] += koef*v2[i];
}
void InsertValueQX(unsigned char* validInputs, unsigned char* v1, unsigned char* v2, unsigned char pos,
unsigned char len)
{
	unsigned char koef = v1[validInputs[pos]];
	v1[pos] = 0;
	for ( int i = 0; i < len; i++) v1[validInputs[i]] += koef*v2[i];
}
int GetNextNonZero(unsigned char* v, int start,int end)
{
	for ( int i = start; i < end; i++ )
		if ( v[i] ) return i;
	return -1;
}
void NegVector( unsigned char* vector, int start, int end)
{
	for ( int i = start; i < end; i++) vector[i] = -vector[i];
}
unsigned char VectorWeight( unsigned char* vector, int len)
{
	unsigned char w = 0;
	for ( int i = 0; i < len; i++)
		if ( vector[i] ) w++;
	return w;
}
int SolveAnEquation(CResult* solution,
                                unsigned char*  matrix,
                                unsigned short* validVars,
                                unsigned char  validVarsCount,
                                unsigned short* validEquations,
                                unsigned char  validEquationsCount)
{
	unsigned char modifmatrix[16][16];
	unsigned char *pEq[16];
	unsigned char *pTmp;
	unsigned char vtemp1[16];
	unsigned char w, minw, iminwEq;
	int i, j, iOddPos, iOddsCnt;
	int iLastOddLine, iLastOddLine2, iLastOddLine4, iLastOddLine8;
	BOOL b128;
	ASSERT( validVarsCount > validEquationsCount );

	for ( i = 0; i < validEquationsCount; i++){
		pEq[i] = modifmatrix[i];
		for ( j = 0; j < validVarsCount; j++){
			modifmatrix[i][j] = matrix[validVars[j]*16+validEquations[i]];
		}
	}
```

```
	for ( i = 0; i < validEquationsCount; i++){
		if ( VectorWeight(modifmatrix[i], validVarsCount) < validVarsCount-validEquationsCount){
			return -1; // no solution
		}
	}
	minw = 100;
	iOddsCnt = 0;
	iLastOddLine = -1;
	for ( i = 0; i < validEquationsCount; i++){
		iOddPos = -1;
		iOddPos = GetNextOdd( pEq[i], iOddPos+1, validVarsCount);
		if (iOddPos != -1){
			if ( i-1 != iLastOddLine ){
				pTmp = pEq[iLastOddLine+1];
				pEq[iLastOddLine+1] = pEq[i];
				pEq[i] = pTmp;
				i = iLastOddLine+1;
			}
			for ( j = i+1; j < validEquationsCount; j++){
				MulNumToVector(vtemp1, Inv[pEq[i][iOddPos]]*pEq[j][iOddPos], pEq[i],
validVarsCount);
				SubVectorsX(vtemp1, pEq[j], validVarsCount);
				if ( (w = VectorWeight(pEq[j], validVarsCount)) < validVarsCount-
validEquationsCount+1){
					// no solution
					return -1;
					} else {
						if ( w < minw ){
							minw = w;
							// ;) allways iminwEq == validEquationsCount-1
							iminwEq = i;
						}
					}
			}
			iOddsCnt++;
			iLastOddLine = i;
			if ( iLastOddLine == validEquationsCount-2 )
				break;
		}
	}
	iLastOddLine2 = validEquationsCount;
	if ( iOddsCnt < validEquationsCount-1 ){
		// find non odd vectors and divide on 2 ??????
		iLastOddLine2 = iLastOddLine+1;
		while ( GetNextOdd( pEq[iLastOddLine2], 0, validVarsCount) == -1 ){
			for ( i = iLastOddLine2; i < validEquationsCount; i++)
				DivVectorToNumX(2, pEq[i], validVarsCount);
		}
		// solve equations
		iLastOddLine = iLastOddLine2-1;
		for ( i = iLastOddLine2; i < validEquationsCount; i++){
			iOddPos = -1;
			iOddPos = GetNextOdd( pEq[i], iOddPos+1, validVarsCount);
			if (iOddPos != -1){
				if ( i-1 != iLastOddLine ){
					pTmp = pEq[iLastOddLine+1];
					pEq[iLastOddLine+1] = pEq[i];
					pEq[i] = pTmp;
					i = iLastOddLine+1;
				}
				for ( j = i+1; j < validEquationsCount; j++){
					MulNumToVector(vtemp1, Inv[pEq[i][iOddPos]]*pEq[j][iOddPos], pEq[i],
validVarsCount);
					SubVectorsX(vtemp1, pEq[j], validVarsCount);
					if ( VectorWeight(pEq[j], validVarsCount) < validVarsCount-
validEquationsCount+1){
						return -1; // no solution
					} else {
						if ( w < minw ){
							minw = w;
							// ;) allways iminwEq == validEquationsCount-1
							iminwEq = i;
						}
					}
				}
				iOddsCnt++;
```

```
                        iLastOddLine = i;
                        if ( iLastOddLine == validEquationsCount-2 )
                              break;
                  }
            }
      }
      iLastOddLine4 = validEquationsCount;
      if ( iOddsCnt < validEquationsCount-1 ){
            // find non odd vectors and divide on 2 ??????
            iLastOddLine4 = iLastOddLine+1;
            while ( GetNextOdd( pEq[iLastOddLine4], 0, validVarsCount) == -1 ){
                  for ( i = iLastOddLine4; i < validEquationsCount; i++)
                        DivVectorToNumX(2, pEq[i], validVarsCount);
            }
            // solve equations
            iLastOddLine = iLastOddLine4-1;
            for ( i = iLastOddLine4; i < validEquationsCount; i++){
                  iOddPos = -1;
                  iOddPos = GetNextOdd( pEq[i], iOddPos+1, validVarsCount);
                  if (iOddPos != -1){
                        if ( i-1 != iLastOddLine ){
                              pTmp = pEq[iLastOddLine+1];
                              pEq[iLastOddLine+1] = pEq[i];
                              pEq[i] = pTmp;
                              i = iLastOddLine+1;
                        }
                        for ( j = i+1; j < validEquationsCount; j++){
                              MulNumToVector(vtemp1, Inv[pEq[i][iOddPos]]*pEq[j][iOddPos], pEq[i],
validVarsCount);
                              SubVectorsX(vtemp1, pEq[j], validVarsCount);
                              if ( VectorWeight(pEq[j], validVarsCount) < validVarsCount-
validEquationsCount+1){
                                    return -1; // no solution
                              } else {
                                    if ( w < minw ){
                                          minw = w;
                                          // ;) allways iminwEq == validEquationsCount-1
                                          iminwEq = i;
                                    }
                              }
                        }
                        iOddsCnt++;
                        iLastOddLine = i;
                        if ( iLastOddLine == validEquationsCount-2 )
                              break;
                  }
            }
      }
      iLastOddLine8 = validEquationsCount;
      if ( iOddsCnt < validEquationsCount-1 ){
            // find non odd vectors and divide on 2 ??????
            iLastOddLine8 = iLastOddLine+1;
            while ( GetNextOdd( pEq[iLastOddLine8], 0, validVarsCount) == -1 ){
                  for ( i = iLastOddLine8; i < validEquationsCount; i++){
                        DivVectorToNumX(2, pEq[i], validVarsCount);
                  }
            }
            // solve equations
            iLastOddLine = iLastOddLine8-1;
            for ( i = iLastOddLine8; i < validEquationsCount; i++){
                  iOddPos = -1;
                  iOddPos = GetNextOdd( pEq[i], iOddPos+1, validVarsCount);
                  if (iOddPos != -1){
                        if ( i-1 != iLastOddLine ){
                              pTmp = pEq[iLastOddLine+1];
                              pEq[iLastOddLine+1] = pEq[i];
                              pEq[i] = pTmp;
                              i = iLastOddLine+1;
                        }
                        for ( j = i+1; j < validEquationsCount; j++){
                              MulNumToVector(vtemp1, Inv[pEq[i][iOddPos]]*pEq[j][iOddPos], pEq[i],
validVarsCount);
                              SubVectorsX(vtemp1, pEq[j], validVarsCount);
                              if ( VectorWeight(pEq[j], validVarsCount) < validVarsCount-
validEquationsCount+1){
                                    return -1; // no solution
```

```
                                } else {
                                        if ( w < minw ){
                                                minw = w;
                                                // ;) allways iminwEq == validEquationsCount-1
                                                iminwEq = i;
                                        }
                                }
                        }
                        iOddsCnt++;
                        iLastOddLine = i;
                        if ( iLastOddLine == validEquationsCount-2 )
                                break;
                }
        }
    }
    b128 = FALSE;
    for ( i = validEquationsCount-1; i >= 0; i--){
        iOddPos = GetNextOdd( pEq[i], 0, validVarsCount);
        if ( iOddPos != -1 ){
                MulNumToVector(&solution->m_varInput[i][2], Inv[pEq[i][iOddPos]], pEq[i], validVarsCount);
                solution->m_varInput[i][0] = iOddPos;//(unsigned char)validVars[iOddPos];
                solution->m_varInput[i][iOddPos+2] = 0;
                if ( i >= iLastOddLine2 || b128 ){
                        solution->m_varInput[i][1] = 128;
                        b128 = FALSE;
                }
                else
                        solution->m_varInput[i][1] = 0;
                NegVector( &solution->m_varInput[i][2], 0, validVarsCount);
                for ( j = i-1; j >= 0; j--){
                        InsertValueX( pEq[j], &solution->m_varInput[i][2], iOddPos, validVarsCount);
                }
                if ( VectorWeight(&solution->m_varInput[i][2], validVarsCount) < validVarsCount-
validEquationsCount){
                        return -1; // no solution
                }
        } else {
                DivVectorToNumX(2, pEq[i], validVarsCount);
                b128 = TRUE;
                i++;
        }
    }
    solution->m_iInputWeight = validVarsCount;
    solution->m_ivarCount = validVarsCount - validEquationsCount;
    for ( i = 0; i < validVarsCount; i++){
        solution->m_nzInputBytes[i] = (unsigned char)validVars[i];
    }
    return 0;
}
unsigned char GetNonzeroOutputs(CResult* sol, unsigned char*  matrix)
{
    int i, j;
    int w = 0;

    for ( i = 0; i < 16; i++){
        for ( j = 0; j < sol->m_iInputWeight; j++)
                sol->m_varOutput[i][j] = matrix[sol->m_nzInputBytes[j]*16+i];
    }
    for ( i = 0; i < sol->m_iInputWeight-sol->m_ivarCount; i++){
        for ( j = 0; j < 16; j++){
                InsertValueX( sol->m_varOutput[j], &sol->m_varInput[i][2], sol->m_varInput[i][0], sol-
>m_iInputWeight);
        }
    }
    for ( j = 0; j < 16; j++){

        if ( VectorWeight(sol->m_varOutput[j], sol->m_iInputWeight) ){
                sol->m_nzOutputBytes[w] = j;
                w++;
        }
    }
    sol->m_iOutputWeight = w;
    return w;
}
```

Սա կոդի մի մասն է, ամբողջական կոդը և “exe” ֆայլը կոշտ սկավառակի վրա են: