Универзитет у Београду
Математички факултет

Мастер рад

Криптоанализа шифре KASUMI

Аутор:
Кристина Станојевић

Ментор:
проф. др Миодраг Живковић

Чланови комисије:

проф. др Саша Малков
доц. др Нина Радојичић Матић

29. септембар 2022.

Криптоанализа шифре KASUMI

Апстракт

У последњих неколико деценија тајност већег дела *GSM* комуникација штићена је проточним шифрама *A5/1* и *A5/2*, за које је више аутора показало да су криптолошки слабе. Ове шифре су замењене новим алгоритмима, *A5/3* и *A5/4*, заснованим на блоковској шифри *KASUMI*. У раду се анализира тзв. сендвич напад са повезаним кључевима на алгоритам *KASUMI*. Овај напад не угрожава сигурност комуникације у реалним условима. Поред теоријског дела циљ рада је програмски реализовати препознавач, док се напад на комплетан алгоритам *KASUMI* (или на варијанту алгоритма са смањеним бројем рунди) разматра само теоријски.

Садржај

Листа енглеских скраћеница

GSM	Глобални систем за мобилну комуникацију *(**G**lobal **S**ystem for **M**obile **C**ommunications)*
ETSI	Европски институт за телекомуникацијске стандарде *(**E**uropean **T**elecommunications **S**tandards **I**nstitute)*
3GPP	Трећа геренарија партнерског пројекта *(**3**rd **G**eneration **P**artnership **P**roject)*
ISDN	Интернационални стандард дигиталне телекомуникације *(**I**ntegrated **S**ervices **D**igital **N**erwork)*
DSL	Дигитална претплатничка линија *(**D**igital **S**ubscriber **L**ine)*
MS	Мобила станица *(**M**obile **S**tation)*
BS	Базна станица *(**B**ase **S**tation)*
BSC	Управљачка јединица базне станице *(**B**ase **S**tation **C**ontroller)*
MSC	Мобилни комутациони центар *(**M**obile **S**witching **C**enter)*
HLR	Регистар властитих претплатника *(**H**ome **L**ocation **R**egister)*
VLR	Регистар гостујућих претплатника *(**V**isitor **L**ocation **R**egister)*
LAI	Идентификациони број подручја *(**L**ocation **A**rea **I**dentity)*
AuC	Ауторизациони центар *(**A**uthentication **C**entre)*
EIR	Регистар идентитета уређаја *(**E**quipment **I**dentity **R**egister)*
PSTN	Јавна телефонска мрежа *(**P**ublic **S**witched **T**elephone **N**etwork)*
SIM	Модул за идентитет претплатника *(**S**ubscriber **I**dentity **M**odule)*
IMSI	Интернационални идентификациони број корисника *(**I**nternational **M**obile **S**ubscriber **I**dentity)*
MCC	Број кода државе

	*(**M**obile **C**ountry **C**ode)*
MNC	Број кода мобилне мреже
	*(**M**obile **N**etwork **C**ode)*
MSIN	Идентификациони број корисника
	*(**M**obile **S**ubscriber **I**dentification **N**umber)*
MSISND	Интернационални ISDN број мобилног корисника
	*(**M**obile **S**ubscriber **ISDN**)*
UMTS	Универзални мобилни телекомуникациони систем
	*(**U**niversal **M**obile **T**elecommunications **S**ystem)*
ECB	Директна употреба блок-шифре
	*(**E**lectronic **C**ode **B**ook)*
CBC	Уланчавање шифрованих блокова
	*(**C**ipher **B**lock **C**haining)*
CFB	Шифровање помоћу повратне спреге
	*(**C**ipher **F**feed**B**ack)*
OFB	Формирање излаза у повратној спрези
	*(**O**utput **F**feed**B**ack)*
SPN	Шифра супституције и пермутације
	*(**S**ubstitution-**P**ermutation **N**etwork Cipher)*

Глава 1

Увод

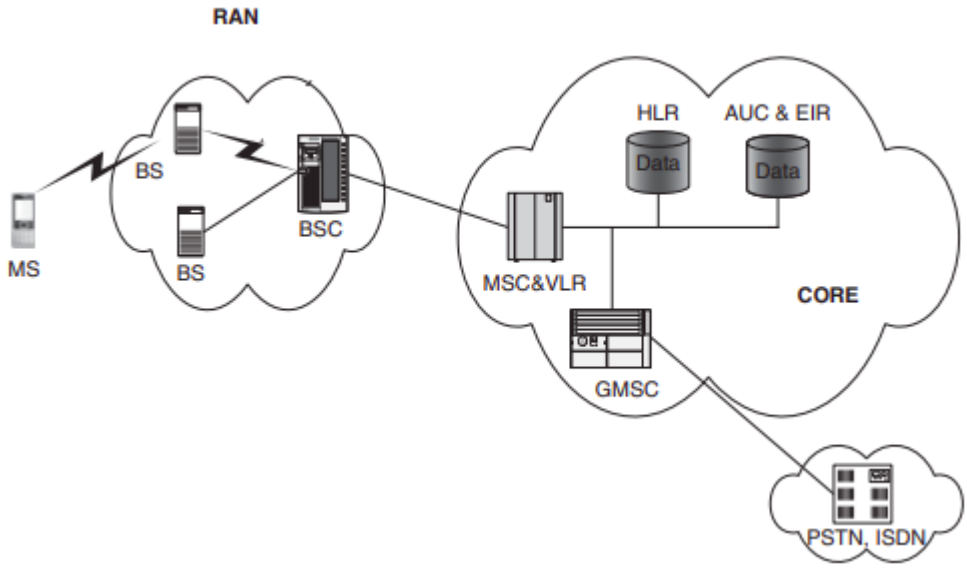
1.1 Глобални систем за мобилну комуникацију (*GSM*)

Током 1980-тих почиње развитак *GSM* технологије. Најважније стандарде развио је Европски институт за телекомуникацијске стандарде (*ETSI*), а касније и *3GPP*[1]. Потреба да се креира дигитални систем који је квалитетнији од аналогног система, али јефтин за масовну производњу, довела је до развитка стандарда у систему за пренос података, гласа и сигнала дигиталном линијом (*ISDN*) кроз бакарну жицу (што је чини јефтином, а корисницима додатно даје могућност приступа интернету широког пропусног опсега). Касније је тај систем еволуирао у *DSL* или кабловске модеме, мада остаје да постоји као алтернатива. *GSM* преставља стандард који описује протоколе мобилне мреже, први који разматра безбедност и могуће претње систему. Омогућене су и услуге слања порука и међународни роминг [7].

На слици 1.2 представљен је пут поруке која се шаље са једног краја и пристиже до другог. Пошиљалац поруку преводи у сигнал како би је одређеним каналима (у телекомуникацији жицом) слао кроз систем. Пријемник од пристиглог сигнала реконструише разговетну поруку за примаоца. Успут је могуће наићи на сметње у преносу сигнала, шумове, тако и прислушкиваче [14].

Компоненте *GSM* мреже приказане на слици 1.1 су: мобилна станица, односно мобилни телефон (*MS*), је уређај који бежично комуницира са базном станицом (*BS*), која се преко антена, струје и опреме за пренос података, директном линијом или радио везом повезује са управљачком јединицом базне станице (*BSC*). Ове две компоненте заједно управљају радио каналима, кодирају канале и прилагођавају брзину преноса, воде рачуна о аутентификацији, шифровању, мерењу саобраћаја, мерењу улазног сигнала итд. Даље, језгро мреже са којим је контролер повезан је мобилни комутациони центар (*MSC*), који приликом позива тражи најјачи сигнал, односно пут којим ће сигнал (порука) ићи, а такође препознаје локацију мобилног уређаја. Порука се лако прослеђује на одговарајући циљ, јер

[1]Значење енглеских скраћеница које се појављују у раду погледати у делу "Листа енгеских скраћеница".



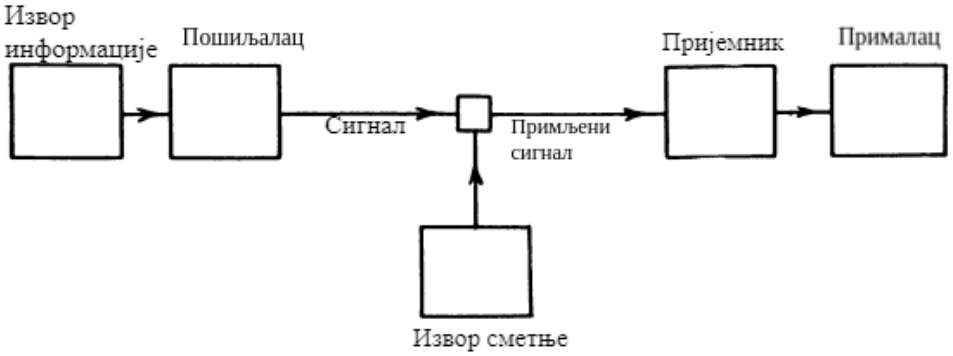
Слика 1.1: Компоненте GSM

је мрежа целуларног типа[2]. Регистар властитих претплатника (*HLR*) и регистар локација гостујућих претплатника (*VLR*) чувају податке о корисницима (њихове профиле и локацију), уводе нове кориснике у мрежу и чувају их за будуће позиве. У целуларној мрежи сваки део територије има свој идентификациони број (*LAI*). Када корисник са својим мобилним уређајем приступа мрежи, *LAI* са његове локације се уписује у *HLR/VLR*. Сваки пут када корисник промени своју локацију ажурира се број *LAI*. Ауторизациони центар (*AuC*), најважнији део безбедносног подсистема, садржи кључеве за аутентификацију корисника и шифровање података. Регистар идентитета уређаја (*EIR*) чува податке о уређајима корисника, који су на пример украдени или надзирани како би могли брзо да се пронађу. Последња компонента *GSM*-а је јавна телефонска мрежа (*PSTN*) која заправо представља скуп свих светских телефонских мрежа које пружају инфраструктуру и услуге телекомуникације, дакле све до сад наведено, додатно и каблове, сателите итд.

1.2 Слабости мреже и напади

Сваки криптографски систем има две главне компоненте: алгоритам и кључ [3]. Сам систем мобилне телефоније подразумева преношење података бежично па се тиме сигурност система знатно отежава. Са друге стране развитак технологије, брз и нагли пораст броја корисника задао је тежак задатак систему, да

[2]Територија је покривена ћелијама. Свака ћелија има одређени фреквентни опсег и једну базну станицу (ћелије са различитих делова територије могу бити везане на исту базну станицу док један део територије (*Location Area – LA*) мора да садржи бар једну базну станицу). Исти фреквенцијски канал се може користити у више ћелија ако су довољно удаљене. Лако се може повећати капацитет мреже.



Слика 1.2: Пут информације

се све то испрати одговарајућом количином каблова и контролом комуникације. Поред техничких решења, једну од слабости система представља и могућа природна непогода која може трајно уништити инфраструктуру.

Оно што комуникациона мрежа треба да пружа корисницима јесте:

1. **поверљивост информација** на путу од пошиљаоца до примаоца поруке,
2. **интегритет** послатих података,
3. **аутентификацију**, како нападач не би могао да се представи као нека друга особа и прикрије свој идентитет и намеру,
4. **доказ о пореклу поруке**, како би примаоц био сигуран од кога она стиже,
5. **поузданост** комуникације и њену заштиту.

Сваки од ових услова мрежа испуњава захваљујући криптографским системима. Развој мобилне мреже *3G* попут *UMTS* гарантују у најмању руку аутентификацију корисника и обезбеђивање поверљивости у току размене података. Криптографски алгоритми имплементирани у различите сврхе заштите корисника мобилне мреже су алгоритми аутентификације *A3*, алгоритми генерисања кључа *A8* и алгоритми шифровања података *A5*. Постоји више верзија ових алгоритама. На пример, код криптографских алгоритма *A5/1* и *A5/2* пронађена је слабост и напад на њих могућ је у реалном времену. Наравно, постоји доста простора за развој чиме се развијају нове верзије ових алгоритама.

Нападач мреже, без обзира на разлог напада, користи се углавном неком од следећих техника за приступ мрежи:

- Не чинећи никакву физичку штету инфраструктури и без слања информација о себи, нападач прислушкује комуникационе канале коришћењем алата за мерење фреквенције. Могућа је такорећи крађа идентитета неке приступне тачке и њена имитација, чиме се такође преузимају пакети послати између корисника.

- Лажирање података, као што је *IP адреса*, омогућава нападачу да обмањивањем корисника представи себе као поузданог члан мреже.

- "Човек у средини"је метод којим се нападач налази између пошиљаоца и примаоца поруке, прислушкује поруку, мења је без знања обе корисничке стране и шаље своју измењену поруку, са неким вирусом и слично. То се постиже тако што се онемогући повезивање на оригиналну приступну тачку, а затим њеном имитацијом се клијент прикључује на лажну приступну тачку и шаље податке. Лажирање базне станице је омогућено јер се аутентификација врши само од мобилног уређаја ка базној станици док обрнути смер идентификовања није имплементиран у систему *GSM*.

- Онемогућавање услуга мреже изазивањем пуцања мреже, тако што нападач шаље поруке које изазивају пад система или преоптерећењем тако што нападач шаље велику количину података. Углавном се изазива пад сајтова неких банака или компанија, чиме се нарушавају и дестабилизују ти системи.

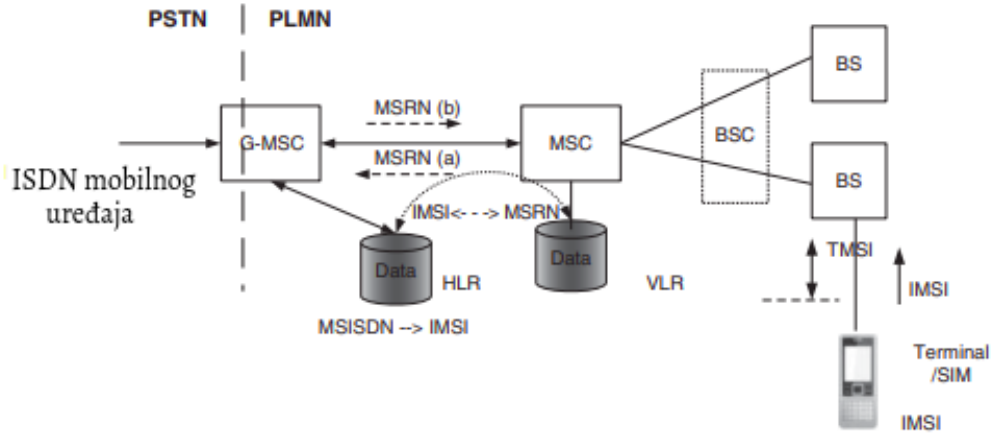
У односу на све ове грубо представљене врсте напада, видимо да нападачи могу имати разне мотиве напада, од преузимања идентитета, прислушкивања порука, уметања вируса, тако и до претње или активизма због неке огорчености. Све ово је наравно подлежно променама, како и сама технологија напредује и како се нуде разна решења за спречавање напада.

Главна идеја одбране система мобилне комуникације је да обезбеди такву сигурност и јачину криптографских алгоритама да се нападачу не исплати напад (пошто се не може гарантовати непостојање пропуста у алгоритмима), тако што му је потребно превише скупих ресурса и времена да напад успе. Подаци којима би приступио би већ застарели и не би имао никакву корист у реалном времену. Стална мотивација за напредак, успех и развитак нових идеја увек постоји.

1.3 *SIM* картица

SIM (*Subscriber Identity Module*) картица је чип на коме се смешта интернационални идентификациони јединствени број корисника (*IMSI*) којим се корисник идентификује на мобилној телефонској мрежи. Мобилни уређај постаје мобилна станица тек када се убаци *SIM* картица у уређај. Ово значи да постојање *SIM* картице омогућава да корисник може користити било који мобилни телефон који садржи све остале потребне могућности за прикључење на локалну мрежу. Главна намена *SIM* картице јесте чување идентификационог броја, тајног кључа (дужине 128 бита) који се креира у току производње картице, као и криптографских алгоритама којима се врши аутентификација корисника помоћу ових бројева. Идентификациони број *IMSI* је максималне дужине 15 цифара и састоји се од: три цифре кода државе (*MCC*), две цифре за код мобилне мреже (*MNS*), највише десет цифара за идентификациони број корисника (*MSIN*).

Сви корисници мобилних уређаја морају бити евидентирани у глобалној мрежи како би били ауторизовани и лоцирани. Приликом позива неке особе бира се број *MSISDN*, који је додељен кориснику. Постојање овог броја је неопходно како би се сачувала тајност броја *IMSI*, због сигурности приликом аутентификације. Ови бројеви се чувају у регистру *HLR*. На слици 1.3 приказан је описан ток мреже и размена наведених јединствених бројева ради аутентификације. Постоји још доста делова мрежног система, али се тиме нећемо бавити у овом раду. [10]

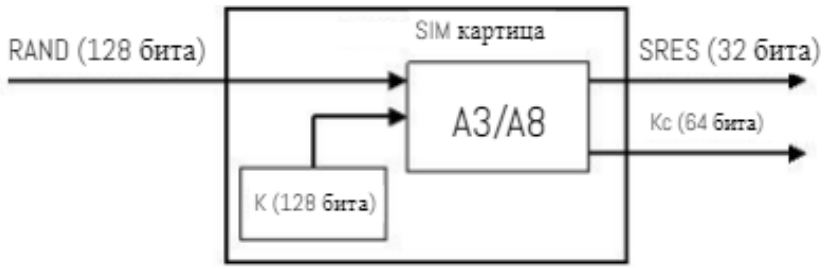


Слика 1.3: Ток мреже

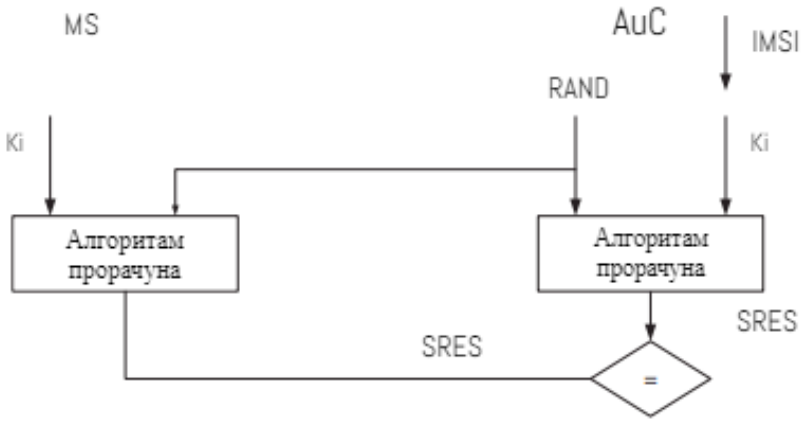
1.4 Аутентификација

Аутентификација представља један од примарних процеса заштите комуникационих мрежа, јер не пушта неповерљиве људе у систем, већ само оне са јасним унапред познатим идентитетом. Шифровање података који се шаљу различитим алгоритмима шифровања представља такође битан процес у комуникационим системима.

SIM картица садржи имплементиран алгоритам *A3* којим се испуњава протокол аутентификације користећи кључ K и број *IMSI* унапред задат картици, што се види на слици 1.4. Кључ K се додатно чува и у центру за аутентификацију *AuC*. Наиме, приликом иницирања позива *AuC* проверава идентификацију мобилног уређаја тако што му шаље случајни 128-битни број и број који добије коришћењем алгоритма аутентификације над кључем K (види слику 1.5, ти бројеви су *RAND* и *SRES*). Мобилна станица користећи *RAND* и K алгоритамом добија свој *SRES* број. Ако се та два *SRES* броја поклапају аутентификација је успешна те се процес слања информација усмерава даље. Дакле, нико сем *AuC* и *SIM* картице нема приступ кључу K.



Слика 1.4: Аутентификација у SIM картици



Слика 1.5: Процес аутентификације

Глава 2

Криптографски алгоритми

2.1 Шифровање података

Како мобилна мрежа поред свих мера заштите не може да гарантује апсолутну сигурност, идеја је шифровати податке који се шаљу кроз мрежу. Порука која се пошаље од тачке А до тачке Б се може шифровати на више начина тако да прислушкивачи не могу разгонетнути текст без познавања кључа који је коришћен приликом шифровања. Симетрични систем користи исти кључ за шифровање и дешифровање поруке, с тим да се А и Б интерно договоре око вредности кључа и не деле је ни са ким како би били сигурни. Постоји и асиметрични систем са јавним кључем где учесници А и Б деле свој кључ јавно, док свој интерни кључ за дешифровање чувају у тајности. У систему *GSM* за шифровање порука користи се алгоритам А5, који има више својих верзија, које су настајале као нека врста надоградње, јер су претходне верзије успешно нападнуте, а напади јавно објављени [9].

Генерално, идеја је да се након успешне аутентификације унутар мобилног уређаја генерише кључ за шифровање користећи алгоритам *A8*, који је попут алгоритма *A3* имплементиран на *SIM* картици. Затим се алгоритмом *A5* из мобилног уређаја добијеним кључем шифрује порука и шаље даље ка базној станици. На другом крају се такође помоћу алгоритма *A5* дешифрују подаци између базне станице и примаоца поруке, након што је утврђен идентитет примаоца. *UMTS* (*Universal Mobile Telecommunications Service*) је систем попут *GSM*-а, с тим да превазилази његове мане. *UMTS* на пример гарантује интегритет података, обезбеђује да се базна станица аутентификује мобилном уређају итд. Овај систем тражи нова решења криптографије унутар мреже још од како су разбијене верзије алгоритма *A5/1* и *A5/2*. Појављује се и *A5/3*, алгоритам заснован на алгоритму *KASUMI*.

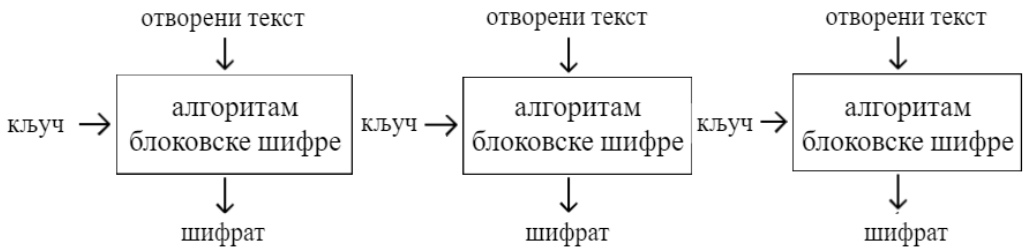
2.2 Блоковске шифре

Криптографија се као област развија великом брзином. Како су информације данас скупе и како се њихова крађа очекује, шифровање података представља кључни корак у развитку било које гране у индустрији. Креирани су разни

алгоритми шифровања за различите потребе. У овом раду фокусираћемо се на блоковске шифре, јер је *KASUMI*, алгоритам о коме ће бити речи, управо такав.

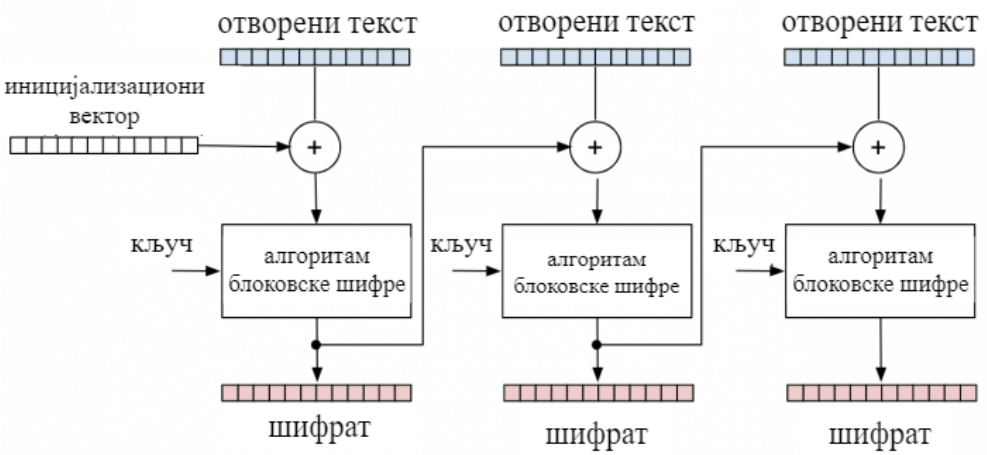
Блоковске шифре су детерминистички алгоритми којима се отворени текст (порука коју треба шифровати) дели на блокове, они се појединачно шифрују и на крају заједно представљају шифрат [15]. Блоковске шифре могу бити:

- *ECB* (Директна употреба блок-шифре) - од блока отвореног текста одговарајућом трансформацијом добија се блок шифроване поруке. Врши се иста трансформација, што доводи до тога да исти текст даје увек исти шифрат, што је потенцијална слабост поступка. Поступак је приказан на слици 2.1.
- *CBC* (Уланчавање шифрованих блокова) - користи се исти принцип као *ECB* с тим да се у трансформацију додаје и иницијализациони вектор (IV). За сваку нову поруку креира се нови IV, који улази у трансформацију првог блока и даје први блок шифрата. Тај шифрат улази у трансформацију са другим блоком отвореног текста и тако даље. Процес је приказан на слици 2.2.
- *CFB* (Шифровање помоћу повратне спреге) - попут *CBC* користи се иницијализациони вектор, с тим да се може шифровати неки низ бајтова, док се у *CBC* шифрују комплетни блокови.
- *OFB* (Формирање излаза у повратној спрези) - ова метода подразумева да се трансформацијом иницијализационог вектора добија низ кључа за шифровање. Потом дати низ кључа и отворени текст сабирањем дају шифрат, видети слику 2.3.

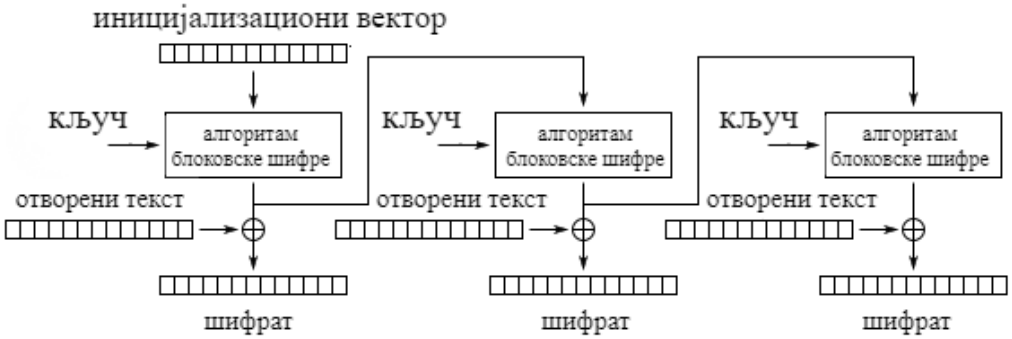


Слика 2.1: Коришћење блоковске шифре методом *ECB*

Веома важан корак представља и бирање кључа у блоковској шифри. Идеја је направити такав кључ да се он не може лако открити. Када би нападач грубом силом желео да погоди вредност кључа дужине k бита, њему би требало 2^k покушаја. Продужавањем кључа за један бит, вероватноћа да ће нападач овом



Слика 2.2: Коришћење блоковске шифре методом *CBC*



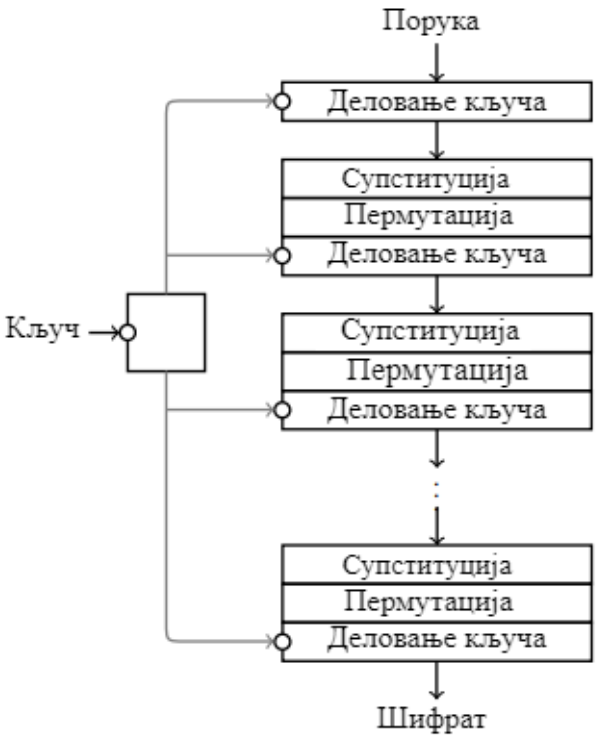
Слика 2.3: Коришћење блоковске шифре методом *OFB*

аматерском методом да разбије кључ се двоструко смањује. Пошто се свака шифра овако може разбити, идеја је да кључ буде довољне дужине, али на пример и да се кључ мења довољно често, тако да нападач нема довољно времена да га открије у међувремену, било којом методом, или да му је потребно превише ресурса, те би му тај процес био скуп и неисплатив. Не постоји правило које каже колика је дужина кључа оптимална како би процес био сигуран, али се представа о томе може стећи на основу табеле 2.1. У криптографским алгоритмима користи се углавном кључ дужине 128 бита.

Године 1975. настаје *DES* (*Data Encryption Standard*), систем шифровања који користи кључ дужине 56 бита, блокове од 64 бита и служи се пермутацијама и заменама битова у току шифровања. Пошто је 90-их разбијен, расписан је конкурс после чега је *DES* наследио алгоритам *AES* (*Advanced Encryption Standard*) са 128-битним кључем. Блоковске шифре користе и супституцију и пермутацију, операције чијом се вишеструком применом безбедност података повећава. Порука која се шифрује дели се у блокове и пролази кроз рунде супституције, пермутације и деловања кључа (користећи "ексклузивно или"(*XOR*), битску операцију) као на слици 2.4.

K бита	број операција	статус	
40	2^{40}	лако за разбијање	не постоји сигурност
64	2^{64}	могуће разбити	веома слаба сигурност
80	2^{80}	тешко изводљиво	сигурно
128	2^{128}	веома сигурно	одлична сигурност
256	2^{256}	изузетно јако	савршена сигурност

Табела 2.1: Оцена сигурности кључа у зависности од дужине



Слика 2.4: Рунде у блоковској шифри

2.3 Алгоритам *KASUMI*

Алгоритам *KASUMI* настао је као модификација блоковске шифре *MISTY*, јапанског изумитеља Mitsuru Matsui-ja. Реч *KASUMI* на јапанском значи "магловит". Он се користи у трећој генерацији стандарда мобилне телефоније. Користи кључ од 128 бита и отворени текст од 64 бита, а извршава осам рунди заснованих на Феистеловој структури, те је ефикасан и лак за имплементацију и хардверски прихватљивији од других верзија. *KASUMI* у својој имплементацији користи и функције *FO*, *FL*, *FI* са којим ћемо се упознати у наставку. У следећој глави рада видећемо да је напад на *KASUMI* могућ, али најпре упознајмо се са радом алгоритма [12].

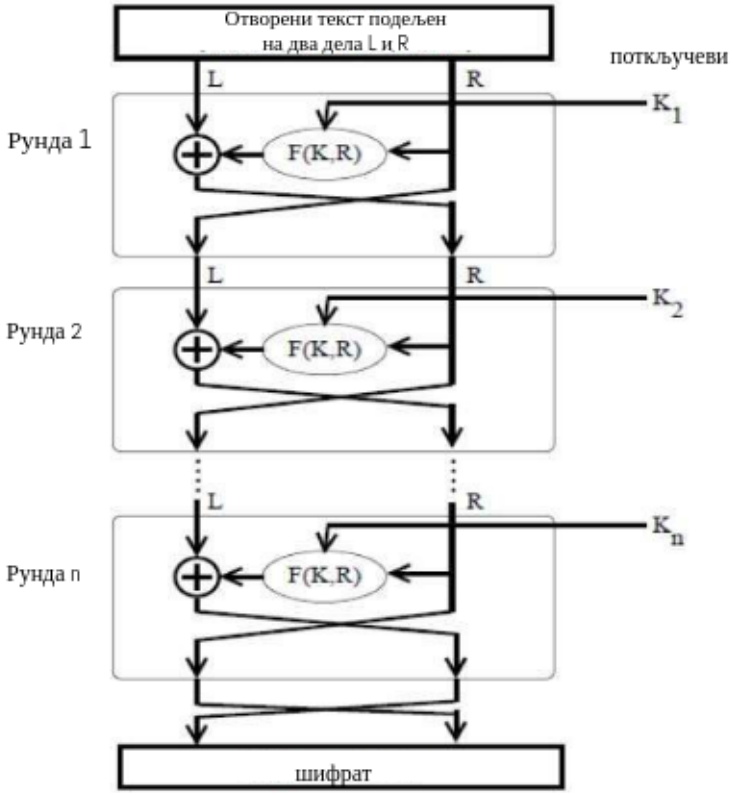
2.3.1 Феистелова структура

Феистелова структура се користи у више различитих блоковских алгоритама и користи исти метод за шифровање и дешифровање, тј. представља симетричну шифру. Идеја је да отворени текст пролази кроз више рунди трансформације тј. супституције и пермутације. Већи број рунди обезбеђује већу сигурност система, али и спорију имплементацију. На слици 2.5 приказана је основна концепција структуре. Наиме, отворени текст се дели на два дела L и R, с тим да L улази у *XOR* трансформацију са резултатом фунције *F(R, K)*, која користи други део поруке и поткључ[1] K текуће рунде. Пре уласка у следећу рунду поруке L и R мењају места. Процес се понавља. На крају последње рунде се места L и R опет замене и тако се добија шифрат.

Дакле, у свакој рунди *i+1* се обављају операције

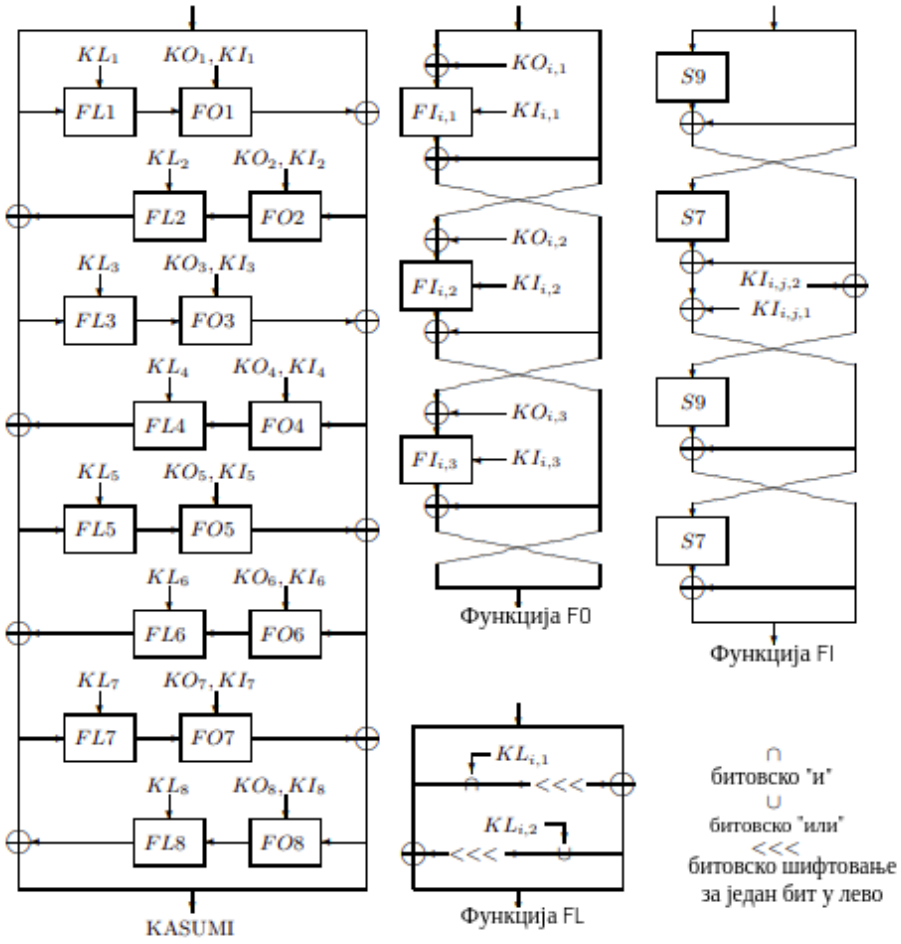
$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus F(R_i, K_i)$$



Слика 2.5: Феистелова структура

[1]Поткључ представља део настао из оригиналног кључа из алгоритма. Поткључ се користи како би се обезбедила другачија вредност кључа у свакој рунди.



Слика 2.6: Илустрација функција и корака у алгоритму *KASUMI*

У *KASUMI* алгоритму улаз се дели на два 32-битна дела. У свакој рунди извршавају се функције *FO* и *FL*. Илустрације функција које се користе у алгоритму *KASUMI* можемо видети на слици 2.6. Алгоритам *KASUMI* користи наведене функције *FO* и *FL* у редоследу који зависи од рунде. Ако је у парној рунди *FO* се изврши прва, а у непарној рунди се *FL* функција извршава прва.

Код 2.1: Функција која у одговарајућем редоследу позива одговарајуће функције

```
def kasumi(self, input, round):
    #Odgovarajuci redosled poziva funkcija u zavisnosti od
    #toga da li je runda parna ili neparna
    if round % 2 == 1:
        tmp  = self.FL(input, round)
        output = self.FO(tmp, round)
    else:
        tmp  = self.FO(input, round)
        output = self.FL(tmp, round)
```

```
    return output
```

2.3.2 Функција *FO*

FO функција прима улаз од 32 бита, састоји од Феистелове стурктуре са три рунде и даје излаз од 32 бита. Улаз се дели на два 16-битна дела и један део се комбинује са два поткључа (од 16 бита) по рунди. У свакој рунди се такође извршава једна Феистелова структура *FI* са четири рунде.

Код 2.2: Функција *FO*

```
def FO(self, input, round):
    #Delimo ulaz na dva 16-bitna dela
    left = input >> 16
    right = input & 0xFFFF

    #Tri puta se transformise
    first_left = right
    first_right = self.FI(left ^ self.key_KO1[round], self.key_KI1[round_)
        ^ right

    second_left = first_right
    second_right = self.FI(first_left ^ self.key_KO2[round],
        self.key_KI2[round]) ^ first_right

    third_left = second_right
    third_right = self.FI(second_left ^ self.key_KO3[round],
        self.key_KI3[round]) ^ second_right

    return (third_left << 16) | third_right
```

2.3.3 Функција *FI*

FI функција прима 16-битни улаз, дели га на два дела од 7 и 9 бита и трансформише га користећи седмобитне и деветобитне табеле супституције (означене као *S7* и *S9* кутије), креирајући коначно излаз од 16 бита. У трансформацији користи и 16-битни поткључ.

Код 2.3: Функција *FI*

```
#Ova funkcija koristi tabele S7 i S9
def FI(self, input, key):
    #Sesnaestobitni ulaz se deli na dva dela od sedam i devet bita
    left = input >> 7
    right = input & 0b1111111

    key_1 = key >> 9
```

```
        key_2 = key & 0b111111111

        tmp_l = right
        tmp_r = S9[left] ^ right

        left = tmp_r ^ key_2
        right = S7[tmp_l] ^ (tmp_r & 0b1111111) ^ key_1

        tmp_l = right
        tmp_r = S9[left] ^ right

        left = S7[tmp_l] ^ (tmp_r & 0b1111111)
        right = tmp_r

        #Spajanje podeljenog ulaza
        return (left << 9) | right
```

2.3.4 Функција *FL*

Функција *FL* ради са 32 бита улаза који се дели, где један улази у операцију битовско *"или"* (*OR*), а други улази у операцију битовско *"и"* (*AND*) са шеснаестобитним поткључевима.

Код 2.4: Функција *FL*

```
def FL(self, input, round):
    #Ulaz se opet deli na levi i desni deo
    left = input >> 16
    right = input & 0xFFFF

    right = right ^ _shift(left & self.key_KL1[round], 1)
    left = left ^ _shift(right | self.key_KL2[round], 1)

    return (left << 16) | right
```

2.3.5 Табеле (кутије) *S7* и *S9*

Табеле супституције *S7* и *S9* садрже логику која за дати улаз даје одговарајући излаз користећи задате вредности у табелама. У спецификацији [1] налазимо да за дати улаз, са битовима означеним као

$$x = x6||x5||x4||x3||x2||x1||x0,$$

добијамо излаз чији се битови рачунају на следећи начин (користећи табелу *S7*):

$$y0 = x1x3 \oplus x4 \oplus x0x1x4 \oplus x5 \oplus x2x5 \oplus x3x4x5 \oplus x6 \oplus x0x6 \oplus x1x6 \oplus x3x6 \oplus x2x4x6$$

$$\oplus x1x5x6 \oplus x4x5x6$$

$$y1 = x0x1 \oplus x0x4 \oplus x2x4 \oplus x5 \oplus x1x2x5 \oplus x0x3x5 \oplus x6 \oplus x0x2x6 \oplus x3x6 \oplus x4x5x6 \oplus 1$$

$$y2 = x0 \oplus x0x3 \oplus x2x3 \oplus x1x2x4 \oplus x0x3x4 \oplus x1x5 \oplus x0x2x5 \oplus x0x6 \oplus x0x1x6 \oplus x2x6$$

$$\oplus x4x6 \oplus 1$$

$$y3 = x1 \oplus x0x1x2 \oplus x1x4 \oplus x3x4 \oplus x0x5 \oplus x0x1x5 \oplus x2x3x5 \oplus x1x4x5 \oplus x2x6 \oplus x1x3x6$$

$$y4 = x0x2 \oplus x3 \oplus x1x3 \oplus x1x4 \oplus x0x1x4 \oplus x2x3x4 \oplus x0x5 \oplus x1x3x5 \oplus x0x4x5 \oplus x1x6 \oplus x3x6$$

$$\oplus x0x3x6 \oplus x5x6 \oplus 1$$

$$y5 = x2 \oplus x0x2 \oplus x0x3 \oplus x1x2x3 \oplus x0x2x4 \oplus x0x5 \oplus x2x5 \oplus x4x5 \oplus x1x6 \oplus x1x2x6 \oplus x0x3x6$$

$$\oplus x3x4x6 \oplus x2x5x6 \oplus 1$$

$$y6 = x1x2 \oplus x0x1x3 \oplus x0x4 \oplus x1x5 \oplus x3x5 \oplus x6 \oplus x0x1x6 \oplus x2x3x6 \oplus x1x4x6 \oplus x6x5x6$$

Ознака *x1x3* означава операцију битовско *"и"* између *x1* и *x3*. Слично, постоји и логика рачуна за табелу *S9* за 9-битне улазе, која се може видети у спецификацији. Ове табеле су унапред дате и могу се видети у наставку текста у делу 2.5 и 2.6. Садрже децималне вредности, тако да се могу лако користити. На пример, нека нам је дат улаз $x = 38 = 0100110_2$, прочитаћемо вредност $S7[x] = S7[38] = 58 = 0111010_2$, и то ће бити одговарајући излаз.

Подаци 2.5: Табела *S7*

```
S7 = (
     54, 50, 62, 56, 22, 34, 94, 96, 38, 6, 63, 93, 2, 18,123, 33,
     55,113, 39,114, 21, 67, 65, 12, 47, 73, 46, 27, 25,111,124, 81,
     53, 9,121, 79, 52, 60, 58, 48,101,127, 40,120,104, 70, 71, 43,
     20,122, 72, 61, 23,109, 13,100, 77, 1, 16, 7, 82, 10,105, 98,
     117,116, 76, 11, 89,106, 0,125,118, 99, 86, 69, 30, 57,126, 87,
     112, 51, 17, 5, 95, 14, 90, 84, 91, 8, 35,103, 32, 97, 28, 66,
     102, 31, 26, 45, 75, 4, 85, 92, 37, 74, 80, 49, 68, 29,115, 44,
     64,107,108, 24,110, 83, 36, 78, 42, 19, 15, 41, 88,119, 59, 3,
)
```

Подаци 2.6: Табела *S9*

```
S9 = (
     167,239,161,379,391,334, 9,338, 38,226, 48,358,452,385, 90,397,
     183,253,147,331,415,340, 51,362,306,500,262, 82,216,159,356,177,
     175,241,489, 37,206, 17, 0,333, 44,254,378, 58,143,220, 81,400,
     95, 3,315,245, 54,235,218,405,472,264,172,494,371,290,399, 76,
     165,197,395,121,257,480,423,212,240, 28,462,176,406,507,288,223,
     501,407,249,265, 89,186,221,428,164, 74,440,196,458,421,350,163,
     232,158,134,354, 13,250,491,142,191, 69,193,425,152,227,366,135,
     344,300,276,242,437,320,113,278, 11,243, 87,317, 36, 93,496, 27,
```

```
        487,446,482, 41, 68,156,457,131,326,403,339, 20, 39,115,442,124,
        475,384,508, 53,112,170,479,151,126,169, 73,268,279,321,168,364,
        363,292, 46,499,393,327,324, 24,456,267,157,460,488,426,309,229,
        439,506,208,271,349,401,434,236, 16,209,359, 52, 56,120,199,277,
        465,416,252,287,246, 6, 83,305,420,345,153,502, 65, 61,244,282,
        173,222,418, 67,386,368,261,101,476,291,195,430, 49, 79,166,330,
        280,383,373,128,382,408,155,495,367,388,274,107,459,417, 62,454,
        132,225,203,316,234, 14,301, 91,503,286,424,211,347,307,140,374,
        35,103,125,427, 19,214,453,146,498,314,444,230,256,329,198,285,
        50,116, 78,410, 10,205,510,171,231, 45,139,467, 29, 86,505, 32,
        72, 26,342,150,313,490,431,238,411,325,149,473, 40,119,174,355,
        185,233,389, 71,448,273,372, 55,110,178,322, 12,469,392,369,190,
        1,109,375,137,181, 88, 75,308,260,484, 98,272,370,275,412,111,
        336,318, 4,504,492,259,304, 77,337,435, 21,357,303,332,483, 18,
        47, 85, 25,497,474,289,100,269,296,478,270,106, 31,104,433, 84,
        414,486,394, 96, 99,154,511,148,413,361,409,255,162,215,302,201,
        266,351,343,144,441,365,108,298,251, 34,182,509,138,210,335,133,
        311,352,328,141,396,346,123,319,450,281,429,228,443,481, 92,404,
        485,422,248,297, 23,213,130,466, 22,217,283, 70,294,360,419,127,
        312,377, 7,468,194, 2,117,295,463,258,224,447,247,187, 80,398,
        284,353,105,390,299,471,470,184, 57,200,348, 63,204,188, 33,451,
        97, 30,310,219, 94,160,129,493, 64,179,263,102,189,207,114,402,
        438,477,387,122,192, 42,381, 5,145,118,180,449,293,323,136,380,
        43, 66, 60,455,341,445,202,432, 8,237, 15,376,436,464, 59,461,
)
```

2.3.6 Поткључеви

Кључ од 128 бита, који улази у алгоритам, се најпре дели линеарно на поткључеве дужине 16 бита.

$$K = K_1||K_2||K_3||K_4||K_5||K_6||K_7||K_8$$

Сваки од ових поткључева улази у линеарну трансформацију са фиксираним константама C_i (које су наведене у *3GPP* спецификацији [1] и приказане у табели 2.2):

$$K'_i = K_i \oplus C_i$$

Подаци 2.7: Трансформација кључа

```
    def KeySchedule(self, k):
        assert _bitlen(k) <= 128

        C = [0x0123, 0x4567, 0x89AB, 0xCDEF, 0xFEDC, 0xBA98, 0x7654, 0x3210]
        key       = [None] * 8
        key_prime = [None] * 8
```

C1	0x0123
C2	0x4567
C3	0x89AB
C4	0xCDEF
C5	0xFEDC
C6	0xBA98
C27	0x7654
C8	0x3210

Табела 2.2: Вредности константе C_i за осам рунди

```
for i in range(0, 8):
    key[i] = (k >> (16 * (7 - i))) & 0xFFFF
    #transformacija sa konstantama unaped datim
    key_prime[i] = key[i] ^ C[i]

for i in range(0, 8):
    self.key_KL1[i] = _shift(key[_mod(i + 0)], 1)
    self.key_KL2[i] = key_prime[_mod(i + 2)]
    self.key_KO1[i] = _shift(key[_mod(i + 1)], 5)
    self.key_KO2[i] = _shift(key[_mod(i + 5)], 8)
    self.key_KO3[i] = _shift(key[_mod(i + 6)], 13)
    self.key_KI1[i] = key_prime[_mod(i + 4)]
    self.key_KI2[i] = key_prime[_mod(i + 3)]
    self.key_KI3[i] = key_prime[_mod(i + 7)]
```

Промене ових осам поткључева по рундама у алгоритму *KASUMI* приказане су на слици 2.7. Овим процесом се у свакој рунди 128-битни кључ линеарно мења, тако да увек представља другачију верзију главног кључа K.

Рунда	$KL_{i,1}$	$KL_{i,2}$	$KO_{i,1}$	$KO_{i,2}$	$KO_{i,3}$	$KI_{i,1}$	$KI_{i,2}$	$KI_{i,3}$
1	$K_1 \lll 1$	K'_3	$K_2 \lll 5$	$K_6 \lll 8$	$K_7 \lll 13$	K'_5	K'_4	K'_8
2	$K_2 \lll 1$	K'_4	$K_3 \lll 5$	$K_7 \lll 8$	$K_8 \lll 13$	K'_6	K'_5	K'_1
3	$K_3 \lll 1$	K'_5	$K_4 \lll 5$	$K_8 \lll 8$	$K_1 \lll 13$	K'_7	K'_6	K'_2
4	$K_4 \lll 1$	K'_6	$K_5 \lll 5$	$K_1 \lll 8$	$K_2 \lll 13$	K'_8	K'_7	K'_3
5	$K_5 \lll 1$	K'_7	$K_6 \lll 5$	$K_2 \lll 8$	$K_3 \lll 13$	K'_1	K'_8	K'_4
6	$K_6 \lll 1$	K'_8	$K_7 \lll 5$	$K_3 \lll 8$	$K_4 \lll 13$	K'_2	K'_1	K'_5
7	$K_7 \lll 1$	K'_1	$K_8 \lll 5$	$K_4 \lll 8$	$K_5 \lll 13$	K'_3	K'_2	K'_6
8	$K_8 \lll 1$	K'_2	$K_1 \lll 5$	$K_5 \lll 8$	$K_6 \lll 13$	K'_4	K'_3	K'_7

$(X \lll i)$ — шифтовање у лево за i бита у броју X

Слика 2.7: Промене у кључу у свакој од осам рунди

2.4 Прелазак са *MISTY* на *KASUMI*

Пребацивање у *GSM* систему са алгоритма *MISTY* на *KASUMI* показало се као лоша одлука по систем, с обзиром да није постигнут напад на целокупни алгоритам *MISTY* од осам рунди, док је напад на *KASUMI* могућ. Идеја дизајнирања *KASUMI* алгоритма је креирање мање хардверски захтевног алгоритма, бржег у извршавању [1].

Неке измене које су извршене у односу на *MISTY* су:

- Премештање функције *FL* (поједностављује се хардвер).
- Додавање битовског померања (овиме се претпоставља отежавање криптоанализе).
- Промене у табелама супституције *S7* и *S9*.
- Поједностављање алгоритма због хардвера уклањањем функција *FI* у процесу трансформације кључа.
- Додавање случајно изабраних константи у трансформацији поткључа и тако даље.

У самој техничкој спецификацији *GSM* система једна од лоше постављених претпоставки јесте да одређене измене у процесу креирања поткључа неће проузроковати могућност напада на алгоритам, и то баш техникама којима је алгоритам разбијен. Алгоритам је ослабљен доста тиме што је креирање кључа и поткључева линеарна операција док то у *MISTY*-ју није била. У раду [12] говори се о нападу који успева да 128-битни кључ одреди за два сата.

Глава 3

Криптоанализа

3.1 Диференцијална криптоанализа

Концепт диференцијалне криптоанализе биће представљен над алгоритмом шифровања *SPN*, као на слици 3.1. Посматрајући ту слику биће јасна и процедура по којој овај алгоритам ради. *SPN* очекује 16-битни улаз који пролази кроз четири рунде супституција, пермутација и деловања кључа. Попут *KASUMI*-ја заснован је на Феистеловој структури. Улаз од 16 бита се дели на четири блока, после чега се сваки блок води на улаз у табелу супституција S, која за одговарајући 4-битни улаз даје 4-битни излаз (нелинеарним трансформацијама, због веће сигурности). Супституциона табела S је 3.1. Пермутација представља замену позиције битова након изласка из табеле S, видети табелу 3.2, а примењује се пред улазак у табелу супституције следеће рунде (након последње рунде корак пермутације се изоставља). Свака рунда има свој део кључа који се, пре уласка у табелу S, *XOR*-ује са битовима улаза. Након последње рунде овај корак се не изоставља [8].

Диференцијална криптоанализа се заснива на посматрању разлика у отвореном тексту и разлика у последњој рунди блоковске шифре, где су нам занимљиве оне разлике са великом вероватноћом појављивања [5]. Ако имамо улаз у систем облика

$$X = [X_1, X_2, ..., X_n],$$

њему одговарајући излаз биће облика

$$Y = [Y_1, Y_2, ..., Y_n].$$

Ако посматрамо два таква улаза X' и X'' и два излаза која њима одговарају Y' и Y'', разлике између њих рачунамо битовском операцијом XOR:

$$\Delta X = X' \oplus X''$$

$$\Delta Y = Y' \oplus Y''$$

улаз	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
излаз	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

Табела 3.1: Табела S алгоритма *SPN*

Дакле, разлика између улаза биће битски вектор

$$\Delta X = [\Delta X_1, \Delta X_2, ..., \Delta X_n],$$

где је

$$\Delta X_i = X_i' \oplus X_i''.$$

Аналогно добијамо и разлику између излаза:

$$\Delta Y = [\Delta Y_1, \Delta Y_2, ..., \Delta Y_n],$$

где је

$$\Delta Y_i = Y_i' \oplus Y_i''.$$

Пар $(\Delta Y, \Delta X)$ назива се *диференцијал*. Произвољно бирајући отворени текст (улаз) нападач покушава да одреди кључ, фиксирајући ΔX и бирајући X' и X'' тако да њихова разлика буде ΔX, знајући да се за то ΔX одређено ΔY појављује са великом вероватноћом[1].

Процес се заснива на томе да је излазна разлика за једну рунду заправо улазна разлика за следећу рунду. Тражење врло вероватних низова улазних и излазних разлика (диференцијалних карактеристика) врши се анализом особина табеле S. Табела S је у шифри *SPN* бијективно пресликавање четворке бита $S : 0,1^4 -> 0,1^4$. Може се представити као табела битова (где се види пресликавање једне четворке битова у другу), или краће декадним бројевима који њима одговарају.

Табела S се анализира тако што се разматрају сви парови (X', X''). За задату разлику ΔX посматрају се све вредности X' из табеле S, рачунајући вредност за $X'' = \Delta X \oplus X'$, што је могуће због комутативности операције *XOR*. За различите парове таквих улаза $(X', X'' = \Delta X \oplus X')$ табеле S (на пример табеле 3.1) рачуна се ΔY.

У табели 3.2 у последње три колоне приказане су одговарајуће вредности ΔY за различито X и ΔX (колона Y се добија из табеле 3.1). Комплетне податке броја појављивања одговарајуће разлике излаза ΔY, за улазну разлику ΔX, приказујемо у такозваној табели расподеле разлика, као у табели 3.3, у којој врсте представљају ΔX, а колоне ΔY.

Идеална табела била би она која има све вредности једнаке 1, што би означавало вероватноћу $1/16$ (тј. $1/2^n$), међутим таква табела не постоји, јер улазној

[1]Вероватноћа да се за задато ΔX појави одређена разлика ΔY код статистички најнепредвидљивије шифре је $1/2^n$ (n представља број битова улаза). Нападача занима диференцијал који има знатно већу вероватноћу да се ΔY појави за неко ΔX.

X	Y	ΔY		
		$\Delta X = 1011$	$\Delta X = 1000$	$\Delta X = 0100$
0000	1110	0010	1101	1100
0001	0100	0010	1110	1011
0010	1101	0111	0101	0110
0011	0001	0010	1011	1001
0100	0010	0101	0111	1100
0101	1111	1111	0110	1011
0110	1011	0010	1011	0110
0111	1000	1101	1111	1001
1000	0011	0010	1101	0110
1001	1010	0111	1110	0011
1010	0110	0010	0101	0110
1011	1100	0010	1011	1011
1100	0101	1101	0111	0110
1101	1001	0010	0110	0011
1110	0000	1111	1011	0110
1111	0111	0101	1111	1011

Табела 3.2: Разлика парова табеле S

разлици $\Delta X = 0$ одговара увек излазна разлика $\Delta Y = 0$, па је вредност у одговарајућој ћелији табеле увек бити 2^n (за табелу димензије $n \times n$) тј. 16 у табели 3.3. У табели расподела разлика видимо да је за $\Delta X = B$ и $\Delta Y = 2$ вредност у табели 8, што значи да за случајно изабрани пар улаза са разликом $\Delta X = B$, вероватноћа да је излазна разлика $\Delta Y = 2$ је $8/16$.

Анализом улазних и излазних разлика табеле S оређује се она излазна разлика која има велику вероватноћу појављивања за дату улазну разлику. Проналаском улаза у последњу рунду (тако што се, од рунде до рунде, анализирају излазне разлике једне рунде, које представљају улазне разлике следеће рунде) може се одредити последњи поткључ шифре. Добија се диференцијал који се састоји од разлике два отворена текста и два улаза у последњу рунду. Битови поткључа се елиминишу када те разлике пролазе кроз операцију *XOR*, јер поткључ учествује у оба податка.

Пошто комбинујемо кључ са улазом у сваку табелу S, улазе заправо треба посматрати као $W' = X' \oplus K$ и $W'' = X" \oplus K$. За два улаза W' и W'', који се комбнују са истим кључем, улазна разлика за овакав проширени блок је:

$$\Delta W = [W'_1 \oplus W''_1, W'_2 \oplus W''_2, ..., W'_n \oplus W''_n].$$

Битови кључа не утичу на улазну разлику и могу се игнорисати, јер важи:

$$\Delta W_i = W'_i \oplus W''_i = (X'_i \oplus K_i) \oplus (X''_i \oplus K_i) = X'_i \oplus X''_i = \Delta X_i$$

Дакле, табела разлика за проширени блок је иста као табела разлика за S без кључа.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	0	0	0	2	0	2	4	0	4	2	0	0
2	0	0	0	2	0	6	2	2	0	2	0	0	0	0	2	0
3	0	0	2	0	2	0	0	0	0	4	2	0	2	0	0	4
4	0	0	0	2	0	0	6	0	0	2	0	4	2	0	0	0
5	0	4	0	0	0	2	2	0	0	0	4	0	2	0	0	2
6	0	0	0	4	0	4	0	0	0	0	0	0	2	2	2	2
7	0	0	2	2	2	0	2	0	0	2	2	0	0	0	0	4
8	0	0	0	0	0	0	2	2	0	0	0	4	0	4	2	2
9	0	2	0	0	2	0	0	4	2	0	2	2	2	0	0	0
A	0	2	2	0	0	0	0	0	6	0	0	2	0	0	4	0
B	0	0	8	0	0	2	0	2	0	0	0	0	0	2	0	2
C	0	2	0	0	2	2	2	0	0	0	0	2	0	6	0	0
D	0	4	0	0	0	0	0	4	2	0	2	0	2	0	2	0
E	0	0	2	4	2	0	0	0	6	0	0	0	0	0	2	0
F	0	2	0	0	6	0	0	0	0	4	0	2	0	0	2	0

Табела 3.3: Табела расподеле разлика за табелу S

Диференцијалну карактеристику целог система, када имамо дате табеле S, одређујемо разматрањем парова разлика табела, по рундама. Занимају нас утицаји не-нула битских улазних разлика (ако је улазна разлика у табелу S нула, излазна разлика је такође нула, па се такве разлике не разматрају).

Треба добити диференцијал који обухвата битове отвореног текста и битове улаза у последњу рунду у табелу S. Када се одреди такав диференцијал, могуће је напасти систем одређивањем поткључа који се користи после последње рунде.

Пример са слике 3.1 илуструје диференцијалну карактеристику за прве три рунде истичући црном линијом табеле које учествују у изабраној диференцијалној криптоанализи, табеле са улазном разликом различитом од нуле. Користећи ову диференцијалну карактеристику може се одредити парцијални поткључ последње рунде.

У првој означеној табели S_{12} улазна разлика је $\Delta X = B$, а излазна разлика $\Delta Y = 2$ са вероватноћом $8/16$. Тако одредимо и диференцијале остале три табеле S_{23}, S_{32}, S_{33}.

На основу дијаграма приметимо да је улазна разлика прве рунде

$$\Delta U_1 = [0000101100000000],$$

док је излазна разлика из S табеле у тој рунди

$$\Delta V_1 = [0000001000000000].$$

Улазна разлика прве рунде представља и улазну разлику целе шифре (коју ћемо означити са ΔP), па је $\Delta P = \Delta U_1$. После пермутација у првој рунди добијамо

улазну разлику друге рунде

$$\Delta U_2 = [0000000001000000]$$

са вероватноћом појављивања $8/16$ у односу на дату улазну разлику ΔP. Аналогно разматрамо остале улазне и излазне разлике. Излазна разлика из одговарајуће табеле S у другој рунди биће

$$\Delta V_2 = [0000000001100000].$$

Након пермутације друге рунде, са вероватноћом $6/16$ у односу на ΔU_2, добија се

$$\Delta U_3 = [0000001000100000],$$

а потом и

$$\Delta V_3 = [0000000001100000].$$

На крају, са вероватноћом $(6/16)^2$ у односу на ΔU_3, добија се

$$\Delta U_4 = [0000011000000110].$$

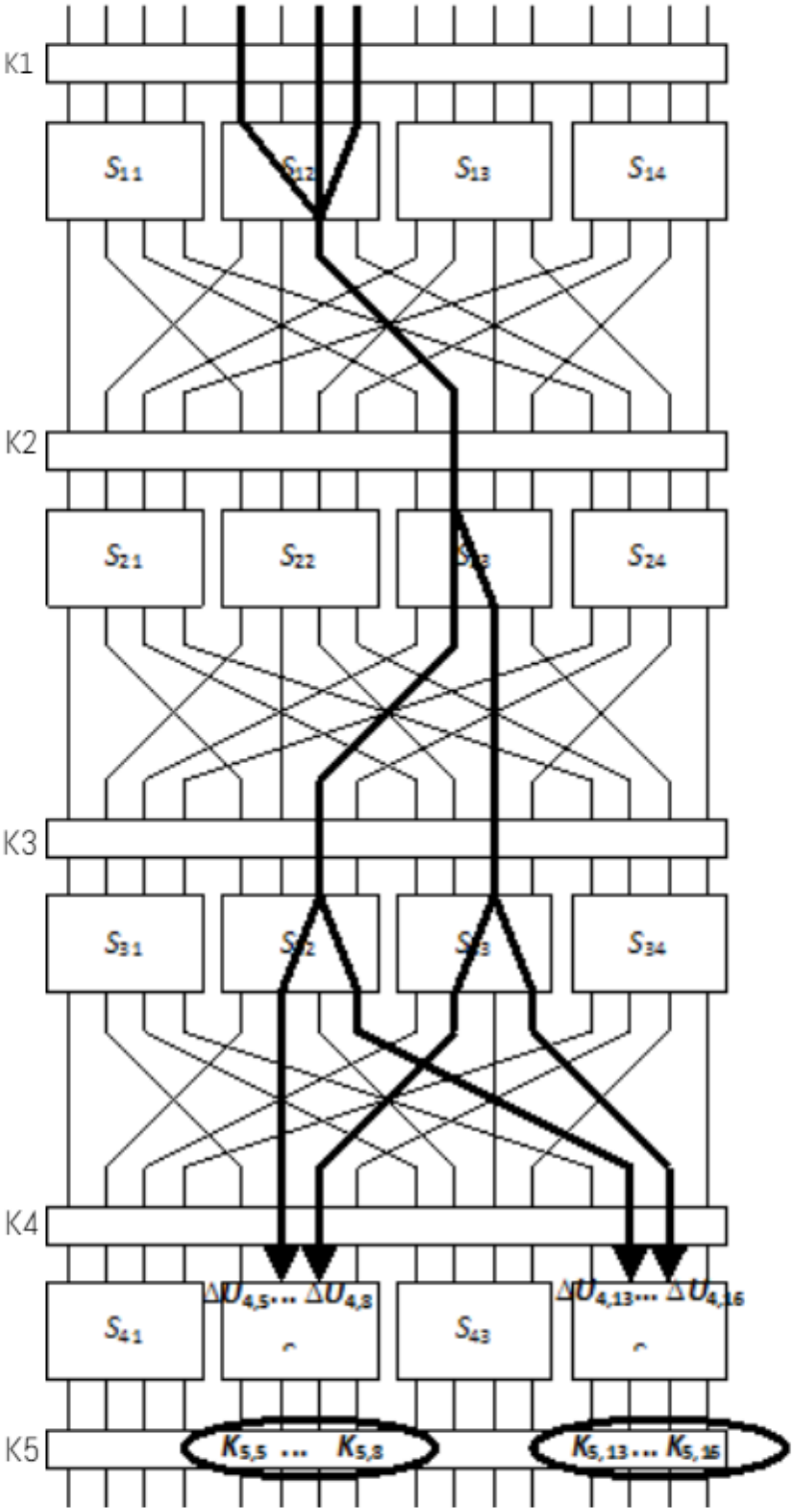
Пошто се претпоставља независност у разликама по рундама, добијене вероватноће се множе и дају укупну вероватноћу последњег излаза $8/16 * 6/16 * (6/16)^2 = 27/1024$, за улазну разлику ΔP. Много различитих парова улаза P^1 и P^2 се шифрује, тако да за њих важи

$$\Delta P = P^1 \oplus P^2 = [00000101100000000],$$

а са вероватноћом $27/1024$ добиће се разматрана диференцијална карактеристика.

У свакој рунди постојао је и поткључ који улази у одговарајуће табеле S. Циљани поткључ који се одређује у примеру са слике 3.1 биће K_5, поткључ после последње рунде са не-нула разликама. Могу се добити битови друге и четврте четворке тог парцијалног поткључа, као што је означено на слици. То је могуће тако што се крене од познатог излаза $V4$, излаза из последње рунде, и делимичним дешифровањем са свим могућим вредностима поткључа K_5 добија се део улаза последње рунде. Обележава се успешност за сваки поткључ тј. обележава се када разлика улаза последње рунде одговара датој диференцијалној карактеристици која је коришћена у рачуну. Најуспешнији поткључ се бира као тражени.

Дакле, за неки дати пар шифрата испробавају се све могуће вредности битова последњег поткључа $K_{5/5}, ..., K_{5/8}..., K_{5/13}, ...K_{5/16}$, њих 256. За сваки могући парцијални поткључ, проласком уназад са шифратима кроз табеле S_{24} и S_{44} одређују се $\Delta U_{4/5}, ..., \Delta U_{4/8}, ..., \Delta U_{4/13}, ...U_{4/16}$. За те парцијалне поткључеве се затим обележава успешност, ако се добије таква улазна разлика која је једнака



Слика 3.1: Пример диференцијалне карактеристике

унапред датим ΔU_4. Онај парцијални поткључ који има највећу успешност сматра се за циљаним тј. траженим парцијалним поткључем. Закључак у примеру са слике је да се добија $K_{5/5}, ..., K_{5/8}..., K_{5/13}, ...K_{5/16} = [0010, 0100]$[2], који ће имати највећу успешност 0.0244 (што одговара и вероватноћи 27/1024 за одговарајуће парове улаза).

3.2 Бумеранг и сендвич напад

Бумеранг напад је заснован на диференцијалној криптоанализи. У диференцијалној криптоанализи, нападач се служи већ описаним разликама анализирајући цео процес, од отвореног текста до шифрата. Оно што је потребно је наћи диференцијал са великом вероватноћом који покрива целу или скоро целу шифру. Бумеранг напад користи диференцијал који покрива само део шифре, при чему се уместо пара улаза користе четворке улаза, а разлике се примењују и на поруку и на кључ.

3.2.1 Бумеранг напад

Бумеранг напад разматра четири отворена текста

$$P, P', Q, Q'$$

и њима одговарајуће шифрате

$$C, C', D, D'.$$

Нека је алгоритам шифровања E, који се дели на композицију E_0 и E_1 [13]. Бумеранг напад се користи диференцијалним карактеристикама, нека су то $\Delta -> \Delta^*$ за E_0 и $\nabla -> \nabla^*$ за E_1^{-1}. Диференцијална карактеристика за E_0 треба да обухвати улазе P и P', док диференцијална карактеристика за E_1^{-1} треба да обухвати парове P, Q и P', Q'. Када је то задовољено може се закључити да пар Q, Q' задовољава диференцијалну карактеристику $\Delta^* -> \Delta$ за E_0^{-1}.

Са слике 3.2 се види ова шема бумеранг напада, у коме се креира структура четворке. На пола пута шифровања, за инверзну операцију, биће:

$$E_0(Q) \oplus E_0(Q') =$$

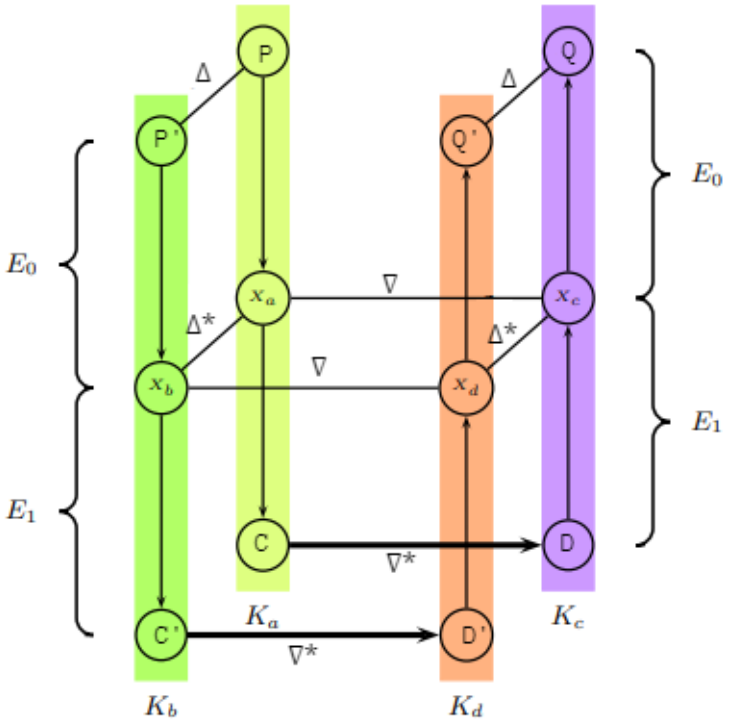
$$E_0(P) \oplus E_0(P') \oplus E_0(P) \oplus E_0(Q) \oplus E_0(P') \oplus E_0(Q') =$$

$$E_0(P) \oplus E_0(P') \oplus E_1^{-1}(C) \oplus E_1^{-1}(D) \oplus E_1^{-1}(C') \oplus E_1^{-1}(D') =$$

$$\Delta^* \oplus \nabla^* \oplus \nabla^* = \Delta^*$$

[2]За детаље погледати текст из референце [5]

Када ово важи, можемо рећи да је разлика у улазима Q, Q' иста као разлика за P, P' [2].



Слика 3.2: Приказ бумеранг напада

Ако имамо диференцијал $\Delta -> \Delta^*$ за E_0, и разлику кључа ΔK_{ab} са вероватноћом p, то се заправо може написати као

$$Pr[E_{0(K)}(P) \oplus E_{0(K \oplus \Delta K_{ab})}(P \oplus \Delta) = \Delta^*] = p,$$

где је $E_{0(K)}$ процес шифровања кроз први део алгоритма E_0 користећи кључ K. Такође, имамо диференцијал $\nabla -> \nabla^*$ за E_1, и разлику кључа ΔK_{ac} са вероватноћом q.

Алгоритам препознавача бумеранг напада ради над алгоритмом шифровања и дешифровања користећи кључ K_a, као и повезане кључеве

$$K_b = K_a \oplus \Delta K_{ab}$$

$$K_c = K_a \oplus \Delta K_{ac}$$

$$K_d = K_c \oplus \Delta K_{ab} = K_b \oplus \Delta K_{ac}.$$

Алгоритам препознавача је следећи:

1. Бира се више отворених текстова, иницијализује се бројач Z на 0. За сваки пар отворених текстова:

- Израчунају се њима одговарајући шифрати $C = E_{K_a}(P)$ и $C' = E_{K_b}(P')$, где је $P' = P \oplus \Delta$.
- Израчунају се отворени текстови $Q = E_{K_c}^{-1}(D)$ и $Q' = E_{K_d}^{-1}(D')$, где важи $D = C \oplus \nabla^*$, $D' = C' \oplus \nabla^*$.
- Након тих корака, ако важи $Q \oplus Q' = \Delta$, увећамо бројач Z за један.

2. Ако Z пређе неку задату границу поверења, праг, онда препознавач даје TRUE (да је добијена циљана шифра), иначе FALSE (добијена је нека случајна пермутација).

Речима би то било објашњено на следећи начин: изабере се неки диференцијал Δ и улаз P. Затим се израчуна $P' = P \oplus \Delta$. Они се пропусте кроз алгоритам шифровања и добију се њима одговарајући шифрати C, C'. Затим се шифрати мало измене за изабрани диференцијал ∇ и добију се шифрати $D = C \oplus \nabla$ i $D' = C' \oplus \nabla$. Ти нови шифрати се пропусте уназад кроз алгоритам да се добију њима одговарајући отворени текстови Q, Q'. Испитује се да ли задовољавају исти диференцијал и анализира се процес за различите изабране отворене текстове [6].

Вероватноћа да је (P, P') одговарајући пар, тј. да је диференцијал након E_0 једнак Δ^*, је p. Вероватноћа да су парови (C, C'), (D, D') добри у односу на други диференцијал је q^2. Пошто су то добро изабрани парови, важи

$$E_1^{-1}(D) \oplus E_1^{-1}(D') = \Delta^* = E_0(Q) \oplus E_0(Q').$$

Следи да је $Q \oplus Q' = \Delta$. Дакле вероватноћа да је испуњен услов $Q \oplus Q' = \Delta$ је бар (pq^2).

3.2.2 Сендвич напад

Сендвич напад базира се на бумеранг нападу с тим да постоји три слоја на која се дели алгоритам шифровања, $E = E_0 \circ M \circ E_1$, где је M углавном једна рунда алгоритма. Разлика у односу на бумеранг је управо у том средњем слоју M. Шему сендвич напада можемо видети на слици 3.3.

Ако су парови (P, P'), (C, C'), (D, D') добро изабрани у односу на неке дате вредности диференцијала, у бумеранг нападу следи да важи:

$$(X_1 \oplus X_2 = \Delta^*) \wedge (X_1 \oplus X_3 = \nabla) \wedge (X_2 \oplus X_4 = \nabla),$$

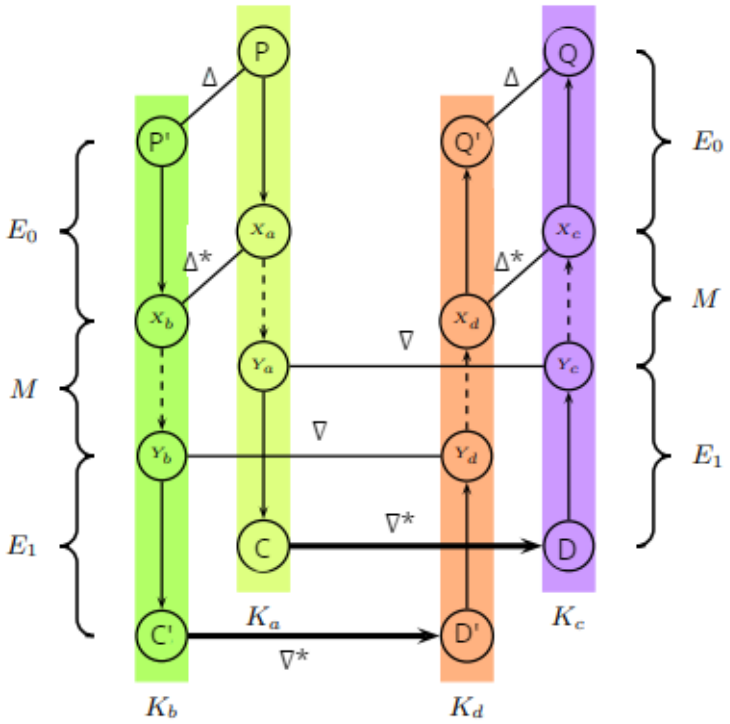
где су Xi вредности шифрата у средини напада (између E_0 и E_1 алгоритма) који одговарају редом отвореним текстовима *P, P', Q, Q'*. У сендвич нападу услов је другачији:

$$(X_1 \oplus X_2 = \Delta^*) \wedge (Y_1 \oplus Y_3 = \nabla) \wedge (Y_2 \oplus Y_4 = \nabla),$$

што значи да је вероватноћа сендвич напада са три слоја шифровања p^2q^2r, где је

$$r = Pr[(X_3 \oplus X_4 = \Delta^*)|((X_1 \oplus X_2 = \Delta^*) \wedge (Y_1 \oplus Y_3 = \nabla) \wedge (Y_2 \oplus Y_4 = \nabla))].$$

Идеја процеса сендвич напада је слична као код бумеранга, полазећи од случајног избора отвореног текста па до провере услова.



Слика 3.3: Шема сендвич напада

Глава 4

Имплементација

4.1 Алгоритам *KASUMI*

Користећи код алгоритма *KASUMI* који се налази у додатку *3GPP* спецификације (који се може видети у додатку A), креиран је *KASUMI* препознавач. Имплементација алгоритма *KASUMI* је рађена у програмском језику *Python*, а прилагођена је тако да има функцију за шифровање и дешифровање улазне поруке, као за дати пример из кода 4.1. Имплементација појединачних функција из алгоритма представљена је другој глави у коду 2.1, 2.2, 2.3, 2.4, 2.7.

Код 4.1: Пример шифровања и дешировања

```
if __name__ == '__main__':
    key     = 0x9900aabbccddeeff1122334455667788
    text    = 0xfedcba0987654321

    print ('Text is: ', hex(text))
    print ('\n')

    my_kasumi = Kasumi()
    my_kasumi.KeySchedule(key)

    encrypted = my_kasumi.encription(text)
    print ('Encrypted: ', hex(encrypted))

    decrypted = my_kasumi.decription(encrypted)
    print ('Decrypted: ', hex(decrypted))
```

Код из примера даће следећи резултат:

```
Text is:  0xfedcba0987654321

Encrypted: 0x514896226caa4f20
Decrypted: 0xfedcba0987654321
```

4.2 Препознавач и напад

Вредности које су изабране за диференцијале се спомињу у раду [11]. Дужине су 64 бита, а заправо су подељене на два дела од 32 бита, као што се дели улаз у *KASUMI* [4]. Хексадекадно представљене вредности диференцијала су

$$\Delta -> \Delta^* = (0x0, 0x00100000) -> (0x0, 0x00100000).$$

Кључеви који се користе у шифровању отворених текстова имају следећу разлику $\Delta K_{ab} = (0, 0, 0x8000, 0, 0, 0, 0, 0)$, $\Delta K_{ac} = (0, 0, 0, 0, 0, 0, 0x8000, 0)$. У односу на то добију се кључеви K_a, $K_b = K_a \oplus \Delta K_{ab}$, $K_c = K_a \oplus \Delta K_{ac}$, $K_d = K_c \oplus \Delta K_{ab}$.

Један тест препознавача обухватиће случајно бирање квартета кључева тако да они задовољавају дефинисане разлике, а затим генерисање 2^{16} квартета података (отворених текстова и шифрата) на начин који је описан у делу 3.2. Тест се покреће сто хиљада пута. У том броју пуштених експеримената, очекује се да је узорачка расподела броја исправних квартета у складу са Поасоновом расподелом, јер сваки изабрани квартет је независтан од следећег. Од 2^{16} изабраних квартета очекује се само 8 исправних [11].

```
delta = 0x0000000000100000
nabla = 0x0000000000100000

#U svakom testu kreiramo kvartet kljuceva
#pa onda kreiramo kvartete podataka, njih 2^16

for i in range(0, 10000):
    key_a      = random.getrandbits(128)
    delta_key_ab = 0x00008000000000000000000000000000
    delta_key_ac = 0x00000000000000000000000080000000
    key_b = key_a ^ delta_key_ab
    key_c = key_a ^ delta_key_ac
    key_d = key_c ^ delta_key_ab

    C_a = [None] * 2**16
    C_b = [None] * 2**16
    C_c = [None] * 2**16
    C_d = [None] * 2**16
    P_a = [None] * 2**16
    P_b = [None] * 2**16
    P_c = [None] * 2**16
    P_d = [None] * 2**16

    Kasumi_P_a = Kasumi()
    Kasumi_P_a.KeySchedule(key_a)
    Kasumi_P_b = Kasumi()
    Kasumi_P_b.KeySchedule(key_b)
    Kasumi_P_c = Kasumi()
```

```
Kasumi_P_c.KeySchedule(key_c)
Kasumi_P_d = Kasumi()
Kasumi_P_d.KeySchedule(key_d)

z = 0
for i in range(0, 2**16):
    #print("****************")
    P_a[i] = random.getrandbits(64)
    C_a[i] = Kasumi_P_a.encription(P_a[i])


    P_b[i] = P_a[i] ^ delta
    C_b[i] = Kasumi_P_b.encription(P_b[i])

    C_c[i] = C_a[i] ^ nabla
    C_d[i] = C_b[i] ^ nabla

    P_c[i] = Kasumi_P_c.decription(C_c[i])
    P_d[i] = Kasumi_P_d.decription(C_d[i])


    if (C_c[i] ^ C_d[i] == nabla):
        z += 1

#z je koliko je dobrih kvarteta u tom testu bilo
if z != 0:
    print(z)
```

Што се тиче напада на KASUMI, најпре се прикупљају подаци. Креира се шифрат $C_a = (X_a, A)$, дужине 64 бита, с тим да је A нека фиксна константа, док је X_a случајна вредност. У односу на добијени шифрат C_a, дешифровањем се добија одговарајући отворени текст P_a, користећи кључ K_a. Након тога се креира други отворени текст, који се од P_a разликује за унапред задати диференцијал. Дакле, важи $P_b = P_a \oplus \Delta$. Шифровањем тог отвореног текста, користећи кључ K_b, добија се шифрт C_b.

Ова процедура се извршава у петљи, тако да се добије задати број различитих X_a, па тако и различитих шифрата C_a, као и свих осталих наведених вредости. Добијене вредности (C_a, C_b) се чувају у хеш табели, тако да индекс тј. кључ сваког пара (по ком претражујемо хеш табелу) представља 32 бита десне половине шифрата C_b.

```
...
    C_a = [None] * 2**24
    C_b = [None] * 2**24
    C_c = [None] * 2**24
    C_d = [None] * 2**24
```

```
P_a = [None] * 2**24
P_b = [None] * 2**24
P_c = [None] * 2**24
P_d = [None] * 2**24

A = 0x2950412b
Kasumi_P_a = Kasumi()
Kasumi_P_a.KeySchedule(key_a)
Kasumi_P_b = Kasumi()
Kasumi_P_b.KeySchedule(key_b)
Kasumi_P_c = Kasumi()
Kasumi_P_c.KeySchedule(key_c)
Kasumi_P_d = Kasumi()
Kasumi_P_d.KeySchedule(key_d)

dict = {}
quartets = {}

for i in range(0, 2**24):
    #print("****************")
    X_a = random.getrandbits(32)
    C_a[i] = hex(append_hex(X_a, A)) #f'0x{A:x}{X_a:x}'

    P_a[i] = hex(Kasumi_P_a.decription(int(C_a[i], base = 16)))

    P_b[i] = int(P_a[i], base = 16) ^ delta
    C_b[i] = Kasumi_P_b.encription(P_b[i])

    #upis u hes tabelu
    dict[hex(C_b[i] & 0xFFFFFFFF)] = (C_a[i], hex(C_b[i]))
```

У другој петљи креирају се шифрати $C_c = (Y_c, A \oplus 0x00100000)$, где Y_c представља случајну вредност. Користећи кључ K_c, дешифровањем се добијају одговарајући отворени текстови P_c. У трансформацији са диференцијалом добијају се нови отворени текстови $P_d = P_c \oplus \Delta$, а након шифровања кључем K_d и шифрати C_d. На крају се рачунају вредности $C_d^R \oplus (0x0, 0x00100000)$, где C_d^R представља десних 32 бита шифрата C_d. Те вредности користе се такође као индекси тј. кључеви по којима се претражује хеш табела направљена у првој петљи. За сваки пронађени индекс, добијени парови (C_a, C_b) се издвајају и памте се њима одговарајући квартети (C_a, C_b, C_c, C_d). Тела петљи се извршавају 2^{24} пута. Из поставке можемо видети да су десне стране шифрата C_a и шифрата C_c једнаке, јер су креиране користећи унапред дефинисану константу A. За разлику од тога, лева половина тих шифрата, када они прођу кроз операцију XOR $(C_a^L \oplus C_c^L)$, даје увек исту вредност, за сваки C_a, C_c, ако су изабрани добри квартети [11].

```
for i in range(0, 2**24):
```

```
#print("-------------------")
Y_c = random.getrandbits(32)
C_c[i] = hex(append_hex(Y_c, A ^ 0x00100000))

P_c[i] = hex(Kasumi_P_c.decription(int(C_c[i], base = 16)))

P_d[i] = int(P_c[i], base = 16) ^ delta
C_d[i] = Kasumi_P_d.encription(P_d[i])

if hex((C_d[i] & 0xFFFFFFFF) ^ nabla) in dict:
    quartets[hex(int(hex(C_a)[:10], base = 16) ^ int(hex(C_c)[:10],
        base = 16))] = \
        ( dict[hex((C_d[i] & 0xFFFFFFFF) ^ nabla)][0], \
          dict[hex((C_d[i] & 0xFFFFFFFF) ^ nabla)][1], \
          hex(C_c), \
          hex(C_d)\
        )
```

Очекивани број издвојених квартета који задовољавају дате диференцијале, од укупног броја свих парова креираних кроз петље, је свега 2^{16}. Ти квартети се чувају у новој хеш табели, чији је кључ вредност $C_a^L \oplus C_c^L$. Претпоставља се да су ови преостали квартети добри. Овај део процеса имплементиран је у програмском језику *Python* коришћењем претходно описане процедуре.

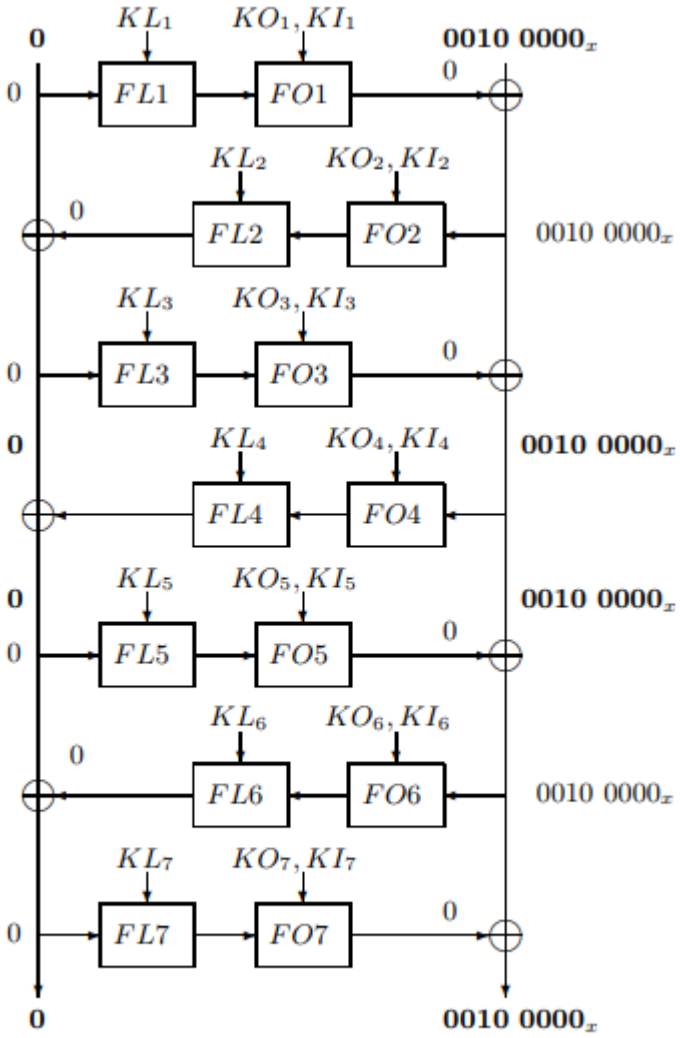
У наставку се може на основу описаног процеса реализовати напад са циљем да се одреди поткључ последње рунде. Потребно је анализирати пронађене квартете, тако да се анализом утврде улази у осму рунду алгоритма *KASUMI*, као и излази из ње. Кораци који би требало да се прођу су да се за сваки издвојени квартет шифрата (C_a, C_b, C_c, C_d), погоди вредност поткључа $KO_{8,1}$ и $KI_{8,1}$, који се користе у функцији *FO* последње, осме рунде *KASUMI* алгоритма.

Користећи парове шифрата (C_a, C_c), (C_b, C_d) и насумичне вредности поткључа, могу се добити вредности $KL_{8,2}$. За добре квартете података добиће се исте тј. одговарајуће вредности кључа ($KO_{8,1}$, $KI_{8,1}$, $KL_{8,2}$). На сличан начин могу се одредити и вредности $KL_{8,1}$ кључа који учествује у операцији битовског *"u"* унутар функције *FL* осме рунде, тако да се одреде ($KO_{8,3}$, $KI_{8,3}$, $KL_{8,1}$).

Процес се завршава одређивањем правог мастер кључа. За добијене 16-битне вредности поткључева који се користе по датим функцијама *KASUMI* алгоритма, ($KO_{8,1}$, $KI_{8,1}$, $KL_{8,2}$, $KO_{8,3}$, $KI_{8,3}$, $KL_{8,1}$), нападачу остаје да пронађе још преосталих $128 - 96 = 32$ бита кључа, тако што ће да погађа њихову вредност, те на крају да провери добијену вредност, извршавањем кода за шифровање са тим добијеним кључем.

У радовима [11] и [4] се процењује да је временска сложеност 2^{32}, због корака одређивања последња 32 бита кључа. Просторна сложеност одређена је потребом да се чувају велике количине шифрата и отворених текстова, тако да се за напад одваја 2^{30} бајтова (око 1 гигабајт меморије). Слика 4.1 илуструје описани напада

на *KASUMI* алгоритам.



Слика 4.1: Илустрација напада на *KASUMI*

Глава 5

Закључак

У овом раду описан је систем *GSM*, посебно са аспекта безбедности. Изложена је спецификација криптографског алгоритма *KASUMI*, који се користи у систему *GSM*. Описан је сендвич напад - криптоаналитички напад на који алгоритам *KASUMI* није отпоран. Сендвич напад настао је развојем бумеранг напада, који је пак настао од диференцијалне криптоанализе; због тога су у раду приказани и бумеранг напад и диференцијална криптоанализа. Описан је и реализован статистички препознавач, који на основу одређеног броја порука и њихових шифрата може да установи да ли су шифрати добијени применом алгоритма *KASUMI*. Сендвич напад на *KASUMI*, који представља доста сложенији процес, само је описан у раду.

Додатак А

Код имплементације алгоритма *KASUMI* из спецификације *3GPP*

Header fajl - zaglavlje

```
/*--------------------------------------------------------
 * Kasumi.h
 *--------------------------------------------------------*/
typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned long u32;

void KeySchedule( u8 *key );
void Kasumi( u8 *data );
```

C kod

```
/*--------------------------------------------------------------------------
 * Kasumi.c
 *--------------------------------------------------------------------------
 *
 * Jednostavna implementacija algoritma KASUMI
 * 3GPP Confidentiality and Integrity algorithms.
 *
 *
 * Verzija 1.1 08 May 2000
 *
 *--------------------------------------------------------------------------*/

#include "Kasumi.h"
```

```
/*--------- 16 bit pomeranje na levo
    ----------------------------------------*/

#define ROL16(a,b) (u16)((a<<b)|(a>>(16-b)))

/*---------- unije
    ------------------------------------------------------------*/
typedef union {
   u32 b32;
   u16 b16[2];
   u8 b8[4];
} DWORD;

typedef union {
    u16 b16;
   u8 b8[2];
} WORD;

/*-------- globalne promenljive ----------------------------------*/

static u16 KLi1[8], KLi2[8];
static u16 KOi1[8], KOi2[8], KOi3[8];
static u16 KIi1[8], KIi2[8], KIi3[8];

/*---------------------------------------------------------------------
 * FI()
 * FI funkcija koja sadrzi tabele S7, S9.
 * Transformisu 16-bitne vrednosti.
 *--------------------------------------------------------------------*/
static u16 FI( u16 in, u16 subkey )
{
   u16 nine, seven;
   static u16 S7[] = {
      54, 50, 62, 56, 22, 34, 94, 96, 38, 6, 63, 93, 2, 18,123, 33,
      55,113, 39,114, 21, 67, 65, 12, 47, 73, 46, 27, 25,111,124, 81,
      53, 9,121, 79, 52, 60, 58, 48,101,127, 40,120,104, 70, 71, 43,
      20,122, 72, 61, 23,109, 13,100, 77, 1, 16, 7, 82, 10,105, 98,
      117,116, 76, 11, 89,106, 0,125,118, 99, 86, 69, 30, 57,126, 87,
      112, 51, 17, 5, 95, 14, 90, 84, 91, 8, 35,103, 32, 97, 28, 66,
      102, 31, 26, 45, 75, 4, 85, 92, 37, 74, 80, 49, 68, 29,115, 44,
      64,107,108, 24,110, 83, 36, 78, 42, 19, 15, 41, 88,119, 59, 3};
   static u16 S9[] = {
      167,239,161,379,391,334, 9,338, 38,226, 48,358,452,385, 90,397,
      183,253,147,331,415,340, 51,362,306,500,262, 82,216,159,356,177,
      175,241,489, 37,206, 17, 0,333, 44,254,378, 58,143,220, 81,400,
      95, 3,315,245, 54,235,218,405,472,264,172,494,371,290,399, 76,
      165,197,395,121,257,480,423,212,240, 28,462,176,406,507,288,223,
      501,407,249,265, 89,186,221,428,164, 74,440,196,458,421,350,163,
```

```
        232,158,134,354, 13,250,491,142,191, 69,193,425,152,227,366,135,
        344,300,276,242,437,320,113,278, 11,243, 87,317, 36, 93,496, 27,
        487,446,482, 41, 68,156,457,131,326,403,339, 20, 39,115,442,124,
        475,384,508, 53,112,170,479,151,126,169, 73,268,279,321,168,364,
        363,292, 46,499,393,327,324, 24,456,267,157,460,488,426,309,229,
        439,506,208,271,349,401,434,236, 16,209,359, 52, 56,120,199,277,
        465,416,252,287,246, 6, 83,305,420,345,153,502, 65, 61,244,282,
        173,222,418, 67,386,368,261,101,476,291,195,430, 49, 79,166,330,
        280,383,373,128,382,408,155,495,367,388,274,107,459,417, 62,454,
        132,225,203,316,234, 14,301, 91,503,286,424,211,347,307,140,374,
        35,103,125,427, 19,214,453,146,498,314,444,230,256,329,198,285,
        50,116, 78,410, 10,205,510,171,231, 45,139,467, 29, 86,505, 32,
        72, 26,342,150,313,490,431,238,411,325,149,473, 40,119,174,355,
        185,233,389, 71,448,273,372, 55,110,178,322, 12,469,392,369,190,
        1,109,375,137,181, 88, 75,308,260,484, 98,272,370,275,412,111,
        336,318, 4,504,492,259,304, 77,337,435, 21,357,303,332,483, 18,
        47, 85, 25,497,474,289,100,269,296,478,270,106, 31,104,433, 84,
        414,486,394, 96, 99,154,511,148,413,361,409,255,162,215,302,201,
        266,351,343,144,441,365,108,298,251, 34,182,509,138,210,335,133,
        311,352,328,141,396,346,123,319,450,281,429,228,443,481, 92,404,
        485,422,248,297, 23,213,130,466, 22,217,283, 70,294,360,419,127,
        312,377, 7,468,194, 2,117,295,463,258,224,447,247,187, 80,398,
        284,353,105,390,299,471,470,184, 57,200,348, 63,204,188, 33,451,
        97, 30,310,219, 94,160,129,493, 64,179,263,102,189,207,114,402,
        438,477,387,122,192, 42,381, 5,145,118,180,449,293,323,136,380,
        43, 66, 60,455,341,445,202,432, 8,237, 15,376,436,464, 59,461};

    /* Sesnaestobitni ulaz se deli na dve nejednake polovine, *
    * od devet i setam bita - kao i potkljuc */

    nine = (u16)(in>>7);
    seven = (u16)(in&0x7F);

    /* Operacije transformacija */

    nine = (u16)(S9[nine] ^ seven);
    seven = (u16)(S7[seven] ^ (nine & 0x7F));

    seven ^= (subkey>>9);
    nine ^= (subkey&0x1FF);

    nine = (u16)(S9[nine] ^ seven);
    seven = (u16)(S7[seven] ^ (nine & 0x7F));

    in = (u16)((seven<<9) + nine);

    return( in );
}
```

```
/*---------------------------------------------------------------------
 * FO()
 * FO funkcija.
 * Transformise 32-bitnu vrednost. Promenljiva <index>
 * odredjuje potljuc koji se koristi.
 *---------------------------------------------------------------------*/

static u32 FO( u32 in, int index )
{
    u16 left, right;

    /* Ulaz se deli na dve 16-bitne reci */

    left = (u16)(in>>16);
    right = (u16) in;

    /* Transformacija, koja se pusta tri puta */

    left ^= KOi1[index];
    left = FI( left, KIi1[index] );

    left ^= right;

    right ^= KOi2[index];
    right = FI( right, KIi2[index] );
    right ^= left;

    left ^= KOi3[index];
    left = FI( left, KIi3[index] );
    left ^= right;

    in = (((u32)right)<<16)+left;

    return( in );
}

/*---------------------------------------------------------------------
 * FL()
 * FL funkcija.
 * Transformise 32-bitne vrednosti. Koristi se <index>
 * za odgovarajuci potkljuc koji ce da se koristi.
 *---------------------------------------------------------------------*/

static u32 FL( u32 in, int index )
{
    u16 l, r, a, b;
```

```
    /* Ulaz se deli na levi i desni deo */

    l = (u16)(in>>16);
    r = (u16)(in);

    /* odgovarajuce operacije */

    a = (u16) (l & KLi1[index]);
    r ^= ROL16(a,1);

    b = (u16)(r | KLi2[index]);
    l ^= ROL16(b,1);

    /* Na pocetku napravljene polovine se spajaju */

    in = (((u32)l)<<16) + r;

    return( in );
}


/*---------------------------------------------------------------------
 * Kasumi()
 * Glavni algoritam. Iste operacije se primenjuju cetiri puta.
 * Transformise 64-bitni ulaz.
 *---------------------------------------------------------------------*/

void Kasumi( u8 *data )
{
    u32 left, right, temp;
    DWORD *d;
    int n;

    /* Ulaz se deli na dva 32-bitna dela */

    d = (DWORD*)data;
    left = (((u32)d[0].b8[0])<<24)+(((u32)d[0].b8[1])<<16)
+(d[0].b8[2]<<8)+(d[0].b8[3]);
    right = (((u32)d[1].b8[0])<<24)+(((u32)d[1].b8[1])<<16)
+(d[1].b8[2]<<8)+(d[1].b8[3]);
    n = 0;
    do{    temp = FL( left, n );
        temp = FO( temp, n++ );
        right ^= temp;
        temp = FO( right, n );
        temp = FL( temp, n++ );
        left ^= temp;
    }while( n<=7 );
```

```
    /* Vraca odgovarajuci rezultat */

    d[0].b8[0] = (u8)(left>>24); d[1].b8[0] = (u8)(right>>24);
    d[0].b8[1] = (u8)(left>>16); d[1].b8[1] = (u8)(right>>16);
    d[0].b8[2] = (u8)(left>>8); d[1].b8[2] = (u8)(right>>8);
    d[0].b8[3] = (u8)(left); d[1].b8[3] = (u8)(right);
}

/*---------------------------------------------------------------------
 * KeySchedule()
 * Kreirnje potkljuceva. Vrednosti su 16-bitne.
 *---------------------------------------------------------------------*/

void KeySchedule( u8 *k )
{
    static u16 C[] = {
        0x0123,0x4567,0x89AB,0xCDEF, 0xFEDC,0xBA98,0x7654,0x3210 };
    u16 key[8], Kprime[8];
    WORD *k16;
    int n;

    /* Osigurava se da su potkljucevi 16-bitni */

    k16 = (WORD *)k;
    for( n=0; n<8; ++n )
        key[n] = (u16)((k16[n].b8[0]<<8) + (k16[n].b8[1]));

    /* Kreiranje kluceva K'[] */

    for( n=0; n<8; ++n )
        Kprime[n] = (u16)(key[n] ^ C[n]);

    /* Kreiranje odgovarajucih potkljuceva */

    for( n=0; n<8; ++n )
    {#define ROL16(a,b) (u16)((a<<b)|(a>>(16-b)))
        KLi1[n] = ROL16(key[n],1);
        KLi2[n] = Kprime[(n+2)&0x7];
        KOi1[n] = ROL16(key[(n+1)&0x7],5);
        KOi2[n] = ROL16(key[(n+5)&0x7],8);
        KOi3[n] = ROL16(key[(n+6)&0x7],13);
        KIi1[n] = Kprime[(n+4)&0x7];
        KIi2[n] = Kprime[(n+3)&0x7];
        KIi3[n] = Kprime[(n+7)&0x7];
    }
}
/*---------------------------------------------------------------------
```

```
* e n d o f k a s u m i . c
*--------------------------------------------------------------------*/
```

Литература

[1] „3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, 3G Security, Specification of the A5/3 Encryption Algorithms for GSM and ECSD, and the GEA3 Encryption Algorithm for GPRS; Document 4:Design and evaluation report; Document 2: Kasumi specification". 2010.

[2] Alex Biryukov. „The Boomerang Attack on 5 and 6-round Reduced AES, Proceedings of AES4 Conference, Lecture Notes in Computer Science". 2004.

[3] Noureddine Boudriga. *Security of Mobile Communications*. CRC Press, 2010.

[4] Mark Blunden, Adrian Escott. „Related Key Attacks on Reduced Round KASUMI, proceedings of Fast Software Encryption 2001, Lecture Notes in Computer Science 2355". 2002.

[5] H. M. Heys. „A Tutorial on Linear and Differential Cryptanalysis".

[6] Jongsung Kim, Guil Kim, Seokhie Hong, Dowon Hong. *The Related-Key Rectangle Attack — Application to SHACAL-1, proceedings of Australasian Conference on Information Security and Privacy 2004, Lecture Notes in Computer Science 3108*. Springer, 2004.

[7] Alexander Kukushkin. *Introduction to Mobile Network Engineering, GSM, 3G-WCDMA, LTE and the Road to 5G*. Wiley, 2018.

[8] Alex Biryukov, Adi Shamir, David Wagner (auth.), Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Bruce Schneier (eds.) „Fast Software Encryption 7th Internat, Lecture Notes in Computer Science 1978". 2004.

[9] Lars R. Knudsen, Matthew J.B. Robshaw. *The Block Cipher Companion*. Springer-Verlag Berlin Heidelberg, 2011.

[10] Shiguo Lian, Andreas U. Schmidt. *Security and Privacy in Mobile Information and Communication Systems, First International ICST Conference, MobiSec 2009, Turin, Italy*. Springer, 2009.

[11] Orr Dunkelman, Nathan Keller, Adi Shamir. „A Practical-Time Attack on the A5/3 Cryptosystem Used in Third Generation GSM Telephony". 2010.

[12] Orr Dunkelman, Nathan Keller, Adi Shamir. *A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. In: Rabin, T. (eds) Advances in Cryptology – CRYPTO 2010. CRYPTO 2010. Lecture Notes in Computer Science, vol 6223. Springer, Berlin, Heidelberg.* 2010.

[13] D. Wagner. „“The boomerang attack in Fast Software Encryption, vol. 1636 of Lecture Notes in Computer Science”. 1999.

[14] Claude E. Shannon, Warren Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, 1963.

[15] Miodrag Živković. „Kriptografija, skripta”. 2020. URL: http://poincare.matf.bg.ac.rs/~ezivkovm/nastava/kripto.pdf.