УНИВЕРЗИТЕТ СИНГИДУНУМ

БЕОГРАД

ДЕПАРТМАН ЗА ПОСЛЕДИПЛОМСКЕ СТУДИЈЕ И МЕЂУНАРОДНУ САРАДЊУ

ДОКТОРСКА ДИСЕРТАЦИЈА

ЈЕДНО РЕШЕЊЕ ЗА ЗАШТИТУ ИЗВОРНОГ КОДА СКРИПТ ЈЕЗИКА НА БАЗИ КРИПТОГРАФСКИХ МЕХАНИЗАМА

МЕНТОР

Проф. Др Младен Веиновић

КАНДИДАТ

Ненад Ристић

Београд, 2016. година

Списак слика

Списак табела

1 Увод

а) Предмет истраживања

Предмет истраживања овог рада чини анализа могућности за заштиту програмских решења, намењених за извршавање у серверским Веб окружењима. Веб окружење представља, по питању заштите, једно од најизазовнијих извршних окружења с обзиром на то да аутор решења често нема никакву везу, нити контролу над њим. Под заштитом се првенствено подразумевају:

- обезбеђивање да се програмско решење извршава само у свом изворном (неизмењеном), потпуном облику и
- обезбеђивање да се програмско решење извршава само у продукционим окружењима за која је аутор дао сагласност.

Посебан фокус рада чини анализа могућности заштите решења заснованих на интерпретерским језицима. Као основна за илустровање актуелних проблема, као и анализу постојећих решења у овом раду, коришћен је *PHP* програмски језик. Овакав избор је направљен из неколико разлога:

- *PHP* је данас изузетно популаран језик и процењује се да је на њему засновано преко 80% Веб сајтова [1].
- *PHP* је изворно интерпретерски језик, али су касније укључени и други концепти, као што су превођење, опкод, *JIT* и виртуална машина, тако да данас постоји више активних имплементација, углавном са јавно доступним изворним кодом.

Нови приступ у овом раду ће бити сагледавање проблема и решења заштите софтверских решења у Веб окружењима кроз призму стандардног криптолошког модела. Анализом оваквог приступа треба утврдити ефикасност у анализи могућности примене компонената са отвореним изворним кодом у креирању поузданог извршног Веб окружења, односно окружења у којем ће бити могуће извршавање софтверског решења заштићеног у претходно дефинисаном смислу. Применом овог модела требало би прошити основни контекст анализе, односно анализу на различитим нивоима превођења и извршавања изворног кода, укључујући реверзни инжењеринг

као средство подразумевано доступно нападачу у покушају нарушавања заштите. Посебан теоријски допринос оваквог приступа може се огледати у постојећим анализама, проналажењу стриктног придржавања овог модела у анализи решења из поменутог домена.

Као претходница развоју сопственог модела решења, поред анализе теоријских модела и принципа, у овом раду ће бити извршена детаљна анализа постојећих приступа у овој области. Примарно, биће анализирана три основна приступа заштити изворног кода - маскирање, шифровање и превођење у језике нижих нивоа. Додатно, кроз описани стандардни криптолошки модел биће анализирана постојећа комерцијална решења, односно утврђивање нивоа заштите који она нуде. Приликом анализе постојећих решења посебна ће бити обрађени додатни проблеми који она проузрокују, пре свега зависност од произвођача решења, као и потреба за поновним третирањем изворног кода и његовом заменом у извршном окружењу приликом сваког значајнијег унапређивања интерпретера или виртуалне машине. Теоријска компонента доприноса овог дела рада огледати ће се у стандардизованој анализи постојећих решења, док његову практичну компоненту чине резултати поменуте анализе, односно јасно указивање на недостатке тих решења.

На основу утврђених проблема и недостатака, биће предложен сопствени модел слободног решења, са подразумевано јавно доступним изворним кодом, чији је циљ да одговори, или испита могућности одговарања на следеће захтеве:

- слободно, отворено решење које елиминише зависност од треће стране од поверења;
- елиминисање додатних проблема која уводе постојећа затворена решења;
- постизање истог или вишег нивоа заштите од онога који нуде постојећа затворена решења;
- илустровање суштинских проблема у остваривању заштите на овом пољу, гледано са криптолошког аспекта, кроз описани стандардни криптолошки модел.

Увођењем захтева за отвореност предложеног решења биће уведена потпуно нова димензија у решавању проблема, а остварени резултати, уз ослањање на резултате вршених анализа, чине централни научни и практични допринос овог рада.

Даље, у раду ће бити анализиране кључне карактеристике хомоморфне криптографије и рачунарства од поверења. Анализом постојећих ограничења ових приступа, односно остваривања практичне употребљивости и могућности увођења у Веб окружење, потребно је идентификовати да ли она могу допринети решењу проблема који се обрађује у овом раду. Додатно, на основу суштинских и архитектурних захтева рачунарства од поверења може се јасније поставити подлога за развој претходно дефинисаног поузданог извршног Веб окружења. Ове анализе и дефинисање могућности представљају посебан допринос у погледу дефинисања даљих праваца истраживања, са циљем остваривања прихватљивог нивоа заштите, чак и код примене најригорознијих криптоаналитичких напада.

b) Мотивација

Кључни мотив овог рада је развој новог модела решења за заштиту скрипт језика отвореног кода, на примеру заштите PHP скрипти. Ниво заштите предложеног решења обезбеђиваће једнак ниво заштите са постојећим комерцијалним решењима. Истраживање се заснива на детаљном проучавању резултата из литературе, експерименталној анализи, симулацији и компаративном поређењу са другим решењима. На крају ће се детаљно издвојити и анализирати сви доприноси ове тезе. Дубинска анализа постојећих решења за заштиту је предуслов за њихово унапређење и увођење нових приступа.

c) Циљеви и задаци истраживања

Заштита *PHP* изворног кода од нежељене употребе, копирања и измене је велики проблем данас.

Научни циљ истраживања овог рада је да испита могућности за заштиту софтверских (програмских) решења од неовлашћеног коришћења и измене гледано кроз призму класичних криптолошких модела. Посебан фокус представља Веб окружење, као једно од данас најчешће коришћених, сложене архитектуре и са учешћем великог броја компонената различитих аутора. Резултати ове анализе треба да омогуће објективну анализу ефикасности постојећих решења, као и да дају основу за предлагање новог решења које би одговорило на евентуалне недостатке постојећих решења.

Посебан циљ истраживања је спровођење свеобухватне анализе и класификовање постојећих решења за заштиту решења развијених коришћењем *PHP* језика. као и давање модела за развој отвореног решења које би нудило исти или виши ниво заштите.

d) Методе које ће се примењивати у истраживању

Методологија истраживања у овом раду обухватаће сложен и организован поступак полазећи од логичких начела и принципа по утврђеним фазама.

Ово истраживање ће се класификовати по општости као појединачно, по критеријуму својства предмета као теоријско-емпиријско, по времену које обухвата као трансверзално, по припадности науци као интердисциплинарно, по актуелности предмета као актуелно, по сврси као верификационо, а по улози у науци ово истраживање ће припадати акционим.

У истраживању и прикупљању примарних и секундарних података биће коришћене методе из групе основних, опште научних, посебних и из групе статистичких метода за прикупљање података. Ту спадају генеричко-развојне, системско-структурално-функционалне, дескриптивне методе и методе теоријске анализе и проучавања литературе и других извора теорије и праксе, синтезе, апстракције, генерализације и класификације, као и остале методе научног објашњења које ћемо применити за закључивање на основу индукције и дедукције.

Овај избор истраживачких метода је употребљен да се истраживање и ток истраживачког процеса у свим фазама, тј. идентификацији и дефинисању проблема, планирању дизајна истраживања, сакупљању, обради и анализи података, као и формулацији закључака коректно спроведе у складу са основним принципима научно истраживачког рада.

Применом ових метода могуће је валидно остварење научног и друштвеног циља истраживања. Приступ истраживању је интегративан, синтетички у том смислу што се ни једном методолошком поступку не даје искључива предност, већ су подједнако заступљени.

e) Очекивани допринос

Имплементација новог модела заштите изворног кода који се базира на анализи слабости постојећих комерцијалних решења. У предложеном решењу заштите извршиће се интеграција познатих метода које се могу наћи у доступној литератури.

Побољшање нивоа заштите изворног кода PHP апликација.

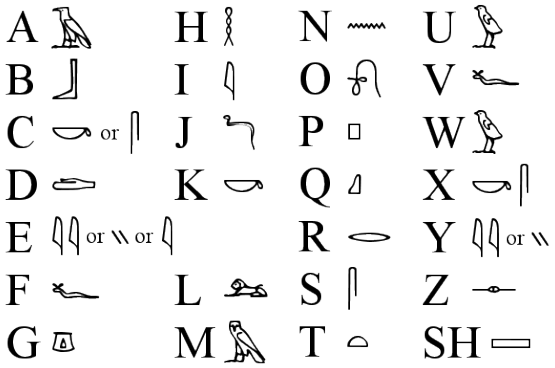
Реализоваће се решење у које се може имати поверење, са јасним параметрима и квалитетом шифрарског алгоритма и дужинама коришћених шифарских кључева.

Резултати рада на овој докторској дисертацији објављени су у више радова у часописима међународног значаја и саопштени на више научних скупова у земљи и иностранству.

2 Теоријске и криптолошке основе и оквири

2.1 Криптографија

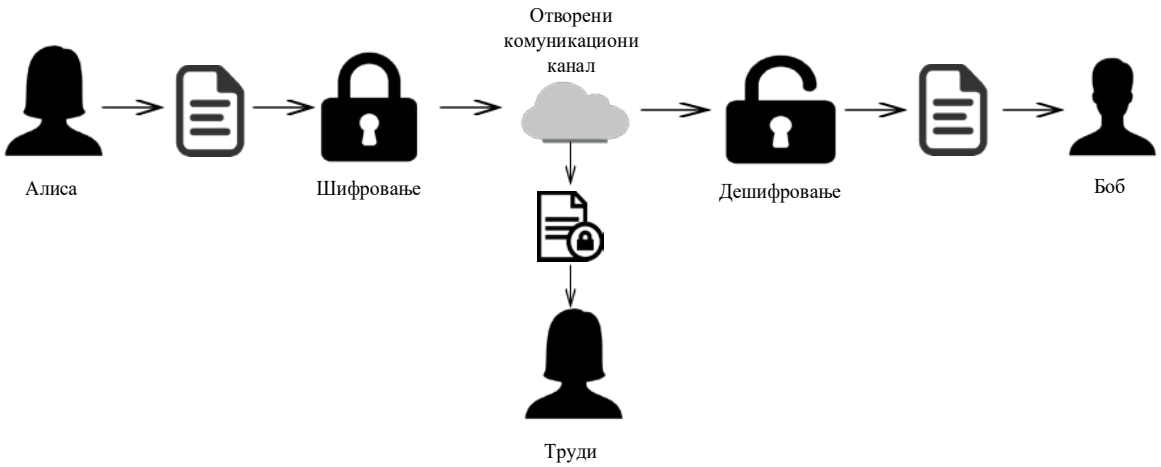
Криптографија је процес који има намену сакривања порука, информација или података и њихово конвертовање из читљивог у нечитљиви облик за све сем оног коме је порука намењена. На свом одредишту порука мора бити дешифрована да би било могуће тумачење од стране примаоца. На почецима употребе шифровања, људи су скривали тајну информацију заменом дела информације симболима, сликама или бројевима. Основни задатак криптографије је да обезбеди поверљивост употребом одређене методе прикривања. Употреба криптографије на самом почетку је била ограничена на комуникацију у војним структурама. Због потребе за брзим шифровањем и дешифровањем порука коришћени су једноставни системи замене или транспозиције [2].



Слика 1. Египатски хијероглифи

Анализа криптографије може бити извршена кроз комуникацију између два учесника (Алиса и Боб). Алиса жели да пошаље поруку која је намењена Бобу, при томе користећи канал комуникације који није заштићен. Канал комуникације може бити телефонска линија или рачунарска мрежа. Проблем се појављује ако је садржај поруке поверљив. Порука може бити пресретнута, прочитана или чак измењена од треће стране (Труди) без знања примаоца поруке (Боб). Порука коју треба послати од стране Алисе у незаштићеном формату називамо отворени текст (енгл. *Plaintext*), ова порука може садржати слику, текст, бројеве или неки други садржај. Потребно је да Алиса шифрује отворени текст да би он био сигурно прослеђен незаштићеним каналом комуникације. На страни пријема Боб добија шифровану поруку који дешифрује из

нечитљиве форме и добија отворени текст. У случају пресретања поруке од стране Труди порука није читљива јер је заштићена шифровањем.



Слика 2. Модел шифроване комуникације

Процес шифровања користи одређени алгоритам за шифровање (Е) и дешифровање (Д). Код употребе класичних шифарских система оба алгоритма Е и Д се ослањају на исти кључ који користе у свом процесу. Због овакве употребе кључа ови алгоритми носе назив симетричних шифарских система.

Објављивањем рада 1976. године под називом „Нови смерови у криптографији“ [3] представљен је нови концепт криптографије, криптографија јавног кључа. Овакав начин шифровања решавао је проблем размене кључа јер је у процесу шифровања кориштен је јавни кључ, док се у процесу дешифровања користи тајни кључ који се налази искључиво код примаоца поруке. Овакав начин шифровања због употребе два различита кључа у процесу шифровања и дешифровања носи назив асиметрични шифарски систем. Ново решење криптографије јавним кључем засновано на дискретним логаритмима и представљено је још 1985. године [4]. Највећи допринос криптографије јавног кључа је дигитални потпис [5]. Први међународни стандард дигиталног потписа усвојен је 1991. године и базиран је на РСА моделу криптографије јавног кључа [6]. Касније се усваја нови модел базиран на раније споменутом решењу са дискретним логаритмима [7].

Анализирајући све шифрарске системе, симетрични шифарски системи нуде најбржу и најједноставнију примену како на хардверу тако и у различитим софтверским решењима. Управо због лаке имплементације симетрични шифарски системи су

погодни за шифровање велике количине података. Ако Алиса и Боб желе да користе симетрично шифровање потребно је да пре тога изврше размену тајног кључа. Размена кључа мора се вршити сигурним комуникационим каналом. За размену кључа се најчешће користе асиметрични шифарски системи. Асиметрични шифарски системи нису погодни за шифровање велике количине података, али имају ефикасно решење које решава проблем сигурне размене кључа. Управо због ових карактеристика симетрични и асиметрични шифарски системи употпуњују се и заједно могу обезбедити практичан и сигуран систем шифровања.

Између шифарских система можемо направити разлику по типу на блоковске и секвенцијалне шифрарске системе [8]. Функција шифровања блоковског шифрарског система врши операције над отвореним текстом фиксне дужине. У случају да је дужина отвореног текста већа од блока за шифровање примењују се различити начини рада који омогућавају успешно шифровање. Секвенцијални шифарски систем врши операције над низом отвореног текста карактер по карактер, при чему низови отвореног текста су произвољне дужине.

Поред основне улоге криптографије да обезбеди поверљивост, то није једина намена самог процеса шифровања. Криптографија помаже у решавању проблема интегритета података, потврђивања идентитета учесника у комуникацији као и непорецивости. Поверљивост представља ограничавање садржаја искључиво на овлаштене учеснике у комуникацији. Проблем интегритета података представља заштиту података од неовлашћене измене. Потврђивања идентитета је процес идентификовања учесника или порекла података у комуникацији. Последњи проблем који би криптографија требала да покрије је непорецивост. Непорецивост спречава одрицање одговорности учесника у комуникацији за почињени поступак или радњу.

2.1.1 Класични шифарски системи

Од давнина, како постоји комуникација постоји потреба за скривеном или тајном комуникацијом. Ови првобитни шифарски системи проналаском рачунара су застарили и немају практичну примену, али поседују одређене принципе који се примењују и у модерним шифарским алгоритмима.

Класичне шифрарске системе према начину на који функционишу можемо поделити у следеће групе.

- Шифарски систем мешања (транспозиције)
- Шифарски систем замене (супституције)

Шифарски системи који раде на принципу мешања карактера, премештају места карактерима и на тај начин се формира се шифровани текст. У периодима када се користио овај систем постојао је велики број делимично неписмених људи па је овакав једноставан систем био поприлично ефикасан. Најпознатији представник шифрарског система транспозиције је Скитала који су користили Спартаци 500 године. Принцип шифровања је једноставан. Обавија се кожна трака око штапа те се исписује одређена порука. Након размотавања траке добија се измешани низ карактера који представља шифровану поруку.

Слика 3. Скитала

Додатно јачање овог принципа шифровања је могуће уз додавање одређене кључне речи. Кључна реч се користи за одређивање редоследа мешања карактера. Одређује се позиција карактера кључне речи у алфабету и на основу редоследа се врши мешање по колонама. На примеру имам приказан принцип шифровања скиталом. Текст за шифровање је: „НенадРистићСинергијаБијељина“, добијени шифрат је „Нсеи етрј хиге ањиљ дСји Риан инБа“

Табела 1. Текст за шифровање

Н	е	н	а	д	Р	и
с	т	и	ћ	С	и	н
е	р	г	и	ј	а	Б
и	ј	е	љ	и	н	а

Ако на претходном примеру применимо додатак кључне речи „*студент*“ који претворен у позиције у азбуци даје 21, 22, 24, 5, 7, 16, 22. Као резултат добија се шифрат: „ањиљ дСји Риан Нцеи етрј инБа“

Табела 2. Текст за шифровање уз додатак кључне речи

с-21	т-22	у-24	д-5	е-7	н-16	т-22
Н	е	н	а	д	Р	и
с	т	и	ћ	С	и	н
е	р	г	и	ј	а	Б
и	ј	е	љ	и	н	а

Шифарски системи у којима се врши замена карактера другим карактерима називају се системи замене или супституције. У системима супституције распоред карактера се не мења али се мења карактер пресликавањем са другим карактером. Најпознатији представник система шифровања заменом је Цезарова шифра. Цезарова шифра врши замену карактера из поруке заменом са карактером који се налази три места даље од по алфабетском распореду.

А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш
Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В

Слика 4. Цезарова шифра на азбуци

У оваквом начину шифровања кључ је величина за коју се врши померање у алфабету. Случај Цезарове шифре кључ је 3. Наследник Јулија Цезара, Аугустов је извршио мању измену променом кључа на 1, и изменом последњег карактера римског алфабета на „АА“ [9]. Велико унапређење у систему шифровања заменом је увођење хомоморфног система замене. Хомоморфни систем замене унапређује систем просте

замене и ојачава ниво отпорности на криптоанализу. Хомоморфни систем замене уводи додатне карактере, самим тиме одређени карактери могу бити замењена и представљена на више начина [10].

Цезарова шифра је поприлично слаб систем шифровања јер постоји 26 могућих кључева за шифровање. Поготово уз данашњу технологију тестирање 26 кључева представља једноставан задатак. Чак и ојачавањем система замене или супституције такође постоји подложност нападу статистичке анализе фреквенције карактера поруке.

Систем шифровања Цезаровом шифром се назива и моноалфабетски систем замене. Још једно ојачање систему замене је полиалфабетска замена. Ово је систем у ком се користи вишеструка замена чиме се постиже већа отпорност на напад анализе фреквенције карактера. Типичан представник полиалфабетске замене је Вижнерова шифра.

Вижнерова шифра је развијена 1553. године од стране Ђована Бастиста (енгл. *Giovan Battista*) али тек касније ушао у употребу, и приписан је Блез де Вижнеру (енгл. *Blaise de Vigenère*). Вижнерова шифра се заснива на Цезар шифри. Разлика је да Вижнер не користи један кључ већ кључну реч која се користи као посебан кључ за сваки карактер засебно. У случају да је реч краћа од текста који се шифрује реч се наставља понављањем док се не добије жељена дужина. Због оваквог начина рада Вижнер шифра представља вишеструко Цезар шифровање.

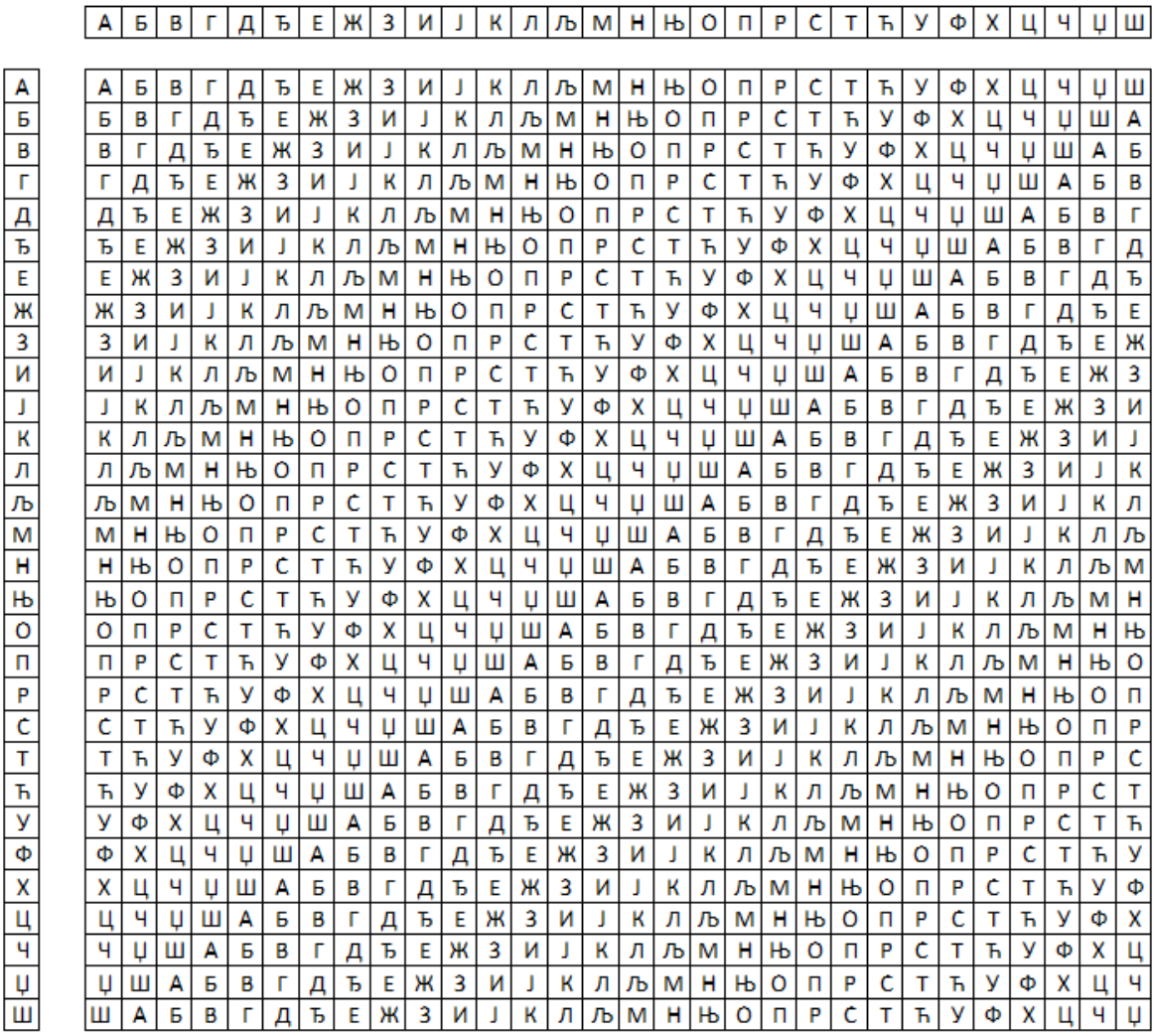
Н	Е	Н	А	Д	Р	И	С	Т	И	Ћ
С	И	Н	Е	Р	Г	И	Ј	А	Б	Н
Е	Њ	Б	Ж	Ф	У	Р	Б	Ћ	К	З

мод(30)

16	7	16	1	5	20	10	21	22	10	23
21	10	16	7	20	4	10	11	1	2	16
7	17	2	8	25	24	20	2	23	12	9

Слика 5. Пример Вижнер шифра

На примеру се види шифровање Вижнеровом шифром. Први ред представља отворени текст, други ред је кључ и последњи ред је шифрат. Вижнерова шифра се може користити тако што се карактери претварају у нумеричке вредности позиције коју заузимају у алфабету, али може се користити и Вижнерова матрица.



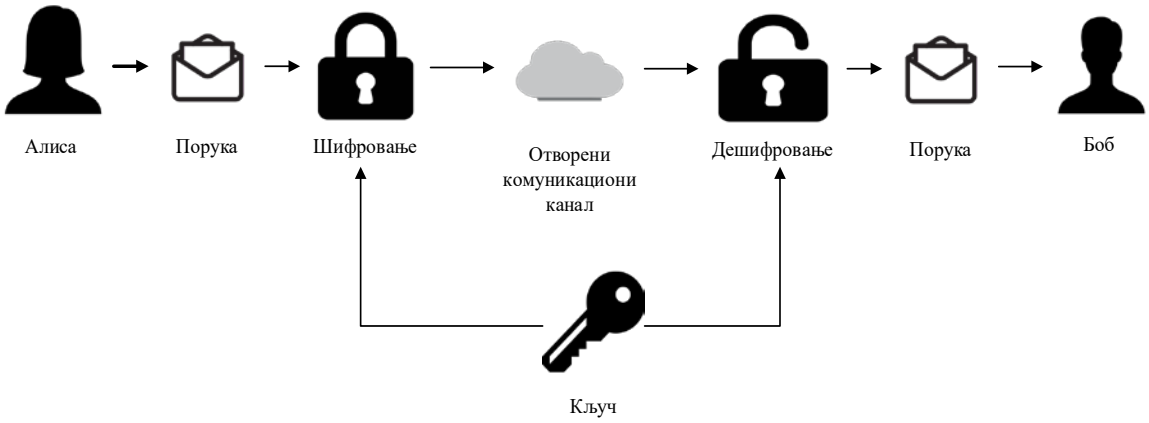
	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш
А	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш
Б	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А
В	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б
Г	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В
Д	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г
Ђ	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д
Е	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ
Ж	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е
З	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж
И	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З
Ј	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И
К	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј
Л	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К
Љ	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л
М	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ
Н	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М
Њ	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н
О	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ
П	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О
Р	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П
С	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р
Т	Т	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С
Ћ	Ћ	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т
У	У	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ
Ф	Ф	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У
Х	Х	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф
Ц	Ц	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х
Ч	Ч	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц
Џ	Џ	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч
Ш	Ш	А	Б	В	Г	Д	Ђ	Е	Ж	З	И	Ј	К	Л	Љ	М	Н	Њ	О	П	Р	С	Т	Ћ	У	Ф	Х	Ц	Ч	Џ

Слика 6. Вижнерова матрица

Због могућности да исти карактери отвореног текста буду шифровани различитим кључем Вижнерова шифра представља знатно тежи систем за криптоанализу од других моноалфабетских система шифровања.

2.1.2 Симетрични шифарски системи

Симетрични шифарски системи представљају групу шифарских система који у процесу шифровања и дешифровања користе исти, заједнички тајни кључ. Размена тајног кључа се врши путем заштићеног канала док се шифрована порука шаље отвореним каналом комуникације.



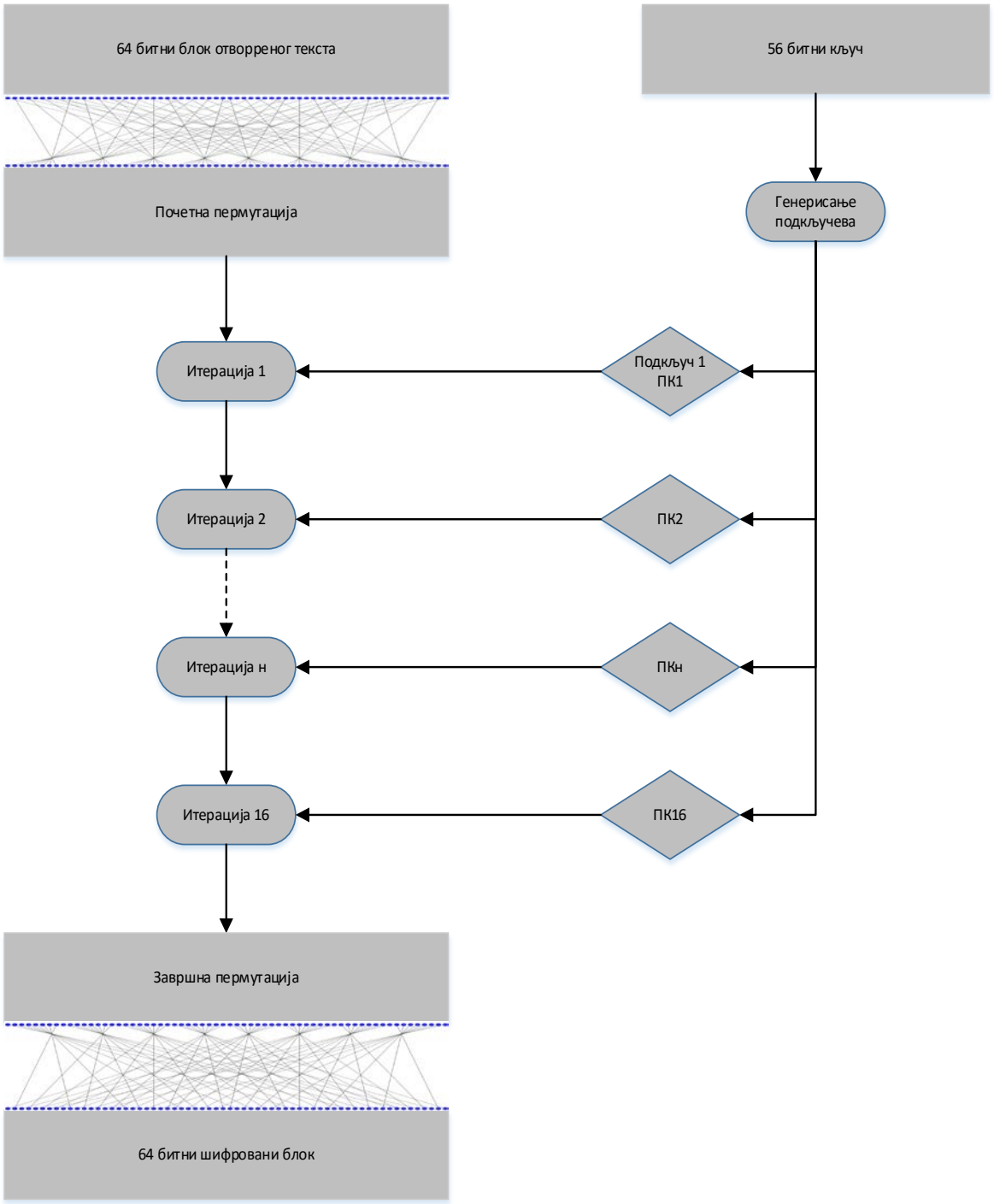
Слика 7. Модел симетричног шифровања

На слици је приказан графички приказ симетричног шифровања. Порука на страни пошиљаоца се шифрује и затим шаље кроз отворени комуникациони канал. Након пријема порука се дешифрује за примаоца употребом идентичног кључа који је употребљен за шифровање. Два најбитнија коришћена симетрична алгоритма за шифровање су ДЕС (енгл. *Data Encryption Standard*) и АЕС (енгл. *Advanced Encryption Standard*).

2.1.2.1 ДЕС алгоритам

ДЕС алгоритам је развијен средином 1970 године. Овај алгоритам дизајниран је од стране ИБМ за потребе конкурса америчког националног бироа за стандарде (енгл. *National Bureau of Standards*) за стандардизовани шифарски алгоритам. ИБМ је базирао ДЕС алгоритам на Луцифер шифри која је била ранији рад Хорста Фајстела [11]. Након консултовања са државном агенцијом за сигурност (енгл. *National Security Agency*) и мањих измена 1976 године прихваћено је решење ИБМ. Овим измењени Луцифер алгоритам постаје ДЕС. Једна од највећих измена на оригиналном Луцифер алгоритму била је дужина кључа. Предложено решење је имало дужину кључа од 128 бита, док је измењено, прихваћено решење имало дужину кључа од 64 бита. Детаљном анализом утврђено је да је стварна дужина кључа сведена на 56 бита јер је 8 бита кључа искоришћено за проверу парности [12]. Важна карактеристика ДЕС алгоритма је ефекат лавине, који условљава малом изменом улаза велику измену излаза. Овим се постиже да у измени једног бита блока отвореног текста или кључа постиже се 50% измене шифрованог блока [10].

ДЕС представља блоковски систем шифровања. У процесу шифровања ДЕС користи блокове од 64 бита отвореног текста и кључ дужине 56 бита. Пре генерисања једног блока шифрата одвија се 16 итерација а дужина подкључа у свакој итерацији је 48 бита.



Слика 8. Шифровање блока ДЕС

На слици је приказан модел шифровања блока у ДЕС. У првом кораку врши се иницијална пермутација. Иницијална пермутација не утиче на безбедност шифровања али се претпоставља да је њена намена била тежа програмска имплементација тј. потреба за хардверском подршком [13]. На слици је приказан блок иницијалне пермутације, бројеви означавају положај улазних бита блока отвореног текста.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

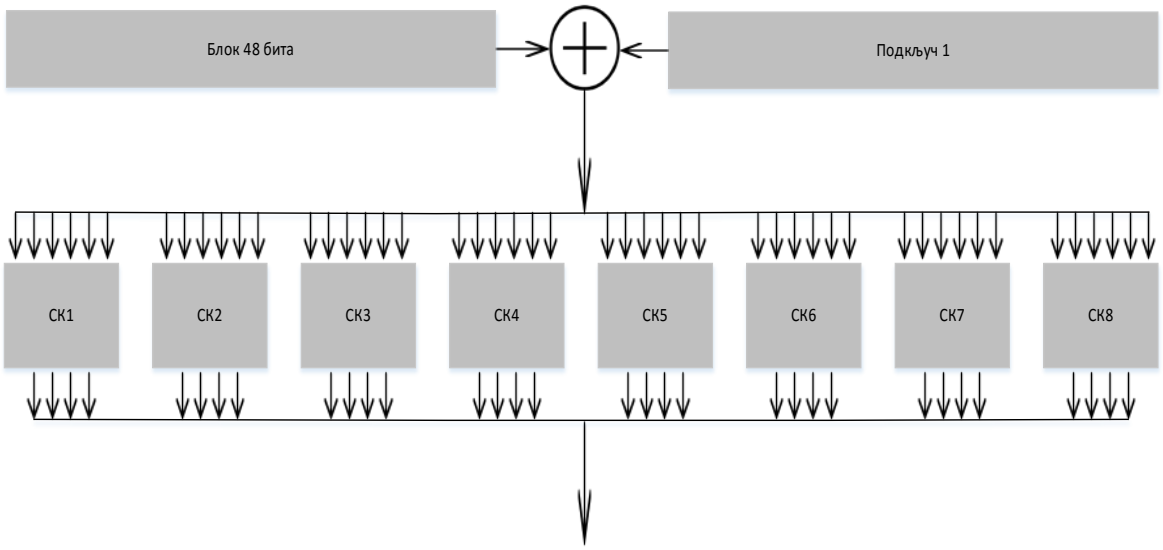
Слика 9. 64 битни блок иницијалне пермутације [13]

Након иницијалне пермутације врши се подела блока за шифровање на два дела, леви и десни величине по 32 бита. Као што је раније наведено кључ дужине 64 бита се своди на дужину од 56 бита употребом сваког осмог бита за проверу парности. Након издвајања 56 битног кључа генеришу се 48 бити кључеви за сваку од итерација.

Генерисање кључева за сваку итерацију се врши на следећи начин. На почетку се 56 бита кључа дели на две 28 битне половине. Затим се половине померају кружно за један или два бита у зависности од итерације. Након померања бира се 48 бита од 56 за подкључ. Следећа пермутација проширује десни блок података за шифровање са 32 бита на 48. Ова операција се назива пермутација проширивања (енгл. *Expansion Permutation*) и има две намене. Дужина десне половине постаје једнака дужини подкључа због операције ексклузивног или (енгл. *XOR*) и добија се дужи резултат који се може компресовати током операције замене.

Након операције ексклузивног или (енгл. *XOR*) са проширеним блоком прелази се на операцију замене. Операција замене се врши употребом осам кутија замене или с-кутија (енгл. *Substitution boxes*). Свака од кутија замене има улаз од 6 бита и излаз од 4 бита. Укупна количина меморије за 8 знаменских кутија је 256 бита. Блокови од по

48 бита се деле у осам подблокова дужине 6 бита. Сваким од блокова управља друга заменска кутија.



Слика 10. Операције с-кутија

Сваку заменску кутију чини табела од 4 реда и 16 колона. Садржај сваке од ћелија табеле је четворобитни број. Шест бита улаза одређује под којим редом и колоном се налази излаз.

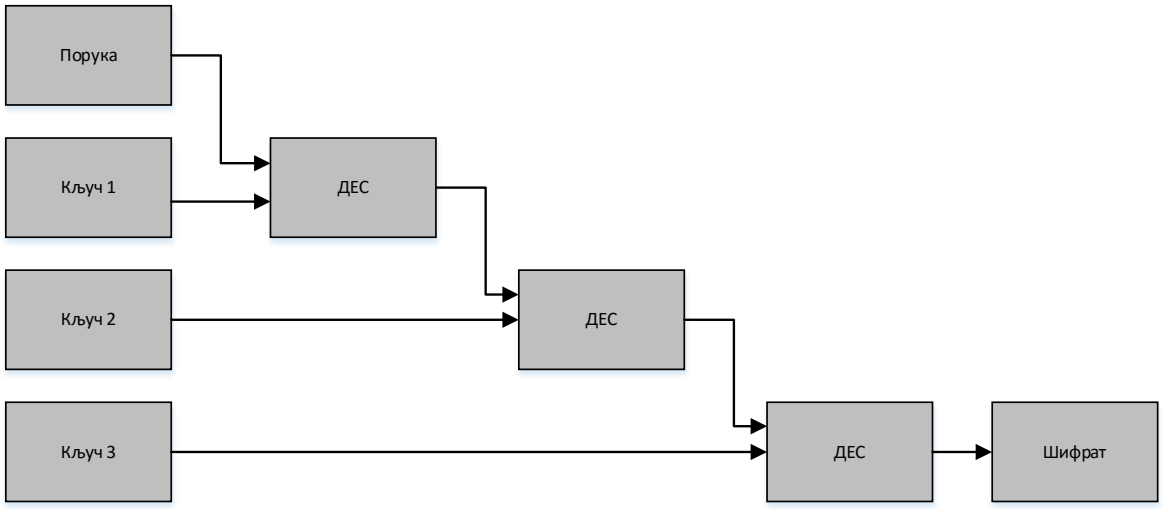
Улазни бити у с-кутије на прецизан начин дефинишу избор. Узимајући у обзир да имамо 6 бита улаза, означићемо их са од Б1 до Б6. Први и последњи бит Б1 И Б6 се комбинују на такав начин да формирају битни број величине од 0 до 3 тј. број дужине два бита. Овај број одговара одређеној колони у табели с-кутије. Преостали бити се комбинују тако да формирају бројеве дужине четири бита који одговарају колонама у табели тј. бројеви од 0 до 15.

Иако је на почетку дефинисања ДЕС-а највише коментара, поред скраћивања кључа, било због увођења с-кутија, управо с-кутије значајно ојачавају овај алгоритам. Сама безбедност ДЕС алгоритма довођена је у питање од самог почетка. Највећа забринутост је била умешаност државне агенције за безбедност у измене над оригиналним концептом алгоритма. Сматрано је да је агенција за безбедност уградила задња врата за процес дешифровања који би тој агенцији омогућио лако дешифровање. Чак и након преко тридесет година активне анализе није доказано да је агенција урадила ишта друго сем ојачала овај алгоритам. Главни криптографи који су развијали ДЕС алгоритам рекли су да је циљ државне агенције био да дефинише јаке

механизме замене, пермутације и операција над кључем [14]. Основна слабост DES-а је напад на кључ. Слабост у дужини кључа је умањена употребом троструког ДЕС шифровања [15].

Троструки ДЕС (енгл. Triple Data Encryption Algorithm) представља значајно ојачање изворног начина шифровања употребом ДЕС алгоритма. Комбиновањем три операције ДЕС-а уз употребу три кључа може се постићи значајан ниво сигурности.

Национални институт за стандарде и технологију (енгл. *National Institute of Standards and Technology*) дефинише три модела управљања кључевима у операцијама троструког ДЕС шифровања [16]. Први модел користи три различита кључа за свако шифровање. Други модел користи два кључа која се разликују док у трећем извршавању ДЕС шифровања поново користи други кључ. Последњи трећи модел користи три идентична кључа. У ревизији модела шифровања употребом троструког ДЕС дата је препорука да се последњи трећи модел не користи [15].



Слика 11. Приказ реализације троструки ДЕС

2.1.2.2 AEC алгоритам

Амерички национални институт за стандарде и технологију (енгл. *National Institute of Standards and Technology*) 1997. године расписује конкурс за нови стандард у шифровању [17]. За разлику од захтева за ДЕС, АЕС би требао да буде лак за имплементацију како на софтверској тако и на хардверској платформи. Још један важан критеријум је био шифровање блока дужине 128 бита уз измењиве дужине кључа. Значајна разлика у односу на ДЕС је био и велики број понуђених решења као

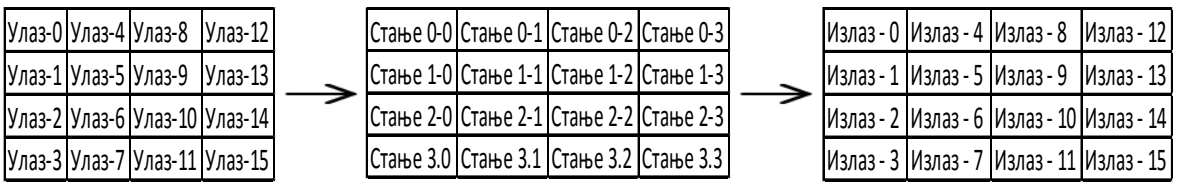
и то да је овај пут државна агенција за безбедност била само у улози једног од судија. У последњем кругу избора за најбољи алгоритам који ће постати стандард преостало је пет алгоритама [18]. Решење које је прихваћено је било решење два Белгијска криптографа Винсента Рејмена (енгл. *Vincent Rijmen*) и Џоуна Дајмена (енгл. *Joan Daemen*) под називом Рајндол (енгл. *Rijndael*). AEC (енгл. *Advanced Encryption Standard*) званично је прихваћен као стандард и објављен 26 новембра 2001. године [19].

Табела 3. Преглед завршних оцена за избор AEC

	Algorithm				
Категорија	MARS	RC6	Rijndael	Serpent	Twofish
Сигурност	3	2	2	3	3
Имплементација	1	1	3	3	2
Перформансе	2	2	3	1	1
Перформансе на паметним картицама	1	1	3	3	2
Перформансе на различитим платформама	1	2	3	3	2
Функционалности дизајна	2	1	2	1	3
	10	9	16	14	13

AEC је такође попут ДЕС блоковски шифарски систем. AEC има дужиny bloka za шифровање од 128 бита а кључ је промењиве дужине и може бити 128, 192 и 256 бита [19]. Једина разлика између AEC и оригиналног алгоритма Рајндол је у величини блока. Величина блока оригиналног алгоритма је промењива као и величина кључа на вредности од 128, 160, 192, 224 и 256 бита [20].

Улази и излази из AEC алгоритма су низови дужине 128 бита. Операције које се врше унутар AEC алгоритма обављају се над дводимензионалним низовима бајта под називом матрица стања (енгл. *State*). На почетку шифровања или дешифровања, улазни низ бајта се пресликава на матрицу стања, над којом се касније врше операције.



Слика 12. Пресликавање улаза на матрицу стања

Након додавања кључа итерације матрица стања пролази кроз итерације промене које се врше кроз 10, 12 или 14 понављања у зависности од дужине кључа. Кроз извршавање трансформација користе се четири функције замена бајта (енгл. *SubBytes),*

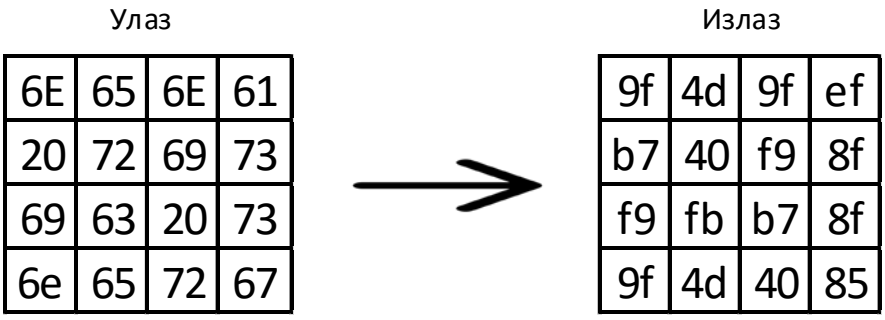
замена редова (енгл. *ShiftRows)*, мешање колона (енгл. *MixColumns)*, и додавање итерацијског кључа (енгл. *AddRoundKey)* [19]. Функције итерација су одређене темпирањем кључа који се састоји из једнодимензионалног низа изведеног из процедуре проширивања кључа.

Функција *SubBytes* је нелинеарна трансформација које се извршава независно на сваком бајту матрице стања употребом кутије замене (С-кутија). AEC алгоритам за разлику од ДЕС алгоритма има јасно дефинисан дизајн с-кутије. Дизајн с-кутије AEC дефинисан тако да спречи линеарну и диференцијалну криптоанализу [21].

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	1	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	4	c7	23	c3	18	96	5	9a	7	12	80	e2	eb	27	b2	75
	4	9	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	0	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	2	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	6	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	8
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	3	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

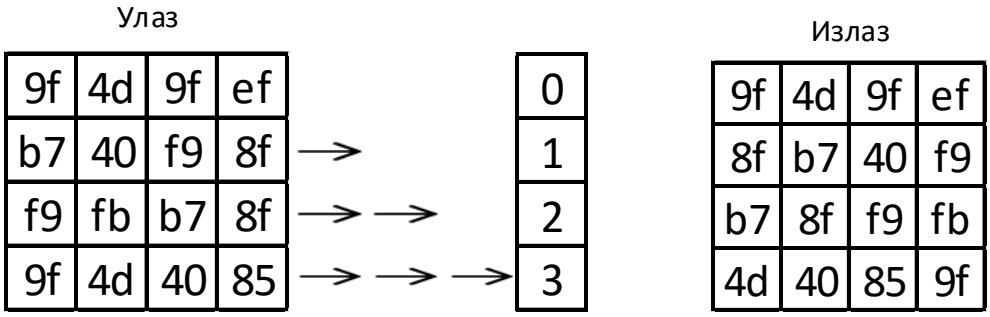
Слика 13. С-Кутија [19]

За матрицу улаза величине 4 пута 4 бајта у функцију дефинише се одговарајућа замена у С-кутији.



Слика 14. Приказ функције замене употребом с-кутије

Следећа трансформација је функција која врши померање редова матрице. Функција *ShiftRows* помера последња три реда бајта. Померање се врши у односу на позицију реда кружним померањем бита у десно.



Слика 15. Приказ функције померања редова

Функција мешања колона (енгл. *MixColumns*) извршава се над сваком колоном појединачно, третирајући сваку колону као полином.

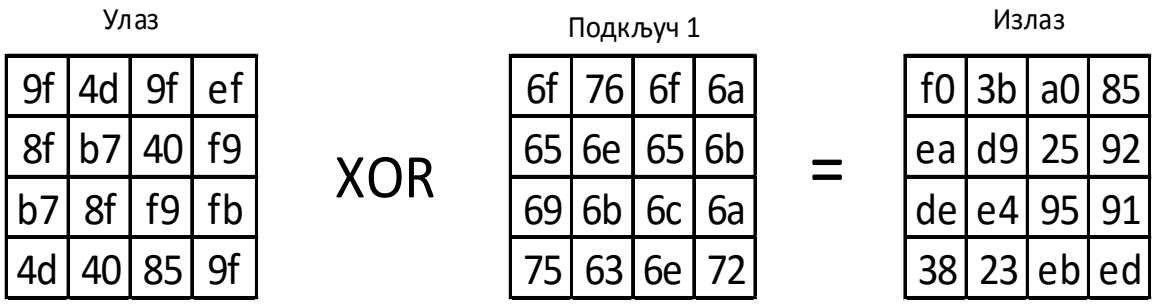
$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Слика 16. Функција мешања колона [22]

Током извршавања функције мешања колона врши се операција ексклузивно или (енгл. *XOR*) као и померање.

Последња функција је додавање генерисаног кључа итерације. Слично као у ДЕС алгоритму користи се алгоритам распореда кључа за генерисање подкључа за сваку

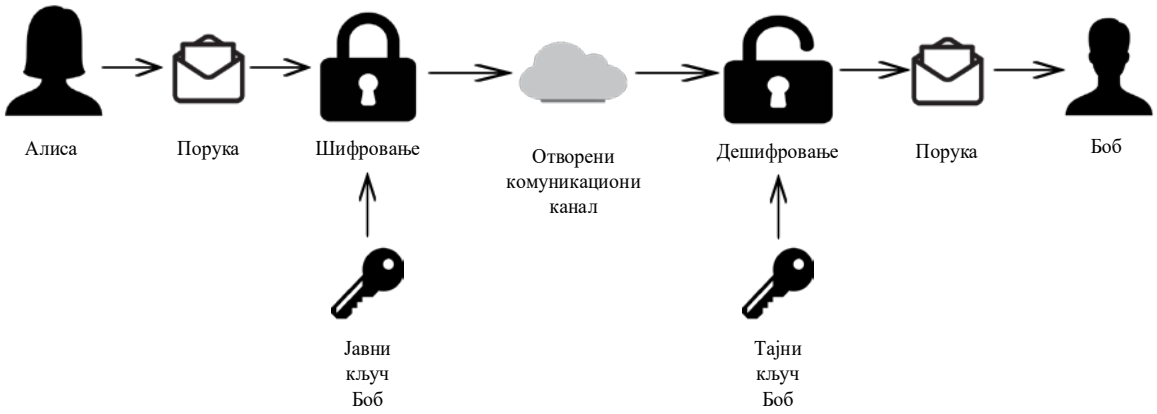
итерацију. Након генерисања подкључа извршава се операција ексклузивно или (енгл. *XOR*) између блока и подкључа.



Слика 17. Итерација са подкључем

2.1.3 Асиметрични шифарски системи

Асиметрични шифарски системи представљају шифрарске системе у којима се у процесу шифровања користи један, јавни кључ, док за процес дешифровања се користи други, тајни кључ. Учесници у комуникацији (Алиса и Боб) поседују свој тајни кључ који користе за дешифровање, док се за шифровање користе јавни кључеви учесника. Темељи за истраживање асиметричних шифарских система постављени су још 1976. године [3]. Тада су аутори рада „Нови смерови у криптографији“ представили могућност криптографије употребом јавног кључа. Нешто касније 1985. године Тахер Елгамал (енгл. *Taher Elgamal*) преставио је још једно решење криптографије јавним кључем и дигиталног потписа базирано на дискретним логаритмима [22].



Слика 18. Процес асиметрично шифровање

За успешну комуникацију учесници у комуникацији поседују по пар кључева, јавни кључ (K_j) који је познат свима и тајни кључ (K_m). Алиса зна Бобов јавни кључ и она шифрује поруку употребом јавног кључа Боба и алгоритма за шифровање. Јавни кључ се не може користити за дешифровање поруке. На страни пријема Боб користи свој тајни кључ за дешифровање поруке. Овакав начин шифровања и дешифровања решава један од највећих проблема симетричних шифарских система, а то је сигурна размена тајног кључа.

Принцип рада асиметрични шифарски система се заснива на једносмерним математичким функцијама. Ове једносмерне функције садрже и замку чија је улога да обезбеди да нападач не може лако или директно да израчуна тајни кључ из јавног кључа [23]. Факторизација чини значајан модел овакве једносмерне функције са замком (енгл. *One-way trapdoor functions*). Асиметрични шифарски системи су јако спори у поређењу са симетричним [24].

Важна директна примена асиметричног шифрарског система је дистрибуција кључева сесије. Кључ сесије је тајни кључ који се користи у класичном симетричном шифровању за процес шифровања порука у току једне сесије. Ако Алиса зна Бобов јавни кључ може га искористити за шифровање генерисаног кључа сесије. Шифрован кључ сесије се шаље Бобу, који користећи свој приватни кључ дешифрује кључ сесије. На овај начин извршена је сигурна размена тајног кључа који се користи за шифровање током сесије. Иницирањем нове сесије врши се нова размена и успоставља нова сесија.

Као најзначајнији асиметрични алгоритам биће размотрен РСА (енгл. *RSA*).

2.1.3.1 РСА (енгл. *RSA*)

Тренутно најзначајнији асиметрични алгоритам је РСА [10]. Име алгоритма је изведено од имена аутора Рон Ривест (енгл. *Ron Rivest*), Ади Шамир (енгл. *Adi Shamir*), и Леонард Адлемен (енгл. *Leonard Adleman*). Овај алгоритам се поред шифровања, успешно користи и за дигитално потписивање. РСА шифарски систем је заснован на теорији бројева која је позната преко 250 година [8].

Снага РСА алгоритма је у факторизацији великих бројева. Тајни и јавни кључ су резултати функције пара великих простих бројева. РСА укључује три функције:

генерисање кључа, шифровање и дешифровање. У процесу генерисања кључа добијају се јавни и тајни кључ. За почетак бирају се два проста броја, *б1* и *б2* затим се формира производ ова два броја.

$$\text{Н} = \text{б}\mathbf{1} * \text{б}\mathbf{2}$$

Затим се одређује експонент *е*. Овај број је потребно да буде мањи од φ(Н) где је

$$\boldsymbol{\varphi}(\text{Н}) = (\text{б}\mathbf{1} - \mathbf{1}) * (\text{б}\mathbf{2} - \mathbf{1})$$

Последња операција је генерисање приватног кључа д где је

$$\text{д} = \text{е}^{-\mathbf{1}}(\text{мод}\boldsymbol{\varphi}(\text{Н}))$$

Овим операцијама добијен је пар кључева. Јавни кључ (*Н,е*) и тајни или приватни кључ (д). Даље се ови параметри користе у процесу шифровања и дешифровања унутар функције модуларног експонента.

$$\text{Шифровање} = \text{порука}^{\text{е}} \text{ мод Н}$$

$$\text{Дешифровање} = \text{шифрат}^{\text{д}} \text{ мод Н}$$

Побољшање брзине извршавања алгоритма је могуће уз избор мањег броја *е*. Иако оваква измена убрзава извршавање потрено је обратити пажњу да се не користи превише кратак *е* за више истих порука које се шаљу за више корисника [6].

Сигурност РСА алгоритма ослања се на тежину факторизације броја *Н*, потребно је изабрати бројеве *б1* и *б2* на такав начин да је израчунавање *Н* потпуно немогуће. Напретком технологије и метода напредују и напади на факторизацију. Последњи успешно факторизован број је дужине 232 или РСА-768 [25].

2.2 Хомоморфна криптографија

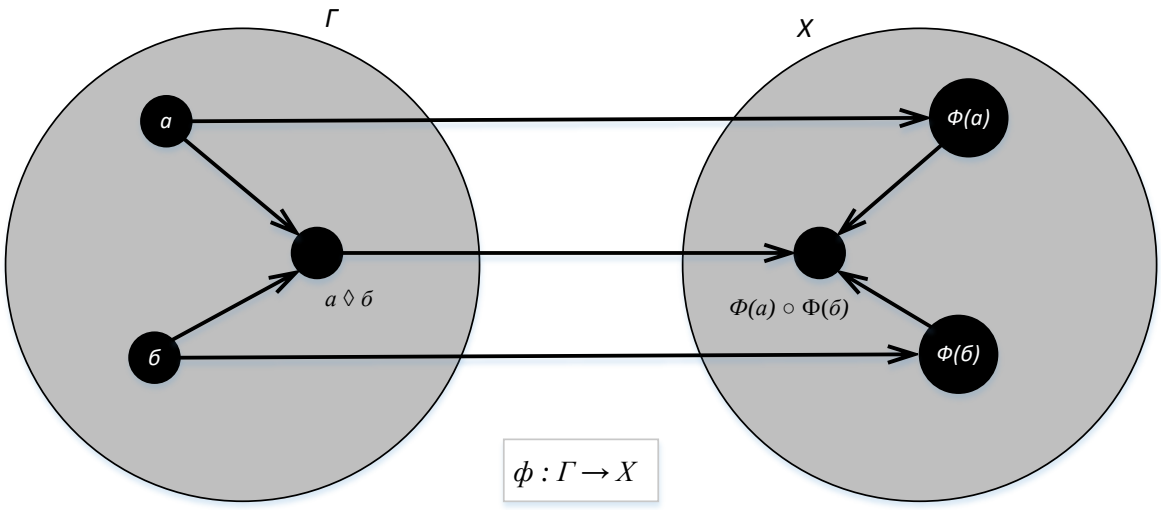
Хомоморфна криптографија је облик криптографије који омогућава да одређене инструкције извршене над шифратом дају шифровани резултат, који након дешифровања одговара инструкцијама извршеним над отвореним форматом. Ова функционалност је изузетно пожељна у модерним комуникационим системима. Први криптографски модел који је имао хомоморфна својства био је РСА. Међутим, због

сигурности шифрата РСА додаје насумичне бите на поруку пре шифровања. Додавањем вредности тј. бита у поруку губи се хомоморфност овог решења.

У апстрактној алгебри хомоморфизам је повезаност између две алгебарске структуре или групе при чему је сачувана структура [26]. Група $Г$ формира се заједно са операцијом ○ и комбиновањем бар два елемента $а$ и $б$ да би се формирао трећи елемент означавајући га са $а ○ б$. Да би се квалификовао као група, $Г$ скуп и операција морају испунити четири услова или аксиома групе [27]:

- Закључење: за све $а$ и $б$ у $Г$, резултат операције $а ○ б$ је такође у $Г$
- Асоцијативност: за све $а$, $б$ и $ц$ у $Г$, $(а ○ б) ○ ц = а ○ (б ○ ц)$
- Идентификујући елемент: У групи $Г$ постоји елемент $е$. За елемент $е$ у групи мора постојати једнакост у функцији $а ○ е = е ○ а = а$. Овакав елемент је јединствен за групу и због тога представља идентификујући елемент
- Повратни елемент: За свако $а$ у Групи $Г$, постоји елемент $б$ који у операцији са $а$ даје $а ○ б = б ○ а = е$ где је $е$ идентификујући елемент

Резултат операција може зависити и од редоследа елемената. Резултат комбиновања елемената $а$ и $б$ не мора дати исти резултат као комбиновање елемената $б$ и $а$. У случају група у којима постоји једнакост у функцијама $а ○ б$ и $б ○ а$ такве групе се називају Абелове или комутативне групе [28].



Слика 19. Хомоморфизам група

Узимајући две групе ($Г$, ◊) и (X, ○), хомоморфизам групе ($Г$, ◊) на групу (X, ○) је функција $ф : Г \rightarrow X$

Представљајући алгоритам за криптографију са чиниоцима *О, Ш, К, Е, Д*. Где карактери *О* и *Ш* чине отворени садржај и шифрат, *К* представља кључ, *Е* и *Д* су функције шифровања и дешифровања. Претпостављајући да отворени текст формира групу (*О,* ◊) и шифрат формира групу (*Ш,* ○) онда је функција шифровања *Е* функција пресликавања групе *О* на групу *Ш* на начин да је $E_к : О \rightarrow Ш$ где *к* чини тајни кључ у симетричном систему шифровања или јавни кључ у систему са јавним кључевима. Ако је за сваки *а* и *б* у *О* и кључ *к* тачно

$$E_к(а) \circ E_к(б) = E_к(а \lozenge б)$$

Онда је такав модел шифровања хомоморфан.

Шафи Голдвасер (енгл. *Shafi Goldwasser*) и Силвио Мицали (енгл. *Silvio Micali*) 1982. године развили су шифарски модел заснован на архитектури јавног кључа, ово решење је имало хомоморфна својства [29]. Голдвасер- Мицали (ГМ) су представили решење које је у потпуности задовољавало Киркохове принципе [30]. Међутим, овај криптолошки систем није ефикасан јер величина шифрата више стотина пута превазилази величину отвореног текста [27]. Голдвасер- Мицали су предложили широко прихваћену дефиницију семантичке сигурности.

ГМ се састоји од три алгоритма:

- Алгоритам вероватноће за генерисање кључева (генерише приватни и јавни кључ)
- Алгоритам за шифровање базиран на вероватноћи
- Детерминистички алгоритам дешифровања

Модел се ослања на то да је дата вредност *а* квадрат *мод Н*, узимајући у обзир факторизацију (*п,м*) од *Н*. Ово се може постићи употребом следеће процедуре:

Израчунати

$$a_п = a \ (мод \ м)$$

$$a_м = a \ (мод \ н)$$

Ако је

$$a_п^{(н-1)/2} = 1 \ (мод \ м)$$

$$a_м^{(м-1)/2} = 1 \text{ (мод н)}$$

Тада је *а* квадратни остатак *мод Н*.

Генерисање кључа у ГМ систему врши се на исти начин као и у РСА систему. Алиса генерише два различита велика проста броја *п* и *м*. Ови бројеви се генеришу на такав начин да су они насумични и нису у директној корелацији. Алиса затим рачуна $Н = нм$. Након тога се рачуна вредност *а* без остатка на следећи начин:

$$a_п^{(н-1)/2} = -1 \text{ (мод м)}, a_м^{(м-1)/2} = -1 \text{ (мод н)}$$

Јавни кључ се састоји од (*а, Н*).

Шифровање се врши према следећем принципу. Узимајући да Боб жели да пошаље поруку за Алису. Боб прво конвертује поруку у низ бита ($м_1, м_2 \ldots м_н$). За сваки од бита $м_и$, Боб генерише вредности $б_и$ из групе *модуло Н*.

$$ш_и = б_и^2 * м_и \text{ (мод Н)}$$

Резултат Боб шаље у виду низа бита ($ш_1, ш_2 \ldots ш_н$) Алиси.

По пријему поруке Алиса може дешифровати $ш_и$ у $м_и$ на следећи начин. За свако *и*, користећи факторизацију (*п, ј*). Алиса за свако $ш_и$ може добити остатак који у случају да постоји вредност поставља се вредност $м_и = 0$ у другом случају вредност $м_и = 1$.

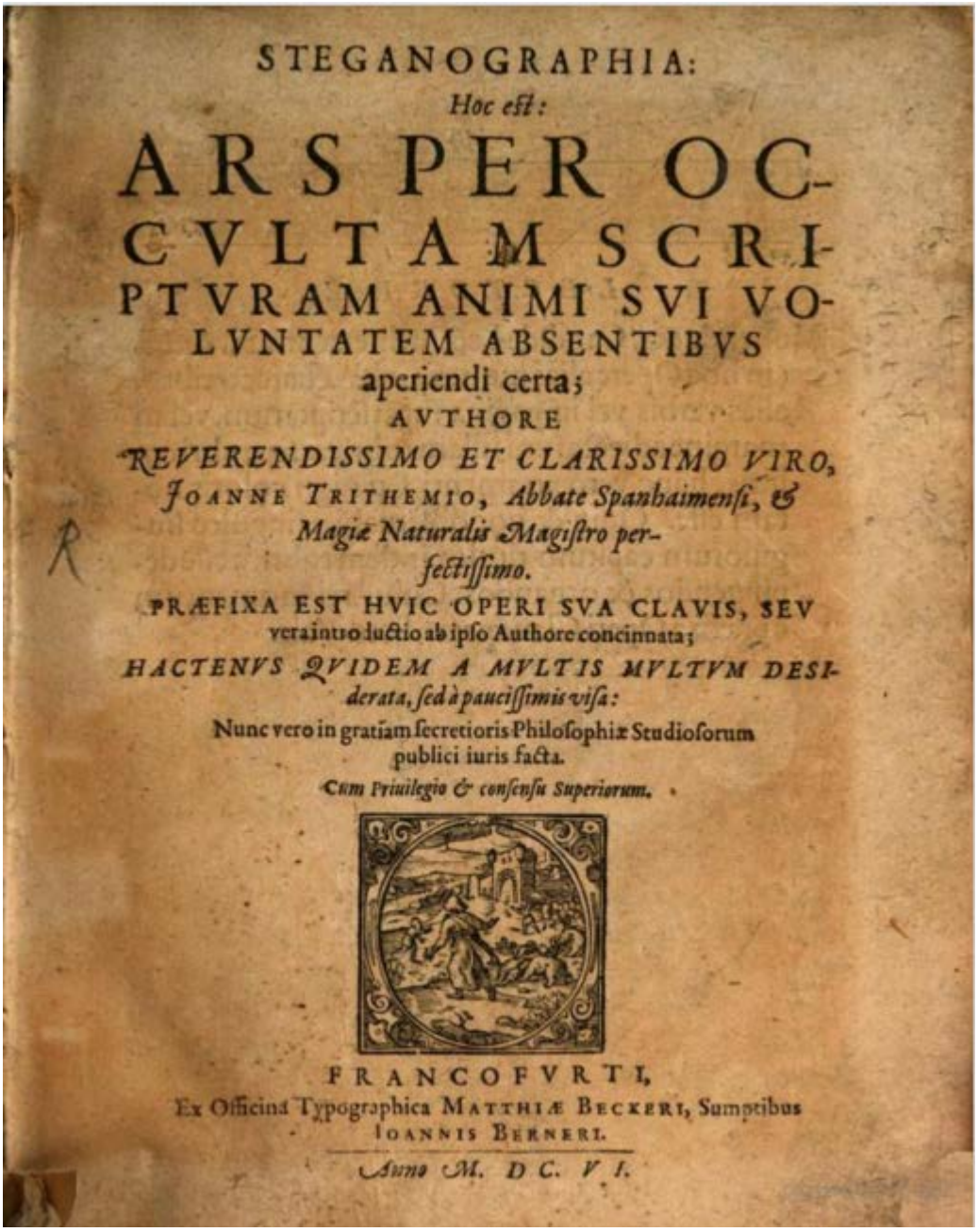
Сигурност ГМ модела заснована је на шифровању уз основе вероватноће. Основе вероватноће се односе на насумичности алгоритма за шифровање, која у основи доводи до тога да у случају шифровања исте поруке више пута резултат шифровања ће бити различит шифрат. Израз основе вероватноће се најчешће користи када се говори о системима шифровања са јавним кључем, мада одређени алгоритми тајног кључа који користе моделе увезивања (енгл. *Chaining mode*) поготово блоковски системи постижу слична својства [31].

2.3 Стеганографија

Стеганографија представља методу сакривања тајне поруке у јаван медиј. Јаван медиј може бити текст, слика, аудио фајл, видео материјал, и сл. У основи поента

стеганографије није да онемогући противника у дешифровању тајне поруке, него у прикривању постојања тајне поруке у комуникацији.

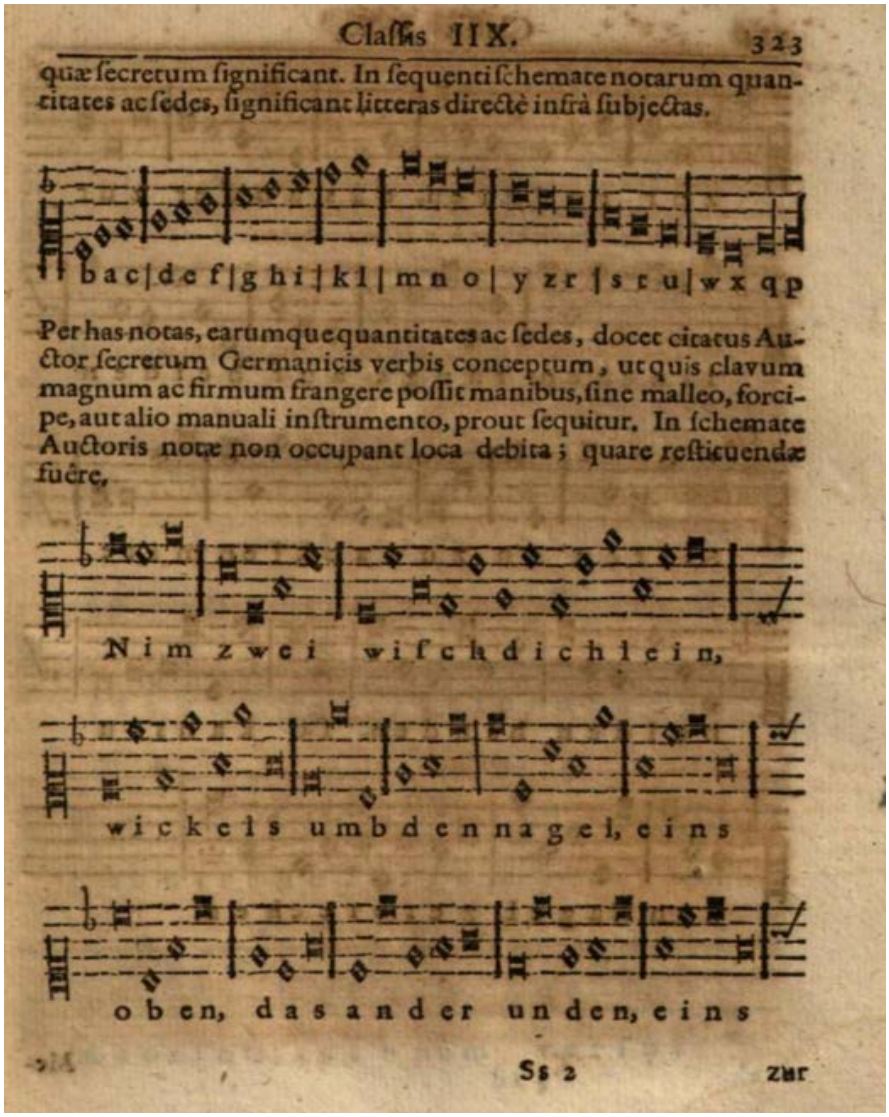
Употреба стеганографије се може пронаћи далеко у прошлости [32]. Писмо старих Египћана било је засновано на симболима/сликама које је могао тумачити само онај коме је познато значење симбола, за оне којима писмо није познато порука изгледа као графички приказ.



STEGANOGRAPHIA:
Hoc eſt:
ARS PER OC-
CVLTAM SCRI-
PTVRAM ANIMI SVI VO-
LVNTATEM ABSENTIBVS
aperiendi certa;
AVTHORE
REVERENDISSIMO ET CLARISSIMO VIRO,
IOANNE TRITHEMIO, Abbate Spanhaimenſi, &
Magiæ Naturalis Magiſtro per-
fectiſſimo.
PRÆFIXA EST HVIC OPERI SVA CLAVIS, SEV
vera intuo luctio ab ipſo Authore concinnata;
HACTENVS QVIDEM A MVLTIS MVLTVM DESI-
derata, ſed à pauciſſimis viſa:
Nunc vero in gratiam ſecretioris Philoſophiæ Studioſorum
publici iuris facta.
Cum Priuilegio & conſenſu Superiorum.
FRANCOFVRTI,
Ex Officina Typographica MATTHIÆ BECKERI, Sumptibus
IOANNIS BERNERI.
Anno M. D C. V I.

Слика 20. Прва страница књиге Steganographia штампана 1606 године у Њемачкој

Грци су користили методу сакривања поруке који су исписивали на дрвеним плочама које би прекрили воском, затим би на том воску исписали неку другу поруку [33]. Тајна порука је остајала сакривена испод воска и јавне поруке. Једна од метода употребе стеганографије током другог светског рата је употреба „невидљивог мастила“ [2]. Порука исписана оваквим мастилом није била видљива док на папир није нанесено посебно средство које правећи реакцију са мастилом приказује скривени текст. Стеганографија има одређену предност над криптографијом. Сакривање поруке у музичким нотама је такође представљено као стеганографска метода сакривања поруке још 1680. године . Свака нота је одређивала једно слово алфабета и на тај начин је сакривана порука [34].



Слика 21. Стеганографија нотама, *Schola steganographica* 1680. [35]

У стеганографији порука је скривена и самим тиме неће изазвати пажњу, док у криптографији сама ситуација увиђања садржаја који није читљив може изазвати откривање скривене комуникације и идентификовати стране у комуникацији.

Развојем технологије почео је и развој примене стеганографије, самим тиме проширена је могућност употребе дигиталних медија за сакривање порука. Дигитализација је отворила низ нових подручја у којима се могу сакрити поруке, већина чак неприметна за људско око. Оваква прикривеност прави окружење у ком је изузетно тешко открити присуство скривеног податка или поруке у дигиталном медију. За учеснике који желе да тајно комуницирају, нарочито на интернету овај метод омогућава једно од најбољих решења. Порука се може шифровати најбоље доступним алгоритмом, а истовремено се јако ефикасно може прикрити постојање поруке. Стеганографске методе се могу груписати у три основне групе:

- Чиста стеганографија
- Стеганографија са тајним кључем
- Стеганографија са јавним кључем

Чиста стеганографија је метода у којој није потребна претходна размена неке тајне информације пре размене садржаја. Код оваквог вида скривене комуникације никаква информација осим медија се не размењује између учесника у комуникацији. Сигурност оваквог система ослања се искључиво на тајност присуства поруке. Оваква метода нуди слаб види сигурности. Узимајући у обзир Керкохове принципе, претпоставља се да је нападачу познат алгоритам који се користи тиме је овакав вид размене несигуран. Увођењем тајне информације која се може разменити раније (стегографски кључ) уводи се сигурни параметар без ког није могуће извлачење тајне информације. Стеганографија са тајним кључем захтева размену тајног кључа, ова размена ремети првобитну предност прикривене комуникације. Слабост решења са тајним кључем решава стеганографија са јавним кључем. Као и у криптографији јавног кључа стеганографија са јавним кључем се не ослања на размену кључа. Овај метод користи пар кључева, тајни и јавни. Јавни кључ се чува у јавној бази. Јавни кључ се користи у процесу уметања скривене поруке, док се тајни кључ користи за извлачење тајне поруке из медија. Имплементација система стеганографије са јавним

кључем подржана је апстрактношћу шифрата до те мере да га је могуће сакрити на такав начин да и по откривању не бива препознато да је реч о шифрату [35] [36].

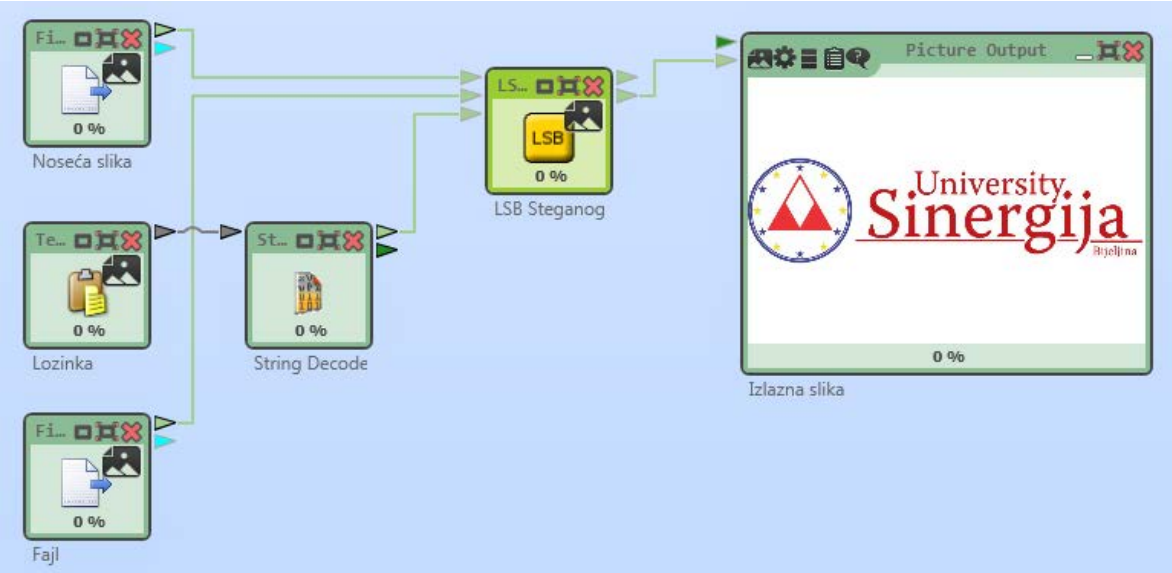
У модерној стеганографији најчешће се корист бит најмање важности као складиште сегмената скривене поруке. Изменом бита најмање важности не утиче се много на садржај фајла али ствара се простор за уметање бита поруке које се касније на страни пријема могу извући.

Текст представља поруку коју је најлакше сакрити у сликовне датотеке. Слике представљају најбољи медиј управо због битмапског формата. Мапа бита садржи запис вредности сваког пиксела у форми низа бита. Измена бита најмање важности представља једну од најједноставнијих техника које се користе у стеганографији приликом рада са сликама. Применом ове технике замењује се део оригиналне информације вредности пиксела са вредношћу поруке која се сакрива. Заменом бита најмање важности постиже се минимално одступање од оригиналне боје пиксела. Оваквом заменом боја пиксела постиже се да комплетна слика која се користи за сакривање информације визуелно људском оку не изгледа другачије од оригиналне слике.

За пример приказа сакривања садржаја кориштен је алат *Cryptool* који се користи за симулирање криптолошких решења. Кориштена је слика у *JPG* формату величине 41 килобајт, слика која је сакривена је у PNG формату и величине 20,1 килобајт.

Слика 22. Слика у којој је скривена слика

Слика 23. Скривена слика



Слика 24. Приказ симулације у *Cryptool* алату

Резултат је нова слика у jpg формату. У овом примеру је кориштен модел стеганографије са тајним кључем.

Слика 25. Резултат стеганографшког скривања

2.4 Криптоанализа

Основна улога криптографије је да обезбеди тајност поруке од учесника у комуникацији за које та порука није намењена. Како је размотрено раније нападач такође може изменити садржај поруке и на тај начин угрозити интегритет поруке. Полази се од претпоставке да нападач има потпун приступ комуникационом каналу.

Криптоанализа се бави анализом, проналажењем слабости и нападима на криптолошке механизама. Успешни напад може доћи до дешифроване поруке, може успешно изменити део поруке или фалсификовати дигитални потпис. Криптоанализу заједно са криптографијом дефинишемо као криптологију, науку која се бави дефинисањем сигурних шифарских алгоритама као и анализом и проналажењем слабости постојећих алгоритама.

Једна од полазних тачака у криптоанализи се ослања на закључке Керкохових принципа [37] [38]. Према овим принципима полази се од претпоставке да нападач познаје све делове система за шифровање, како алгоритам тако и начин имплементације алгоритма. Према овом принципу сигурност система шифровања се заснива на тајном кључу.

Узимајући у обзир који ресурс нападач има на располагању напади могу имати неки од следећих облика:

- Напад на шифрат
- Напад са познатом дешифрованом поруком
- Напад са познатим делом шифроване поруке
- Напад са одабраним текстом
- Напад са одабраним шифратом

Напад на шифрат или шифровану поруку је најчешћи сценарио који се разматра у анализи и дизајнирању модерних шифарских алгоритама. Полази се од претпоставке да је нападач пресрео комуникацију и дошао до шифрата.

Напад са познатом дешифрованом поруком је метода у којој је нападачу позната дешифрована и шифрована порука на основу који покушава да дође до тајног кључа који је кориштен у процесу шифровања.

Напад са познатим делом шифроване поруке много мање је ефикасан од напада у случају када је цела дешифрована порука позната. У овом случају полази се од претпоставке да је познат део поруке нпр. поздрав на крају поруке или позната шаблонска поља мејл поруке.

Напад са одабраним текстом је мање реално изводљив, мада могућ. Овим нападом се покушава утицати на улаз тј. дешифровану поруку на такав начин да она изазива промене у шифрату на такав начин да се добија очекивани излаз (грешка) којим се лакше долази до кључа.

Напад на шифрат или шифровану поруку најчешће се врши нападом сировом снагом (енгл. *Brute force*). Овај напад једноставно испробава све могуће варијације кључа на шифрованом поруком и покушава да дође до смисленог дешифроване поруке. Једино могуће олакшавање комплетне претраге кључа је дељење кључа на делове који се деле између више процесора и на тај начин се убрзава претрага. Напад сировом снагом има једну кључну предност а то је да ће гарантовано доћи до исправног кључа, једини проблем је време потребно за проналажење кључа. У случају кључа дужине 40 бита постоји преко милијарде могућих кључева. Оваква дужина кључа данас није проблем са обзиром на брзину процесора и могућност испробавања чак 4 милиона кључева по секунди, могуће је пронаћи за мало више од једног дана [39].

2.5 Архитектура, програмски језици, реверзни инжењеринг

Основну компоненту рачунара чини централна процесорска јединица (енгл. *Central Processing Unit*). Процесор чини „мозак“ рачунара, контролише остале компоненте, врши рачунање и остале операције са подацима.

Процесор представља струјно коло предвиђено за извршавање инструкција. Инструкције које је потребно извршити се чувају у меморији. Чување у меморији могуће је за све врсте података, подаци добијају нумеричке вредности и чувају се на одређеној локацији у меморији. Локација у меморије назива се адреса. Процесор тражи од меморије одређену адресу а она му враћа податак који се налази на тој адреси.

Инструкције су такође представљене нумерички. Свакој инструкцији додељује се јединствена нумеричка вредност. Када процесор добије одређен број инструкције он

је извршава. На пример број 35 ће дати инструкцију да копира податке са једне меморијске адресе у другу, број 48 даје инструкцију сабирања два броја и сл. Додељивање броја одређеној инструкцији дефинише се приликом дизајнирања процесора, у том тренутку се дефинишу инструкције и нумеричке вредности инструкција. Овим дефинисањем инструкција и додељивање нумеричких ознака инструкција праве се правила која се називају архитектура. На овај начин произвођачи могу производити различите процесоре базиране на истој архитектури, процесори се могу разликовати по брзини, потрошњи електричне енергије или цени али сви разумеју исте нумеричке вредности као исте инструкције.

По извршавању одређене инструкције дефинисане одређеним кодом, процесор наставља вршећи следећу инструкцију. Понекад инструкцијом се може прелазити на различите локације у меморији, или се може вратити уназад да би се одређена инструкција поновила фактички правећи петљу. Низ оваквих инструкција формирају програм и називају се опкод.

Операције које инструкције извршавају су веома једноставне. Само писањем низа оваквих инструкција може се постићи да процесор обави неки задатак. Међутим писање низа или више низова оваквог нумеричког кода је јако захтевно. Због оваквог отежаног писања инструкција и прављења програма направљен је асемблерски језик.

```
00000000:08000060 bb 00 00 00 08    mov ebx, 0x08000000
00000000:08000065 66 b8 2f 62       mov ax, 0x622f
00000000:08000069 66 89 03          mov word ptr [rbx], ax
00000000:0800006c 66 b8 69 6e       mov ax, 0x6e69
00000000:08000070 66 89 43 02       mov word ptr [rbx+2], ax
00000000:08000074 66 b8 2f 73       mov ax, 0x732f
00000000:08000078 66 89 43 04       mov word ptr [rbx+4], ax
00000000:0800007c 66 b8 68 00       mov ax, 104
00000000:08000080 66 89 43 06       mov word ptr [rbx+6], ax
00000000:08000084 66 b8 00 00       mov ax, 0
00000000:08000088 66 89 43 08       mov word ptr [rbx+8], ax
00000000:0800008c 66 b8 00 08       mov ax, 0x0800
00000000:08000090 66 89 43 0a       mov word ptr [rbx+10], ax
00000000:08000094 66 b8 00 00       mov ax, 0
00000000:08000098 66 89 43 0c       mov word ptr [rbx+12], ax
00000000:0800009c 66 b8 00 00       mov ax, 0
00000000:080000a0 66 89 43 0e       mov word ptr [rbx+14], ax
00000000:080000a4 b8 0b 00 00 00    mov eax, 11
00000000:080000a9 bb 00 00 00 08    mov ebx, 0x08000000
00000000:080000ae b9 08 00 00 08    mov ecx, 0x08000008
00000000:080000b3 89 ca             mov edx, ecx
00000000:080000b5 83 c2 04          add edx, 4
00000000:080000b8 cd 80             int 0x80
```

Слика 26. Инструкције и асемблер

Одређене нумеричке инструкције које обављају функције замењују се називима инструкција. Инструкција за померање података из једне ћелије у другу носи нумеричку вредност 34, овој функцији додељује се скраћеница *MOV* скраћеница од енглеске речи помери (енгл. *Move*). Број 48 представља инструкцију сабирања, ова вредност се мења са командом *ADD* што представља енглеску скраћеницу речи додај.

Програмер пише низ инструкција, једноставних операција које процес треба да изврши користећи имена што је много лакше и брже од нумеричких ознака. Након писања инструкција покреће алат под називом асемблер. Назив асемблер се често користи као назив програмског језика али он у суштини чини алат који врши конверзију симбола у одговарајуће нумеричке ознаке које процесор извршава [40].

Међутим у већини случајева сама инструкција није довољна. У случају да је потребно сабрати два броја, потребно је прво дефинисати те бројеве, затим дефинисати операцију. Исто важи у случају померања података, потребно је дефинисати изворну меморијску адресу и одредишну адресу. Ово се обавља уз помоћ додавања оператора инструкцији. Ово оператори дају инструкцији додатне параметре који су потребни за извршавање те инструкције. Ови оператори су такође смештени у меморију уз опкод инструкција.

Ако је потрено да извршимо премештање података са једне на другу локацију у адреси, нпр. да са адресе 1010 преместимо податке на нову адресу 1432 потребно је написати команду:

```
MOV 1432 1010
```

На почетку наводимо инструкцију затим први оператор представља нова локација података, други оператор представља изворну меморијску локацију. Асемблер чува у меморији ове операторе да би при учитавању опкода инструкције знао са које локације на коју врши померање. Након прикупљања потребних параметара у виду кода инструкције и локација процесор ће извршити операцију.

Сама архитектура процесора дефинише операције које процесор може извршити и који опкод дефинише коју инструкцију. Архитектура такође може да дефинише и колико регистра (привремена меморијска складишта) има процесор, како процесор комуницира са осталим уређајима као и остале функционалности. Управо због овог

сваки процесор има свој асемблер. Самим тиме не постоји један асемблер за све процесоре већ у зависности од архитектуре користи се одговарајући асемблер.

Већина процесора личних рачунара заснована је на архитектури *x86* или ако су у питању 64 битни системи онда је ознака *x64* што значи да се за ову платформу користи одговарајући асемблер. Већина мобилних платформи има ARM архитектуру, тако да програмери који програмирају за овај процесор морају користити асемблер искључиво намењен овој платформи. Процесори се могу разликовати по брзини али управо због исте архитектуре програми писани у одговарајућем асемблеру ће радити само се може разликовати брзина извршавања.

Две основне препреке у асемблерском начину програмирања довеле су до развоја програмских језика вишег нивоа. Прва препрека је у величини и броју инструкција које су морале бити написане да би се постигла нека комплекснија операција. Овакво писање је одузимало много времена и било изузетно напорно. Други проблем је архитектура, где програм писан за једну архитектуру бива потпуно неупотребљив на другој платформи. Оба ова проблема су решена увођењем програмских језика вишег нивоа.

Узимајући пример да је потребно извршити комплекснију математичку операцију. Потребно је израчунати вредност *A = 3 + (7 - 5) * 3.* Међутим процесор не подржава овакве операције и може извршити само једноставне операције. Значи потребно је поделити ово рачунање у мање операције. Ово се може поистоветити са ручним рачунањем засебних вредности операција нпр. прво је потребно израчунати разлику између вредности 7 и 5, након тога резултат те операције је потребно помножити са бројем 3 и на крају сабрати резултат са бројем 3. Извођење операција се може извршити на следећи начин:

```
SUB #5 #7 // Одузима се број 5 од броја 7 и смешта у регистар
MUL A, #3 // Множи се вредност из регистра A са бројем 3
ADD #3, A // Сабира се вредност из регистра A са бројем 3 и смешта у регистар A
```

Јако ретко у ће се у реалности користити фиксни бројеви, углавном рачунају се вредности које се налазе на одређеним меморијским локацијама. На тај начин се додатно компликује рачунање израза који т том случају има следећи облик

*@150 = @100 + (@105 - @102) * @107*

У овом изразу симбол „@“ значи на адреси тј. број се налази на задатој адреси. Да би операција била могућа потребно је прву учитати вредности у меморију и онда извођење операције се врши на следећи начин:

```
MOV B, 102  // Смештање вредности са меморијске адресе 102 у регистар B
MOV A, 105  // Смештање вредности са меморијске адресе 105 у регистар A
SUB A, B    // Одузимање вредности (@105 - @102) смештање резултата у регистар A
MOV B, 107  // Смештање вредности са меморијске адресе 107 у регистар B
MUL A, B  // Множење вредности претходне операције и броја са адресе 107 смештеног у регистар B
MOV B, 100  // Смештање вредности са меморијске адресе 100 у регистар B
ADD A, B   // Сабирање вредности регистра A (( @105 - @102 ) * @107) и вредности у регистру B (@100) и смештање резултата у регистар A
MOV 150, A  // Смештање вредности регистра A на меморијску адресу 250
```

Као што се може видети на примеру чак и једноставни изрази и операције могу бити компликоване за извођење и могу захтевати много линија кода. Додатну отежавајућу околност представља разумевање кода и који део кода врши коју операцију. Још један проблем је рад са промењивим. У основи се рад са промењивим своди на адресе промењивих што прави проблем у раду са више промењивих и значајно отежава рад.

Због комплексности и проблематике рада са великим бројем израза, локацијама у меморији и инструкција уводи се додатни слој који чини компајлер. Одређен програмски језик дефинише изразе који се могу користити и на који начин се могу користити, наравно компајлер мора подржавати те изразе. Овим увођењем се израз из претходног примера у *C* програмском језику може приказати на следећи начин:

```
int v1, v2, v3, v4, r;
v1 = 3;
v2 = 7;
v3 = 5;
v4 = 3;
r = v1 + (v2 – v3) * v4;
```

Када се изврши компајлирање овог кода, компајлер прво анализира код и идентификује да је потребно пет промењивих. У писању кода није било потребно дефинисати меморијске адресе јер ово одрађује аутоматски компајлер. У случају из примера компајлер може дефинисати за промењиву *v1* меморијску адресу 100, за

промењиву *v2* меморијску адресу 101 и сл. Компајлер води рачуна и евиденцију о додељеним меморијским адресама.

У датом примеру постоји неколико додељивања вредности промењивим, почевши од додељивања вредности промењивој *v1*.

```
v1 = 3;
```

Компајлер чита израз и према правилима датог програмског језика тумачи да је потребно да се промењивој *v1* додели вредност 3. Компајлер зна којој локацији у меморији је доделио простор за ову промењиву и на основу тога уписује вредност на одређену локацију у меморији. На основу познатих параметара компајлер генерише потребне инструкције које ће процесор разумети. Овим ће излаз из компајлера ка процесору бити инструкције:

```
MOV 100, #3  // Број 3 се смешта на меморијску локацију 100 што одговара v1 = 3;
MOV 101, #7  // Број 7 се смешта на меморијску локацију 101 што одговара v2 = 7;
MOV 102, #5  // Број 3 се смешта на меморијску локацију 100 што одговара v3 = 5;
MOV 103, #3  // Број 3 се смешта на меморијску локацију 100 што одговара v4 = 3;
```

Инструкције се конвертују у опкод који се прослеђује на извршавање процесору. Такође наставак кода који тражи резултат функције *r = v1 + (v2 – v3) * v4;* дели се у инструкције:

```
MOV B, 102 // Смештање вредности са меморијске адресе 102 (промењива v3) у регистар B
MOV A, 101 // Смештање вредности са меморијске адресе 101 (промењива v2) у регистар A
SUB A, B // Одузимање (v2 - v3) и чување резултата у регистру A
MOV B, 103 // Смештање вредности са меморијске адресе 103 (промењива v4) ) у регистар B
MUL A, B // Множење вредности претходне операције и броја са адресе 103 смештеног у регистар B
MOV B, 100 // Смештање вредности са меморијске адресе 100 промењива v1) у регистар B
ADD A, B // Сабирање вредности из регистра A ((v2 - v3) * v4) са вредношћу из регистра B (v1) и чување резултата у регистру A
MOV 150, A // Смештање вредности резултата на меморијску адресу 150 (промењива r)
```

Као што се види из примера програмски језици вишег нивоа дозвољавају лакши рад, већу флексибилност и читљивост у писању инструкција. Увођење компајлера доводи

у питање потребу за познавањем асемблера данас. Постоји неколико разлога за познавање асемблера. Компајлер неће увек генерисати оптималне инструкције. Одређене радње се могу извршити знатно ефикасније употребом нестандардних метода које нису познате компајлеру. Ово је јако важно када је потребно извући максималне перформансе, и у том случају се одређен сегмент може урадити посебно у асемблеру.

Проблем са перформансама је најизраженији код мањих уређаја са ограниченим ресурсима. Код оваквих уређаја се мора постићи максимална искоришћеност већ малих ресурса који су уграђени и не могу се надоградити. Додатно ограничење компајлера је његова ограниченост на функционалности које су му познате. У случају да се користи нека нова функционалност процесора компајлер не може генерисати инструкцију за ту функционалност.

Познавање асемблера у данашње време је потребно за случајеве анализирања софтвера, за хаковање апликација или измену функционалности апликација. Много је једноставније анализирати постојећи програм ако не постоји доступан изворни код, једноставном заменом нумеричких вредности са одговарајућим називима инструкција чиме се добија асемблерски „језик“.

Велики проблем асемблера такође чини преносивост. Ако се креира одређени опкод са инструкцијама за одређен процесор он ће радити само на одређеној платформи. Ако би исте инструкције користили на другој платформи оне би биле неупотребљиве и морало би се извршити превођење на одговарајући опкод.

Добро решење за проблем преносивости нуде интерпретерски језици. Употребом интерпретерских програмских језика изворни код остаје непромењен а компајлира се у општи асемблерски код. Оваква је ситуација код компајлирања Јава програмског језика чији резултат компајлирања чини бајт код. Да би се извршио овакав програм потребан је тумач или интерпретер.

Предност употребе интерпретерских програмских језика је преносивост, флексибилност и сигурност. Довољно је програм написати једном и он ће се извршавати на свакој архитектури на којој је интерпретер доступан, без потребе да се измени нешто у самом програму или да се он прилагоди. Због контроле коју

интерпретер има над тим шта програм може радити, сигурност је такође већа. Интерпретер може блокирати одређене радње и тиме спречити несигурне функције или радње програма. Једна од мана интерпретерских језика је брзина извршавања.

Сама заштита софтвера се донекле ослања на процес компајлирања да би се прикрио начин функционисања као и да се заштити интелектуално власништво. Реверзни инжењеринг чини процес анализирања система или неке системске компоненте, веза које постоје између компоненти и прикаже њихове односе и функционалности [41]. Ово омогућава да се открије структура софтвера, начин функционисања, као и функционалности које поседује. Одређен број метода за анализу и доступних алата за анализу софтвера дају могућност увида у суштину одређеног софтвера.

Реверзни инжењеринг је јако користан у анализи софтвера и има широку намену, нек од намена реверзног инжењеринга су:

- Проналажење малициозног кода. Многи аутори малициозног софтвера свој софтвер сакривају у оквиру легитимног софтвера. Анализом малициозног кода може се разумети начин функционисања злонамерног софтвера и уочити одређени шаблони који се могу искористити за касније детектовање [42].
- Проналажење грешки и рањивости. Чак и најбоље дизајнирана софтверска решења могу имати пропусте и грешке који се могу лако идентификовати и исправити.
- Проналажење и употреба функционалности које изворни аутори софтвера нису предвидели.

Реверзни инжењеринг захтева комбинацију вештине и добро разумевање развоја софтвера. За успешно анализирање неопходно је савладати препреке у виду „отварања“ кода, анализе наредби и логичке анализе. Овакав метод има своје намене као у позитивном решавању проблема тако и у негативном сегменту у виду злоупотребе интелектуалног власништва и неовлаштене употребе софтвера.

2.6 Веб окружење

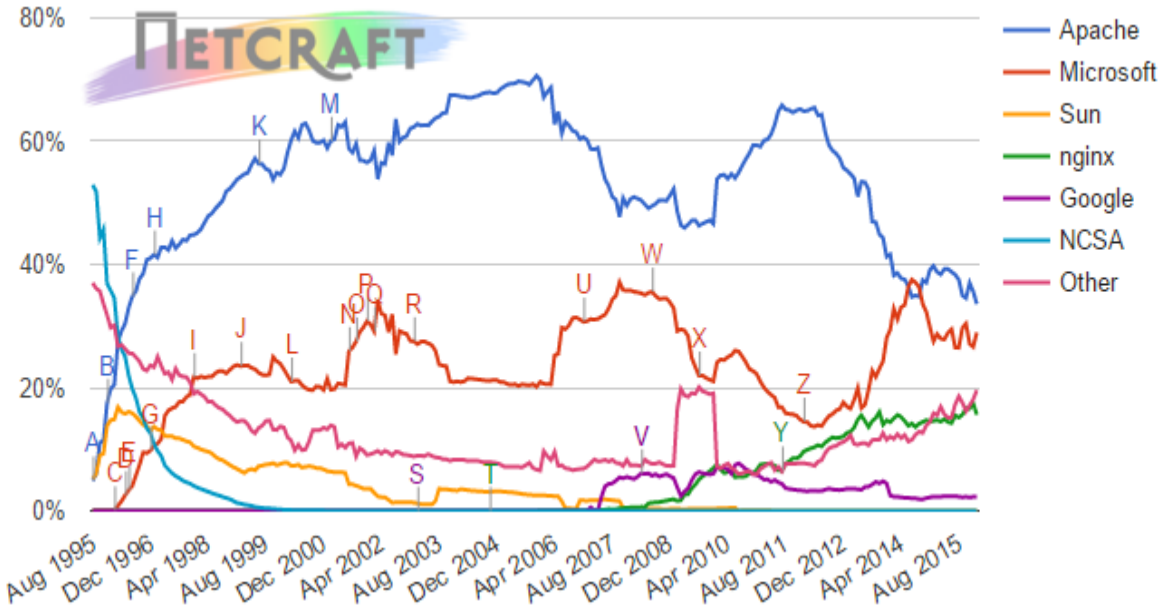
Тренутно најпопуларнији сервис Интернет мреже представља сервис Веб (енгл. *World Wide Web*). Назив Веб се често погрешно користи као синоним за Интернет мрежу. Интернет представља глобално повезане рачунарске мреже, док Веб представља један

од сервиса који за свој рад користи ту глобалну мрежу. Крајем осамдесетих година Тим Бернерс-Ли (енгл. *Tim Berners-Lee*) почео се бавити развојем сервиса који би уз помоћ универзалних идентификатора докумената у мрежном окружењу повезивао документе и информације [43]. Радећи у Европској организацији за нуклеарно истраживање 1990 године Тим Бернерс-Ли (енгл. *Tim Berners-Lee*) завршава развој Веба (енгл. *World Wide Web*) и на пролеће 1991 заједно са првим Веб сервером пушта у рад Веб [44].

Првобитна намена Веба је била да повеже и омогући приступ документима који су се налазили на различитим локацијама повезаним мрежом. Развојем долази се до нових програмских језика који омогућавају извршавање на страни сервера а тиме и креирање докумената са динамичким садржајем. За функционисање Веба Тим Бернерс-Ли развија три технологије:

- Универзални идентификатор докумената (енгл. *Universal document identifier*) који се касније замењен са шаблонском локацијом ресурса (енгл. *uniform resource locator URL*)
- Језик за објављивање *HyperText Markup Language*
- Протокол за пренос *Hypertext Transfer Protocol*

Преглед странице на Вебу почиње уносом локације ресурса у Веб браузер или праћењем линка на страници која води ка ресурсу или другом документу. Веб браузер затим започиње размену порука у позадину да би добио и приказао тражени ресурс или страницу.



Слика 27. Учешће различитих Веб сервера

Апач Веб сервер је од марта 1995 па до данашњег дана најпопуларнији Веб сервер. Прва верзија Апач веб сервера креирана је од стране Роберт Меккола који је био кључни у развоју веб сервера националног центра за примену супер рачунарства (енгл. *National Center for Supercomputing Applications*), познатог као *NCSA HTTPd*. Када је Меккол напустио NCSA средином 1994, развој *HTTPd* је одгођен, одређен број закрпа (енгл. *Patches*) за побољшања кружила је преко електронске поште великог броја других програмера који су сачинили првобитну Апач групу.

Објашњење како је Апач сервер добио име дао је 2007 године Брајан Белендорф говорећи да је због неколико ревизија сервера и примењених закрпа (енгл. *patches*), сервер постаје познат у групи као закрпљени веб сервер (енгл. a patche Web Server).

Верзија 2 Апач сервера била је битна преправка већине прве верзије Апач кода са јаким фокусом на даљу модуларност и развој транспортног слоја. Језгро Апач 2.x сервера има више главних унапређења за разлику од Апач 1.x. Они укључују *UNIX* нити, бољу подршку на не-УНИХ платформа нпр. Мајкрософт платформама, нови Апач *API*, и подршку за шесту верзију интернет протокола (IP V6). Прво алфа издање Апач Веб сервера 2 је било у марту 2000, а прво јавно доступно издање 6-тог априла 2002.

Верзија 2.2 је увела више флексибилних овлашћења. Такође је карактерише побољшање модула за кеширање садржаја и прокси модула. Апач подржава велики број функционалности, многе направљене као саставни модули који надограђују језгро функционалности. Многи опште прихваћени програмски језици за креирање интерфејса су подржани *mod_perl*, *mod phyton*, *Tcl*, і *PHP*. Популарни модули за аутентичност укључују *mod_access*, *mod_auth*, *mod_digest*, и *mod_auth_digest*, наследник *mod_digest*. Пример значајних функционалности је подршка за *SSL* и *TLS* (mod_ssl), *proxy* модул, *URL rewriter* (такође познат као *rewrite engine* унутар модула *mod_rewrite*), прилагођене лог датотеке (*mod_log_config*) и подршка за филтрирање (*mod_include* и *mod_ext_filter*).

Популарна метода компресовања на Апач Веб серверу укључује спољни модул, *mod_zip*, направљен за помоћ код смањења величине (тежине) веб страница.

Виртуелни хостинг дозвољава једној Апач инсталацији да користи више различитих стварних веб сајтова. На пример, једна машина, са једном Апач инсталацијом може истовремено опслуживати бифе различитих Веб домена.

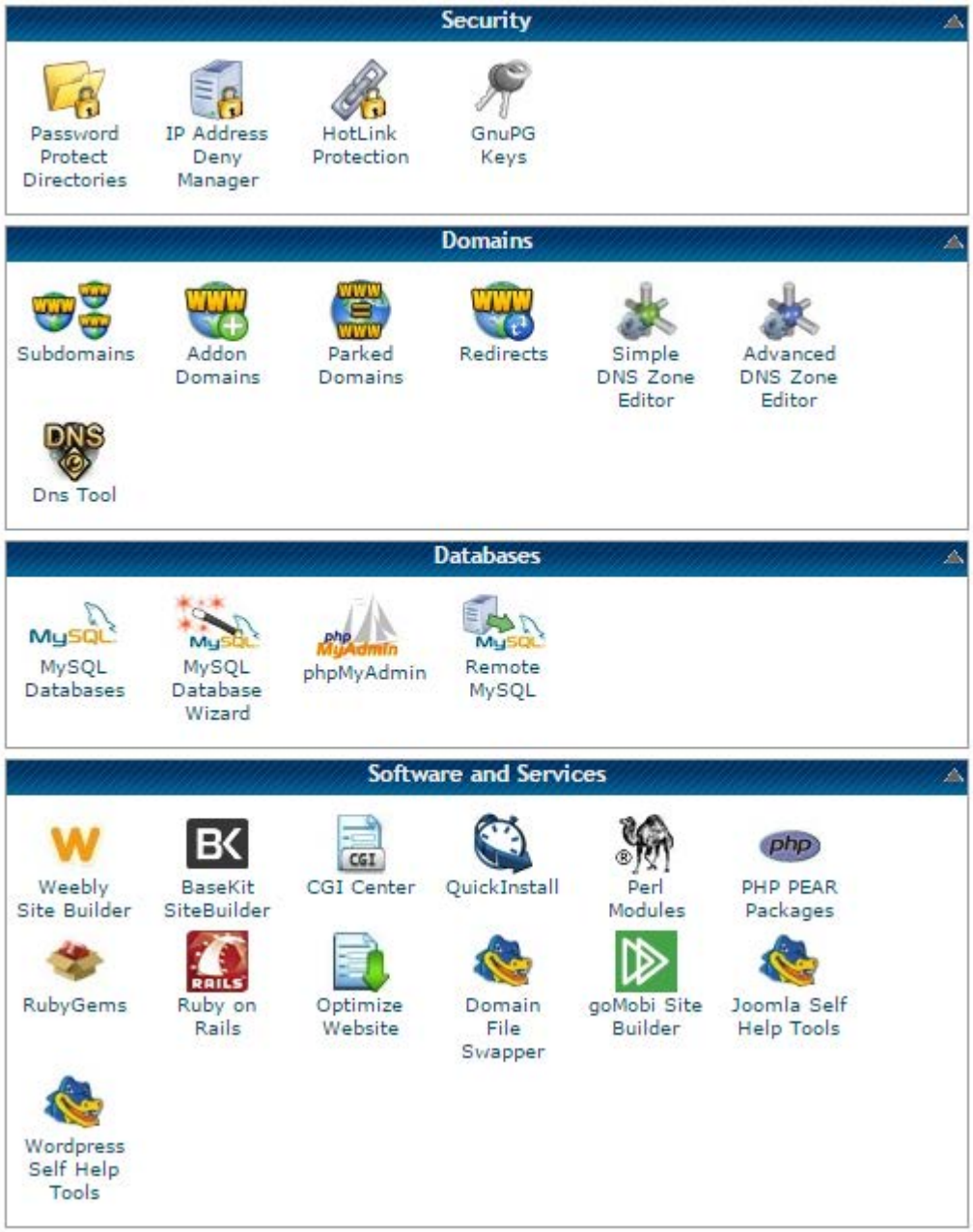
Апач подржава подешавање порука о грешкама, аутентичне базе података базиране на системима за управљање базама података (енгл. *Database management system - DBMS*), и преговарање садржаја. Такође је подржан од стране неколико графички интерфејса. Апач се првенствено користи да опслужује како статички садржај, тако и динамичке интернет странице. Многе Веб апликације су дизајниране на такав начин да очекују режим рада и опције које Апач пружа.

Апач Веб сервер, саставни је део популарног *LAMP* (*Linux, Apache, MySQL, and PHP*) веб сервер апликативног пакета. Апач се доставља као део многобројно заштићених софтверских пакета укључујући и *Oracle Database* или *IBMWebSphere* апликациони сервер. *Mac OS X* интегрише Апач као уграђени Веб сервер и као подршку за свој *WebObjects* апликациони сервер. Апач је укључен са *NovellNetWare 6.5*, где је активни Веб сервер. Апач је такође укључен у многе Линукс дистрибуције. Апач се користи за многе задатке где садржај мора бити на располагању на сигуран и поуздан начин.

Локација где се налази садржај тј. хостинг је важан фактор. Корисници желе да њихов садржај сачува интегритет извршавања, док аутори софтвера желе да заштите софтвер од нелегалног копирања и модификовања. Најчешће се за хостинг користе :

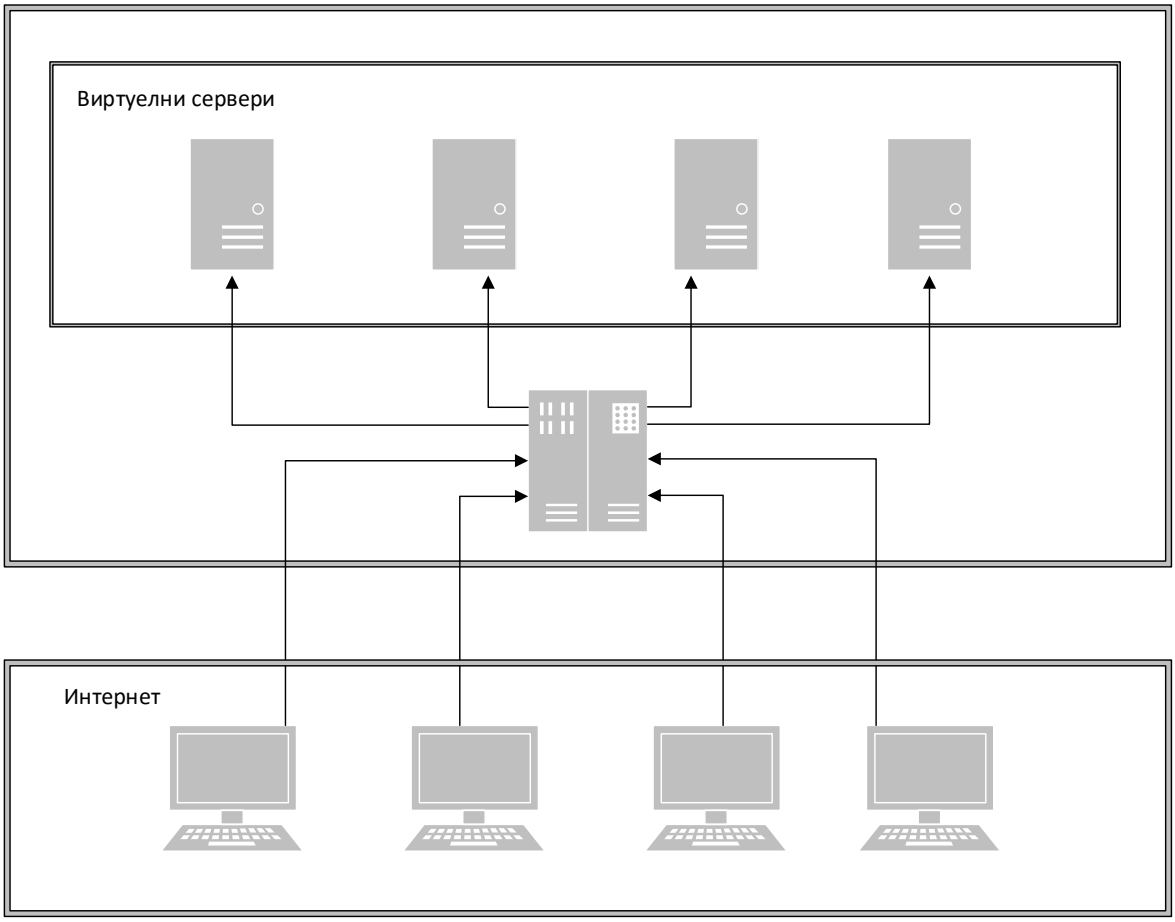
- Дељени хостинг
- Виртуелни приватни сервери

Дељени хостинг је јавни сервис који пружа услугу Веб сервера. Велики број хостинг сервиса је доступан како комерцијалних тако и бесплатних. Како се хостинг сервис дели на више корисника потребна је администрација од стране оног ко пружа услугу. Ово значи много мање контроле за корисника као и могућности за развој. Скоро сав софтвер који је намењен стандардним Веб серверима радиће и на дељеном хостингу, али развој сопствених решења као и контрола су ограничени. Мане оваквог начина држања и дељења садржаја се тичу ограничености самог сервиса. У случају неправилног подешавања привилегија над фајловима, могуће је да неовлаштени корисници добију приступ дељеним ресурсима и да на тај начин остваре приступ свим фајловима. Перформансе оваквих система нису на високом ниво и могу бити проблем код преоптерећења система. Због недостатка приступа кључним сегментима система корисник не може инсталирати своја решења или имати већу контролу над сервером у виду избора портова, слушања саобраћаја на одређеним повлашћеним портовима које може користити за апликацију коју хостује на сервису. Администрација од стране корисника се најчешће обавља употребом веб базиране контролне табле.



Слика 28. Контролна табла Веб хостинг

Виртуелни приватни сервери су виртуални рачунари који се изнајмљују за потребе хостинг сервиса. ВПС свој оперативни систем и корисник има над сервером пуну контролу тако да може извршити инсталацију додатних сервиса и било каквог софтвера који је предвиђен за оперативни систем који се користи на серверу. За већину намена ВПС има функционалности физичког сервера. Због платформе виртуелног рачунара ови сервери се јако лако конфигуришу и инсталирају. Цене изнајмљивања ВПС су значајно мање од изнајмљивања физичког сервера, док перформансе могу бити приближне физичком решењу за хостинг.



Слика 29. Архитектура ВПС

Предност у избору ВПС се огледа у већем нивоу сигурности, јасно дефинисаним ресурсима на располагању и могућност инсталације и подешавања софтвера на серверу у складу са потребама.

3 Преглед постојећих решења

Тренутно постоји велики броја доступних решења за заштиту изворног кода. Одређен броја решења је бесплатан док други део представљају комерцијална решења. Према начину заштите изворног кода ова решења можемо поделити у две основне групе:

- Алати за кодовање/шифровање изворног кода
- Алати за маскирање изворног кода.

Оба приступа имају своје предности и мане. Анализом постојећих решења долази се до закључка да ни једна метода не нуди потпуну заштиту.

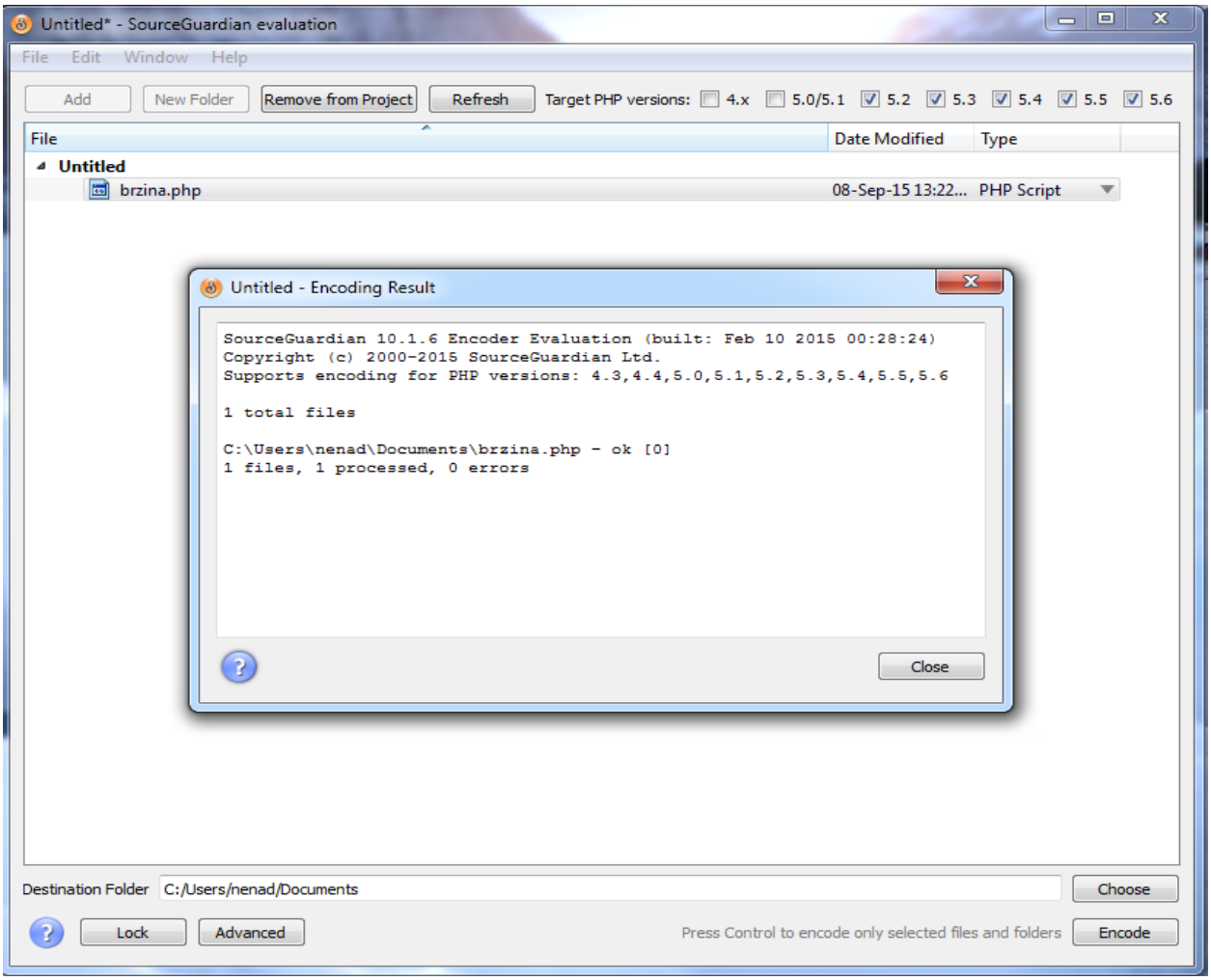
3.1 Кодовање/шифровање изворног кода

Кодовање или шифровање изворног кода су двосмерне методе конвертовања али се суштински разликују. Шифровање се базира на скривеном параметру (кључ) који се користи у процесу конверзије из отвореног у шифровани блок. Обично је познат алгоритам који се користи али то није увек потребно. Кодовање са друге стране преставља конверзију у којој се користи познати алгоритам као и параметри за конверзију тако да свако коме је познат алгоритам може без проблема извршити повратну конверзију у оригинални формат.

Већина PHP кодера се у суштини понаша као механизам за шифровање. Они садрже тајну компоненту која спречава лаку конверзију у отворени формат као и анализу. Оваква конверзија оправдава затворени код механизама за шифровање. Управо овакав затворени код ствара зависност од треће стране тј. твораца механизма за кодовање/шифровање.

Тренутно на тржишту постоји одређен број комерцијалних решења која врше услугу дистрибуције механизама и алата за заштиту изворног кода шифровањем или кодовањем. Међу најпопуларнијим налази се Зенд чувар који врши шифровање конвертовањем или компајлирањем изворног кода у бајт код [45]. Сличан метод користи још један популаран комерцијални алат за заштиту заштитник извора „*SourceGuardian*“ [46], додатно овај алат има могућност ограничавања извршавања на одређену ИП адресу. У класи раније споменутих решења налази се и „*ionCube PHP*

Encoder“ [47] ово решење врши компајлирање у бајт код и тиме штити изворни код од крађе, неовлаштене измене или копирања. Заштита скрипти и изворног кода врши се употребом апликације инсталиране на рачунар.



Слика 30. Приказ након процеса заштите скрипте

Овако заштићена скрипта се поставља на сервер. Употреба заштићене скрипте се врши уз помоћ екстензије инсталиране на серверу.

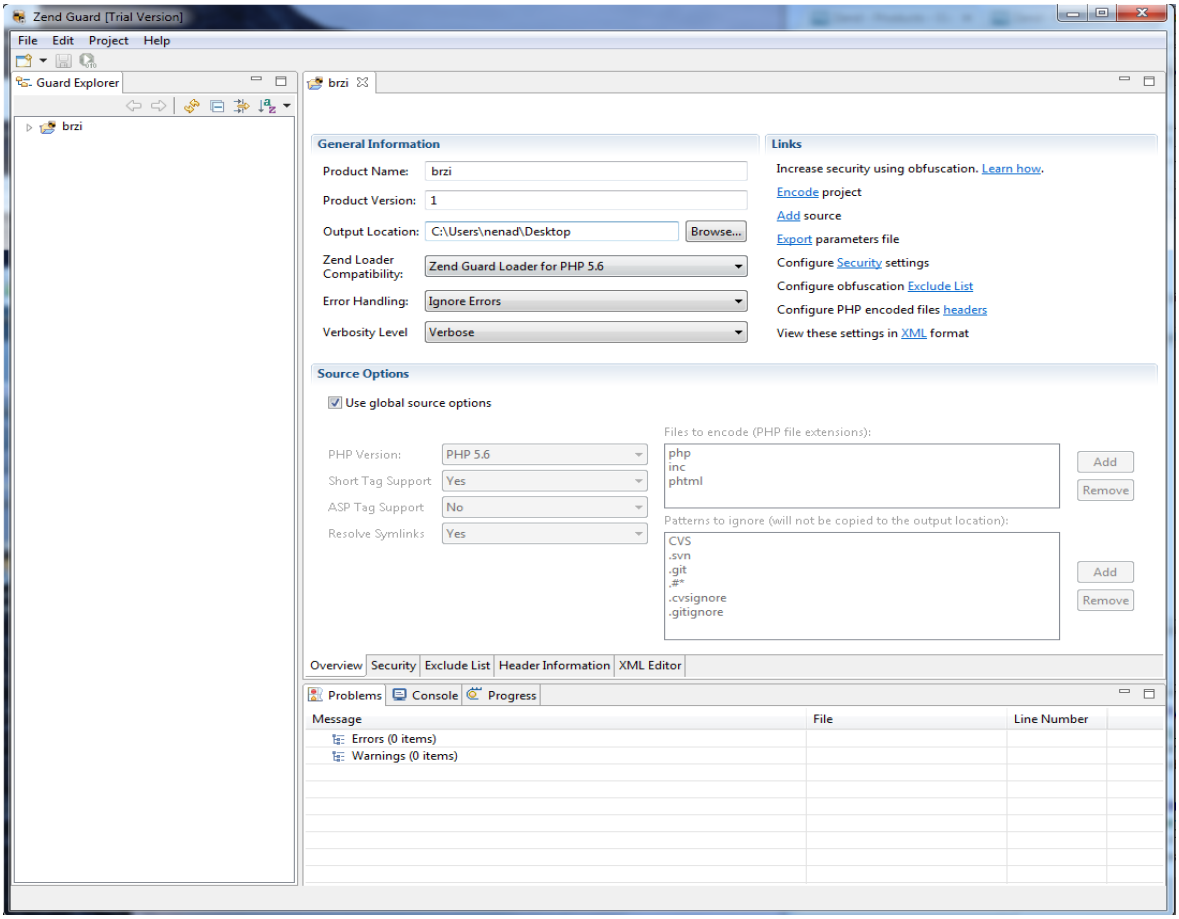
SourceGuardian

SourceGuardian Loader Support	enabled
SourceGuardian Loader Version	10.1.5
SourceGuardian Loader Build Number	0x00000016
phpSHIELD Support	enabled

Directive	Local Value	Master Value
sourceguardian.restrict_unencoded	0	0

Слика 31. Приказ регистроване екстензије у *PHP*

Овакав начин заштите нуди могућност ограничавања извршавања скрипте на одређен временски период, одређен домен, закључавање извршавања на одређену локалну адресу.



Слика 32. Подешавање параметара заштите

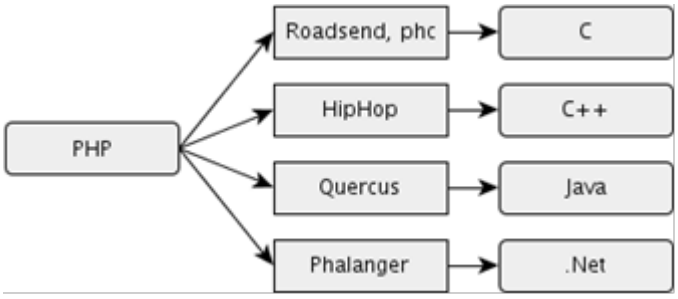
Приметно је да након заштите скрипте долази до значајне измене у њеној величини. Одређени део кода у заштићеној скрипти заузима и метода провере да ли постоји

екстензија на серверу као у одговор у случају грешке. На овој методи провере примењена је техника маскирања изворног кода.

```
<?php @"SourceGuardian"; //v10.1.6
if (!function_exists('sg_load')) {
    $__v = phpversion();
    $__x = explode('.', $__v);
    $__n = $__x[0] * 10000 + $__x[1] * 100 + $__x[2];
    $__u = strtolower(substr(php_uname(), 0, 3));
    $__f = $__f0 = 'ixed.' . substr($__v, 0, strpos($__v, '.', 3)) . '.' . $__u;
    $__ff = $__ff0 = 'ixed.' . $__v . '.' . $__u;
    $__ed = ini_get('extension_dir');
    if (!$__e = realpath($__ed)) die('extension_dir does not exist: ' . $__ed);
    if ($__n < 50205) {
        $__d = getcwd();
        if (@$__d[1] == ':') {
            $__d = str_replace('\',' / ',substr($__d,2));$__e=str_replace('\',' / ',substr($__e,2));}$__e.=($__h=str
_repeat(' / . . ',substr_count($__e,' / ')));$__f=' / ixed / '.$__f;$__ff=' / ixed / '.$__ff;while(!file_exists($__
e.$__d.$__ff) && !file_exists($__e.$__d.$__f) && strlen($__d)>1){$__d=dirname($__d);}if (file_exists($__e.$__d.$__f
f)) dl($__h.$__d.$__ff);else if (file_exists($__e.$__d.$__f)) dl($__h.$__d.$__f);}if(!function_exists('sg_load')){if
(file_exists($__e.' / '.$__ff0)) dl($__ff0);else if(file_exists($__e.' / '.$__f0)) dl($__f0);}if(!function_exists('s
g_load')){die('PHPscript < B > '.__FILE__.' < / B > isprotected by < AHREF = "http://www.sourceguardian.com/" > Sour
ceGuardian < / A > and requirestheSourceGuardianloader < B > '.$__f0.' < / B > . TheSourceGuardianloaderhasnotbeenin
stalled, or isnotinstalledcorrectly . PleasefindtherequiredloaderwithinSourceGuardianinstallationdirectory or visitt
he < AHREF = "http://www.sourceguardian.com/ixeds/" > SourceGuardianphpencoder < / A > sitetodownloadit . ');exit
();}}return sg_load('C2064618C799B627AAQAAAARAAAABHgAAACABAAAAAAAAAD / LXiVg + Gt92BMl0WyNbHVeplBxDPjVpmlN43VhHXB0 /
pBAVIjyrCgQ2eVUtM79M1zogfNhn1 / gRV8SjWyJ62G7fYn3vtCEGD9VxCEQKQhWD2QMC3RhltuFcu5WAszzCcpX7SGhM4oq3Wb5CWuQd4uYDX214 +
TqU6OBAAAAOgAAABzSyKd / cOj7X + ueqH5i1WTDLLGHV / 8WGs3Dy5CcGdlftqeBj4qYSBSfBzmDFH8XjzmTUk09mqNB5OL8Bn / VoBrFV0H31o
2HoHXwCuW5cCfjMC2p3lfKm2COcuSybmu + jWqmc1izG6MeVlbyYvucfAT / 7oQbzntSOv6jwXEsMRTaIpudY8rjahuwvOOir3ibyzXv0GkdLNxic8
BOFG37wfF74ctNIKqUUm / 8hSyh893q8bDnRemJjtaNO6C8ju / tvv14S / za + nFpFEf5bsZqK2GrxjZrnWebj163CXuHn1lhUjM3tG87mnWBQA
AAAABAABbgtDFgzET9l2XZnBV2hMCyg46yYfb + ehqzG24rRWVepBpWRZgri7SUpkvw6W5uIurzGNs8iSAAN + ryGKW / CkN259lgbHEfNZ6UOM9S
i9mMZEHAq + /F1D6nnZObm3AQe8OYX8RBclrDqeLMHQXQ39pEPoesN9q8I2 + SH + r6e7yKypAHeSU1Dk7RaW / AlhAOXokv2yfvdO5AM9d6NkgX
UrqQm6nPybJCCzfDGO13GrPHaKorPZm7prNlaRJgstmKv6LbFmwtEfvSUk2Qpj7JG + s6axDzPTPN2zvBjwVs1Af5L7rw + UYuHtlrkhEipMHiOE6F
FhCiIG6oBpF6u8RbYMVNAAAAMAAAAAapUE05wMrgbq / LUhbd7y7LwBUA4RzBn15cpJMLiiPBN7nXnsc083KZGsDJmGgDlIwJ0rFQzFaFkwFjpmVli7
CKw / BoUqv5dfbPtY1OwuyTfpB3daEyxyU8shLko0ndVxbTWpkQI5Mer9ATkwz9fVJeTwn8w1Lco6tzN1pz0IKT5xSkQTwhscgFrnsahGKH3H93vBNk
6P / eSYiAnx3UEhyD3tPetjrFH6 / DOqgERrxdjD9A4RbiUwBbcO200ofTm01AAAAwAAAAEmXD3XvFBM / rderS2aReLyjml7cwbU94lwgUWSx8Dm
mSLbi89JlGq862HtXozhsdu8C5uEY11F0iBhm5PplbqhIoxtAizxRZgkUVNQ2f2mgujfqmiUOKUKzp + 05mz2yGJNQC7 + 4USSmMPXAeHCZ8NS9zvC
v8HIgXtg93Ten30AZyQKcZ8A9NXLYXVwzvBHhTVaIN + frCA8s3xrbeTQB63fPHT7An8THO + MzBBYtwGNOYgcKjmbIQwBUSeIu0 + lteQAAAAA =
');
?>
```

Слика 33. Приказ шифрованог кода са SourceGuardian

Тренутно постоји велики број комерцијалних и бесплатних решења која врше услугу дешифровања кода шифрованог уз помоћ тренутно популарних комерцијалних решења. Тестирано је три решења која се лазе на интернету. Прво тестирано решење [48] врши успешно дешифровање популарних комерцијалних решења као што је Зенд чувар, „*SourceGuardian*“ и „*ionCube PHP Encoder*“. Још једно квалитетно решење „*DeZender*“ [49] нуди могућност дешифровања фајлова као и могућност декодирања маскираних фајлова.



Слика 34. Алати за компајлирање у друге програмске језике

У групу која врши заштиту изворног кода кодовањем или шифровањем можемо уврстити и алате који врше компајлирање у неки други програмски језик. Овај начин такође може убрзати извршавање или извршити превођење у машински.

3.2 Прикривање изворног кода

Прикривање или маскирање изворног кода је метода у којој се изворни код трансформише у збуњујући и тешко разумљив код а при томе се задржава функционалност оригиналног изворног кода. У процесу маскирања изворног кода потребно је имати што већи ниво не препознатљивости а при томе очувати што боље перформансе извршавања.

Велики број истраживања је вршен на тему маскирања изворног кода. Вршена су теоријска истраживања [50] [51] [52] [53] као и анализе практичних решења маскирања изворног кода [54] [55] [56].

У примени маскирања изворног кода постоји неколико приступа, ови приступи се могу користити појединачно или комбиновати.

```
<?php
${"G\x4c\x4f\x42AL\x53"}["gp\x6f\x78r\x75\x72\x62\x61p\x72"]="\x73\x75\x6d";
${"\x47L\x4f\x42\x41\x4c\x53"}["\x69hj\x7a\x67q"]="v\x61\x72\x31";
${"\x47\x4cO\x42A\x4c\x53"}["\x70\x6efs\x62\x68qmv"]="\x76ar2";
$nkfqxxhtb="\x73u\x6d";
${${"\x47\x4c\x4fB\x41\x4cS"}["\x69\x68\x6a\x7agq"]}=3;
$rioicuswhkf="\x76\x61r\x32";
${${"\x47\x4c\x4fB\x41L\x53"}["\x70\x6e\x66\x73\x62\x68\x71mv"]}=5;
$zjhjlyoakgl="\x76\x61\x721";
${$nkfqxxhtb}=${$zjhjlyoakgl}+${$rioicuswhkf};
echo${${"G\x4cOB\x41LS"}["\x67\x70\x6f\x78ru\x72\x62a\x70\x72"]};
?>
```

Слика 35. Приказ маскираног кода

Случај маскираног кода на први погледа делује нечитљиво, али након једноставне конверзије хексадецималних вредности код је много читљивији.

```
<?php
$ {"GLOBALS"} ["gpoxrurbapr"] = "sum";
$ {"GLOBALS"}["ihjzgq"] = "var1";
$ {"GLOBALS"}["pnfsbhqmv"] = "var2";
$nkfqxxhtb = "sum";
$ {$ {"GLOBALS"}["ihjzgq"]} = 3;
$rioicuswhkf = "var2";
$ {$ {"GLOBALS"}["pnfsbhqmv"]} = 5;
$zjhjlyoakgl = "var1";
$ {$nkfqxxhtb} = $ {$zjhjlyoakgl} + $ {$rioicuswhkf};
echo $ {$ {"GLOBALS"}["gpoxrurbapr"]};
?>
```

Слика 36. Дешифрован код

Маскирање података је још једна метода за маскирање изворног кода. Ова метода мења начин на који се подаци смештај у меморију. Маскирање података се обавља на следећим методама [57]:

- Промена улоге података (локални у глобалне)
- Промена типа енкодирања, замена вредности изразом
- Спајање више промењивих у низове

Одређена истраживања [53] [58] [59] долазе до закључка да маскирање изворног кода представља значајан криптографски модел са широком применом. Програм или изворни код који се маскира би требао да има својства црне кутије која без обзира на измене на коду мора давати предвиђени излаз за одређени улаз. Тренутно постоје истраживања којима се маскирање кода побољшава уз помоћ хардвера [60].

Маскирање изворног кода може представљати само делимично решење. *PHP* садржи грешке и рањивости које се могу искористити и на тај начин се може доћи до заштићеног кода. Ове грешке и рањивости се такође могу изманипулисати контролисаним улазима и генерисањем грешки у *PHP*.

3.3 Рачунарство од поверења (енгл. Trusted Computing)

Рачунари данас у основи представљају отворене платформе које кориснику дају слободу избора који софтвер ће користити, могућност да чита, уписује и брише

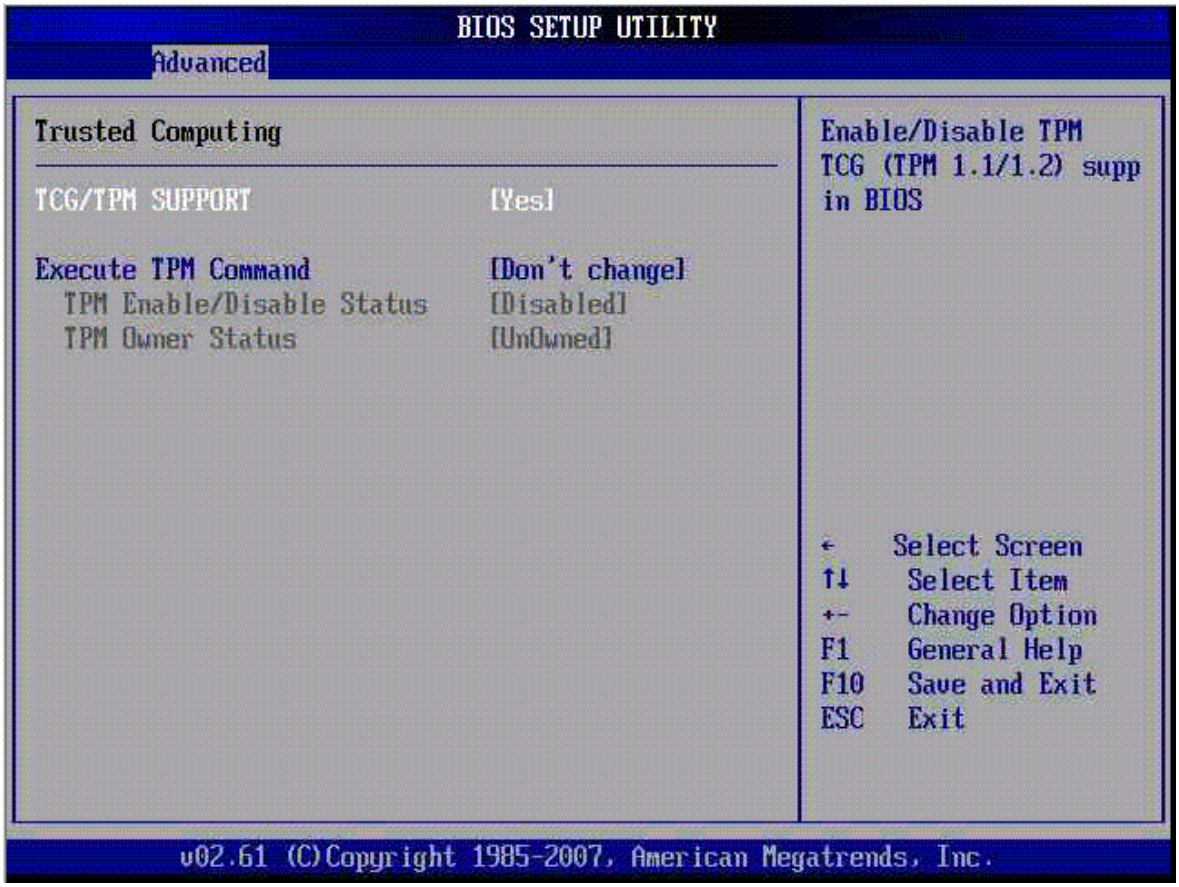
податке који се налазе на уређају. Овакав ниво слободе довео је до одређених проблема, као на пример:

- Опасност по корисника, као су отворене платформе подложне нападима различитих вируса, тројанаца и шпијунског софтвера као и различитим програмским решењима које могу прикупљати информације о кориснику.
- Опасност по мрежу, како је могуће да је платформа заражена одређеним злонамерним софтвером може доћи до напада ускраћивањем услуге, ширења заразе путем мреже и угрожавање осталих рачунара у мрежи.
- Опасност по дистрибутере мултимедијалног садржаја, ауторе апликација и софтверских решења је отворена платформа дозвољава дељење без ограничења или утицаја на квалитет.

Да би се постигао већи ниво безбедности, као и могућност заштите софтвера и садржаја од пиратизовања и неовлаштене употребе потребна је затворена архитектура рачунара која ће на хардверском нивоу обезбедити чување кључеве као и процес шифровања. Кључеви и процес шифровања не би требали бити под контролом корисника. Овакво решење може омогућити серверима и софтверским решењима да имају платформу од поверења. Ова платформа може спречити цурење информација, дељење мултимедијалног садржаја или неовлаштену употребу програмских решења. Платформа која нуди овакав ниво заштите ипак има и негативну конотацију која се може сажети у питањима цензуре, личне слободе и власништва. Рачунарство од поверења узима контролу од корисника и даје је дистрибутерима садржаја и софтверским компанијама.

Рачунарство од поверења представља решење затворене рачунарске архитектуре која гарантује сигуран рад апликација, сигурну комуникацију између апликација и апликација са серверима. У најснажнијем облику обезбеђује се шифровање комуникације између рачунара и периферија као што су тастатура, миш и монитор. Кључеви за шифровање су уграђени у хардвер уређаја и нису доступни кориснику. Рачунар покреће оперативни систем само ако може да потврди његов идентитет и да верификује његов интегритет. Сигурна комуникација система и сервера је гарантована такође потврдом интегритета и идентитета чак и пре покретања комуникације са апликацијом.

Хардверска подршка представља важан део имплементације рачунарства од поверења. Поред хардверске подршке која је ту због кључева за шифровање, такође је потребан добар систем заштите меморије и отпорност на физичке преправке. На почетку иницирања оперативног система контрола може бити предата програму који се налази у меморији која се може само читати (енгл. *ROM*) на потврђивање хеш вредности оперативног система због потврђивања идентитета и интегритета [61]. На исти начин се може потврдити поверење програмима прије дозвољавања њиховог покретања [62].

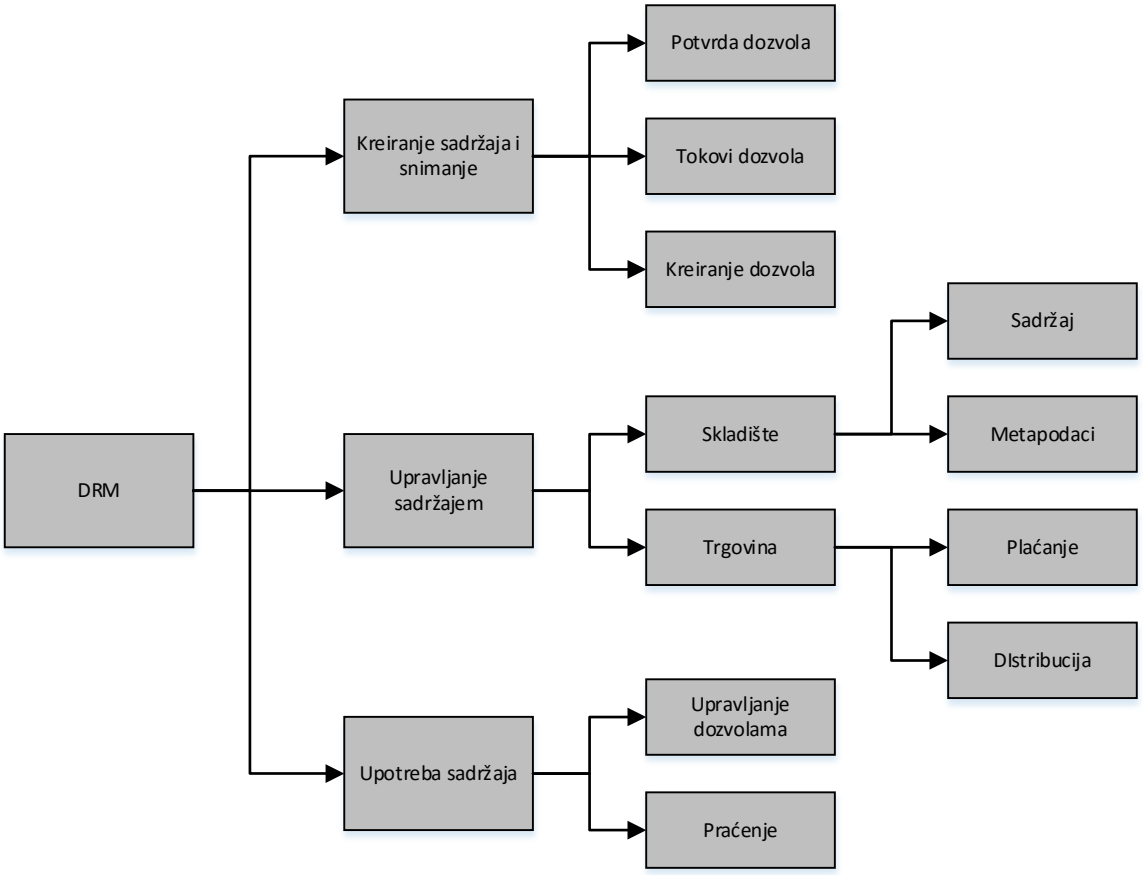


Слика 37. Активирање функционалности рачунарства од поверења

Криптографски кључеви који су хардверу задужени су за извођење сигурних операција. У тренутку производње криптографски кључ се генерише и смешта се на хардвер. Овај кључ се никад не шаље некој другој компоненти, хардвер је направљен на такав начин да се отежава добијање кључа реверзним инжењерингом, чак и за власника система. Програм може само проследити податке који су шифровани хардверској компоненти на дешифровање. Дешифровани податак ће се затим чувати

у заштићеној меморији, чинећи тиме да буде не доступан осталим апликацијама система.

Рана примена рачунарства од поверења мотивисана је управљањем заштитом права на дигитални мултимедијални садржај (енгл. *Digital Rights Management*) [63]. Музика и видео материјал су шифровани и могуће вршити репродукцију само на препознатим апликацијама на одређеној платформи. Апликације нису дозвољавале корисницима да праве копије. Такође апликације могу да ограниче број репродукција као и време репродукције.



Слика 38. Функционална архитектура управљања заштитом права на дигитални садржај

Овакав механизам заштите садржаја омогућивао је заштиту одређено време. Проблем је настајао након првог проналажења слабости или успешног заобилажења механизма заштите који се могао применити на сав заштићени садржај.

Примена рачунарства од поверења протеже се даље од заштите мултимедијалног садржаја. У данашње време много важнију имовину у електронском окружењу чине документи и електронска пошта. Заштита електронске поште може бити у форми

поште коју није могуће проследити, која се уништава након одређеног времена, забрањује штампање садржаја поруке и слично [64]. Организације могу применити систем заштите над документима да би спречили цурење информација применом система рачунарства од поверења. Документи креирани на рачунарима који су верификовани имају могућност читања, мењања и снимања документа, на сваком другом рачунару документ је неупотребљив. Овакав метод заштите докумената такође може омогућити сигурну размену докумената између две стране путем канала комуникације који није заштићен [65].

Примена рачунарства од поверења у мрежном окружењу може обезбедити да да само ауторизовани рачунари могу приступити мрежи или бити њени чланови. Тренутне технике обезбеђивања приступа које се користе, нпр. идентификовање физичке адресе мрежног адаптера, нису адекватне јер се овај параметар може лажирати. Употребом рачунарства од поверења обезбеђује се да само платформе са одређеним хардвером могу приступити ресурсима. Протоколом потврђивања (енгл. *Attestation protocol*) може се идентификовати и обезбедити да платформа користи одобрен софтвер.

Безбедносна баријера (енгл. *Firewall*) представља још једну примену рачунарства од поверења. Изворно, баријера може претпоставити да може веровати свим члановима унутрашње мреже. Међутим, великим развојем бежичних приступних тачака, виртуелних приватних мрежа и тунеловања смањује се разлика између унутрашњих и спољашњих чланова мреже по претњама. Такође проблем представља код дистрибуираних безбедносних баријера сигурност да су подешени према организационој полиси, јер корисник може на рачунару извршити измену локалне полисе. Ово се може избећи имплементацијом централизоване мрежне заштите. Али у случају дистрибуираних мрежних баријера рачунарство од поверења неће дозволити мењања параметара и уз комуникацију у мрежи обезбедиће како заштиту мреже од клијента тако и заштиту клијента од мреже.

Многи сервиси као што је Амазон и слично базирају своје услуге на поверењу корисника. Трговина која се обавља између корисника на сервисима као што је *EBay* ослања се на поштење корисника у трговини, као и на систем оцењивања за утврђивање поверења. Проблем је јер у таквом систему корисник који има лоше поверење може креирати нови налог и тиме утицати на препознавање. Такође

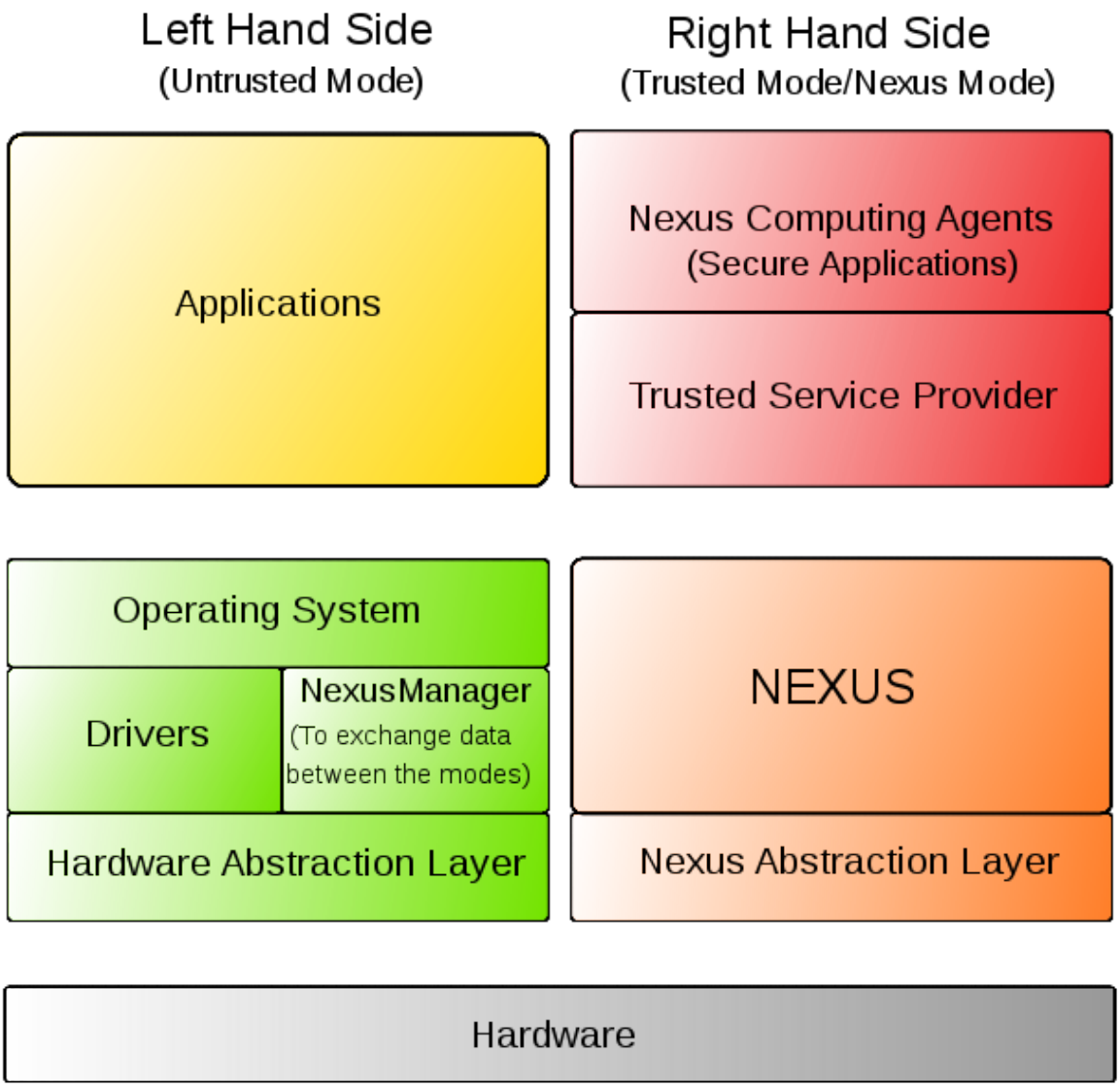
корисници могу поставити лажне информације на сервисе као што је википедија (енгл. *Wikipedia*) а затим након блокирања налога могу једноставно креирати нови налог. Рачунарство од поверења обезбеђује да један идентитет корисника буде везан за једну хардверску платформу. Овакав систем обезбеђује једноставну идентификацију једне платформе и њену повезаност са корисником. Проблем који може настати у оваквој примени је управо везаност корисника од поверења за хардверску платформу. Ако би дошло до куповине половног уређаја корисник би имао проблем за идентификовањем поверења јер би оно било везано за претходног власника.

Да би рачунарство од поверења исправно функционисало потребна је да се за почетак у производњи хардвер производи са паром кључева јавним и приватним. Приватни кључ се чува у чипу и никад се не објављује. Чип је отпоран на физичке утицаје и нападе и пре ће доћи до уништавања чипа него до проналажења приватног кључа. Меморија мора бити заштићена да би се спречили софтверски напади у покушају добијања приватног кључа у процесу операција потписивања. Апликације се верификују серверу шаљући хардверски јавни кључ заједно са дигиталним потписом апликације. Сервер проверава да ли је хардвер и апликација од поверења пре него што пошаље садржај. Заштита меморије мора бити хардверски имплементирана и јака да би спречила програме, према којима не постоји поверење, да јој приступе. Хардвер за рачунарство од поверења мора имати и сигурне улазно излазне канале. Ово је неопходно због предњи које чине хардверски уређаји који снимају куцање на тастатури, садржај екрана или звук који се репродукује.

Слика 39. Хардверска компонента за снимање садржаја

Сигуран улазно излазни канал комуникације обезбеђује да са улаз физички присутног корисника разликује од програма који покушава да се представи као корисник.

Мајкрософт је један од главних покретача примене рачунарства од поверења. На почетку Мајкрософт развија решење *Palladium* [66] које касније мења име у Сигурна рачунарска основа нове генерације (енгл. *Next Generation Secure Computing Base*). Решење на ком је Мајкрософт радио је нудило механизам укључивања рачунарства од поверења у постојеће системе [67]. Два режима рада би радила паралелно, сигурни режим и несигурни режим. Несигурни режим рада би био попут *Windows* оперативног система тренутно и корисник би имао потпуну слободу. Сигурни режим би био затворен. Корисник не би имао потребу да га користи до тренутка када треба да приступи садржају који је заштићен механизмом рачунарства од поверења [68]. Имплементација сигурне рачунарске основе нове генерације (*СРОНГ*) планирана је у *Vista* оперативни систем. Мајкрософт се ослонио на хардверски модел дизајниран од стране Групе за рачунарство од поверења (енгл. *Trusted Computing Group*) да би обезбедио паралелно окружење под називом Nexus које је требало да функционише уз матични оперативни систем. СРОНГ је требао да обезбеђује и дистрибуира дозволе и привилегије везане за употребу података.



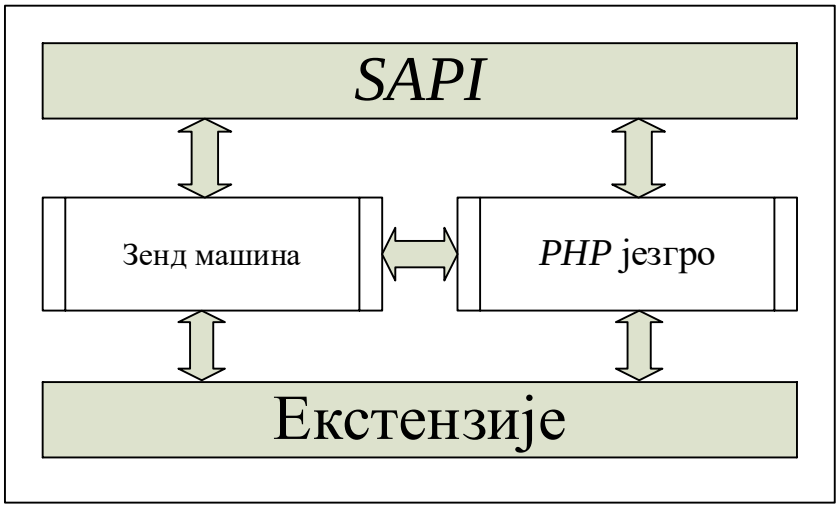
Слика 40. Сигурна рачунарска основа нове генерације (енгл. *Next Generation Secure Computing Base*) [70]

Мајкрософт није укључио СРОНГ у *Vista* оперативни систем и сужава развој свог решења за рачунарство од поверења и усмерава га у правцу имплементације у виртуализацији на серверским платформама [69]. Слабости овог система су пронађене и објављене у оквиру рада о темпираним нападима на рачунарске уређаје [70].

Највеће критике на рачун рачунарства од поверења тичу се могућност да компаније стекну монополске позиције. У одређени сегментима данас постоје проблеми са компатибилности између стандарда формата докумената. Увођење рачунарства од поверења може изазвати још већи проблем у овој области па и ситуацију да је корисник принуђен да користи одређено софтверско решење.

4 Модел предложеног решења

PHP је сачињен из две засебне целине. Целине које чине *PHP* су Зенд машина (енгл. *Zend Engine*) и *PHP* језгро.



Слика 41. Архитектура *PHP*

Зенд машина се налази на најнижем нивоу *PHP* језгра. Једна од основних функција коју Зенд машина обавља је превођење, за људе читљивог кода, у Зенд оперативни код и извршавање овог кода. Зенд машина такође управља опсегом промењивих, меморијом и упућује позиве функцијама. Други део представља *PHP* језгро. *PHP* управља комуникацијом и везама са апликативним програмским интерфејсом сервера *PHP*. *API* сервера или *SAPI* слој односи се на повезивање са окружењем веб сервера, *Apache* веб сервер, *IIS* или неки други веб сервер. *SAPI* омогућава уз помоћ уграђеног контролног слоја сигурни режим (*safe_mode*) и ограничавање који фајлови могу приступити PHP (*open_basedir*). Приликом покретања *API* сервера покрећу се подсистеми језгра. При крају процедура покретања учитавају се кодови појединачних екстензија. Учитава се код сваке екстензије појединачно, и позива процедуру иницирања модула (*MINIT*). Ово омогућава свакој екстензији да иницира своје интерне промењиве, да региструје управљање ресурсима и да региструје своје функције у Зенд машини. Управо због овога омогућено је да Зенд препозна функције из скрипти и успешно изврши одређене задатке.

У наставку *PHP* чека на захтев SAPI слоја за обраду странице. У случају стандардног излазног интерфејса (*CGI*) или командно линијског излаза (*CLI*) упућивање захтева се

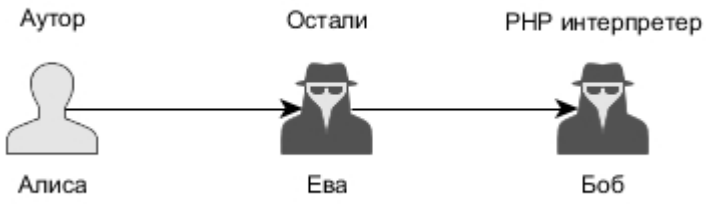
одвија само једном. У случају „*Apache*“ веб сервера, Микрософтовог „*IIS*“ сервера, упућивање захтева се одвија од стране удаљеног корисника и понавља се неодређен број пута. Без обзира на то како од које стране стижу захтеви, PHP започиње давањем захтева Зенд машини за успостављањем окружења у ком ће се скрипта извршавати и затим позива захтев за иницирањем сваке екстензије (*RINIT* функција). *RINIT* функција даје екстензијама могућност иницирања промењивих, дефинисања потребних ресурса и извршавање предефинисаних задатака.

По добијању захтева, Зенд машина преводи PHP скрипту у блокове и затим у опкод који може проследити и извршити. У случају да је за опкод потребна нека од екстензија Зенд машина ће прикупити параметре и привремено препустити контролу екстензији до краја извршавања. Ово омогућава подизање нивоа безбедности приликом употребе екстензије за заштиту изворног кода.

По завршетку извршавања скрипте, *PHP* позива функцију захтев за гашењем (*RSSHUTDOWN* функција) сваке екстензије. Затим, Зенд машина обавља чишћење које извршава функцију „*unset ( )*“ за сваку промењиву кориштену током претходног захтева. Након чишћења које завршава Зенд машина, *PHP* чека нови захтев или сигнал за гашење. У случају стандардног излазног интерфејса (*CGI*) или командно линијског излаза (*CLI*) не постоји команда следећег захтева па апликативни програмски интерфејс сервера започиње гашење. Током гашења *PHP* пролази кроз циклус позивања функције гашења за све екстензије, затим гаси језгро и своје подсистеме.

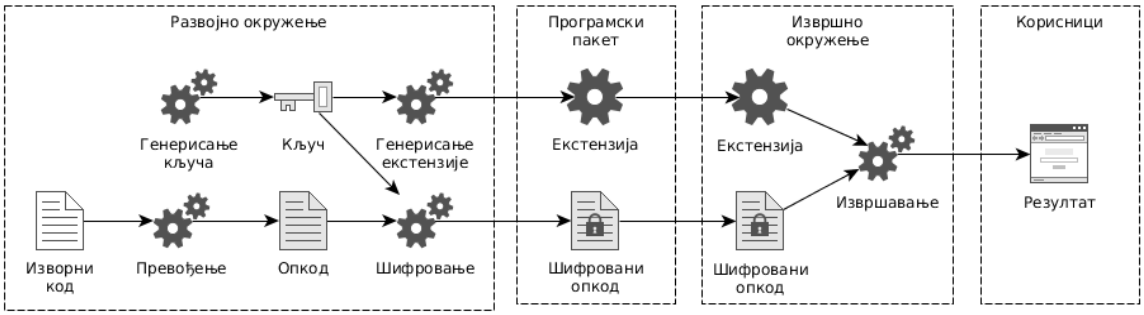
Предложено решење се не базира на поверењу треће стране и то је једна од кључних функција која издваја ово решење од постојећих решења. Предложено решење нуди заштиту на нивоу изворног кода као и извршног кода. У предложеном решењу две компоненте су тајне. Једну тајну компоненту може чинити алгоритам који се користи, мада то није обавезно. Тајни кључ чини компоненту која је тајна у систему како је пожељно према Крикоховим принципима. Додатни заштитни слој пружа нечитљивост компајлиране екстензије. Решење се базира на резултатима и сазнањима истраживања заштите изворног кода употребом криптолошких механизама [71]. Са аспекта криптологије заштита изворног кода може се гледати као успостављање канала сигурне комуникације између творца скрипте и *PHP* интерпретера. Инструкције, у

форми изворног кода, који представља тајну поруку, потребно је да се шифрују на такав начин да само интерпретер може да дешифрује/тумачи поруку/инструкције.



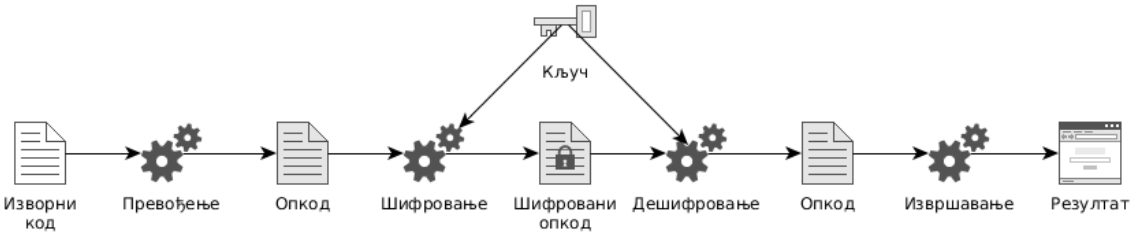
Слика 42. Представљање проблема - интерпретеру се не може веровати

У случају *PHP* интерпретера његов изворни код је слободно доступан и може представљати слабу карику у комуникацији. Злонамерни корисник може извршити превођење изворног кода интерпретера уз измене које му омогућавају да дође до опкода.



Слика 43. Модел затворене екстензије

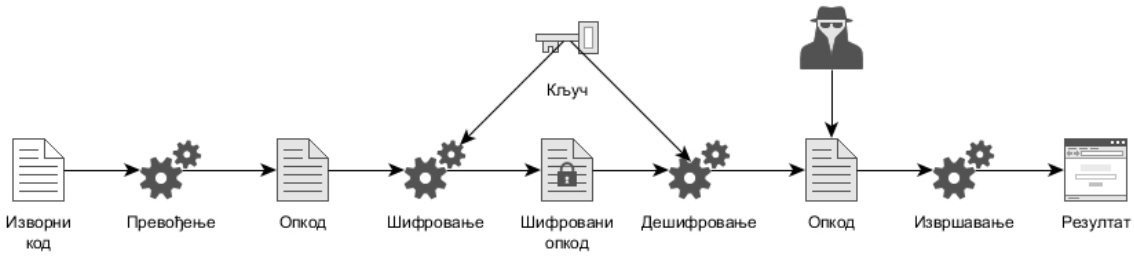
Модел за затвореном екстензијом решава проблем дистрибуције кључа. Проблем који овај модел има огледа се у отежаном извршавању као и потреби за динамичким учитавањем екстензије.



Слика 44. Развојни модел

Слика 45. Модел предложеног решења

Предложено решење чине две компоненте, компонента намењена за шифровање изворног кода и екстензија за *PHP*.



Слика 46. Различити нивои напада

Механизам за шифровање ради попут обичног интерпретатора и преводи изворни код у опкод, уз додатак да такав код шифрује одређеним алгоритмом и кључем. Шифровање које је коришћено у студији случаја је *AES*. Што се тиче алгоритма за

шифровање, теоријски могуће је користити било који симетрични шифарски алгоритам.

```
std::string sifrat(const std::string& ulaz, const std::string& kljuc, const std::string& inicijalnivektor)
{
    std::string izlaz;
    CryptoPP::CFB_Mode<CryptoPP::AES>::Encryption encryption((byte*)kljuc.c_str(), kljuc.length(), (byte*)inicijalnivektor.c_str());

    CryptoPP::StringSource encryptor(ulaz, true,
        new CryptoPP::StreamTransformationFilter(encryption,
            new CryptoPP::Base64Encoder(
                new CryptoPP::StringSink(izlaz),
                false
            )
        )
    );
    return izlaz;
}
```

Слика 47. Део кода механизма за шифровање

Шифровани код се снима у фајл уз додатак позивања функције екстензије за дешифровање. За исправно шифровање потребан је кључ дужине 32 бајта, и иницијални вектор дужине 16 бајта. *AES* је изабран за студију случаја јер је алгоритам који је у прошлости највише тестиран и испитивана његова сигурност [10].

```
<?php
    ppcp('kiJNVuAJtemhPEFUVbrpI3UaM2EZX8YZ7CcKOd/dFRd3nPdJ2KpmgtV24AK5l3+yID9KI/gFG8bqHuarJ9FjgA');
?>
```

Слика 48. Шифровани садржај новог фајла

Друга компонента система заштите је екстензија. Екстензија се компајлира и имплементира на сервер. Улога екстензије је дешифровање кода и прослеђивање на извршавање.

PHP последује велики број екстензија. Узимајући у обзир пар изузетака, функције PHP програмског језика су дефинисане унутар неке од екстензија. Изворно *PHP* долази са преко 80 уграђених екстензија, огроман број додатних екстензија може се пронаћи у „*PECL*“ складишту [72].

Постоји више метода за креирање екстензије. Основна метода се састоји из компајлиране екстензије уз помоћ садржаја три фајла. Први фајл је изворни фајл, и он садржи функције. Други фајл је фајл заглавља и он садржи референце које се користе од стране *PHP* за учитавање екстензије, и трећи фајл који је конфигурациони фајл чија је улога припрема екстензије за компајлирање у окружењу. У литератури се препоручује прављење костура екстензије на почетку [73]. Овај процес креира основне фајлове и садржај потребан за почетак.

На самом почетку извршавањем скрипте креира се директоријум са потребним фајловима за почетак прављења екстензије. Назив директоријума и основни параметри се генеришу на основу задатог имена екстензије.

```
nenad@debian:~$ ./ext_skel --extname=imeeksenzije
```

Слика 49. Генерисање директоријума и потребних фајлова

Ако није потребна провера постојања спољних фајлова заглавља, библиотека или функција из њих модул је скоро спреман за компајлирање. Потребно је само уклонити коментаре из конфигурационог фајла и може се започети компајлирање.

```
nenad@debian:~$ ./buildconf; ./configure --enable-imeekstenzije; make
```

Слика 50. Конфигурисање и компајлирање екстензије

Ознаке и дефиниције имена и верзије екстензије уписани су у фајлу „*php_imeekstenzije.h*“ и биће уметнути у дефинисања унутар Зенд машине. Да би дефинисали функције као и промењиве које ће екстензија користити потрено је креирати фајл који дефинише функције. Ово се обавља додавањем параметра код извршавања скрипте за прављење костура.

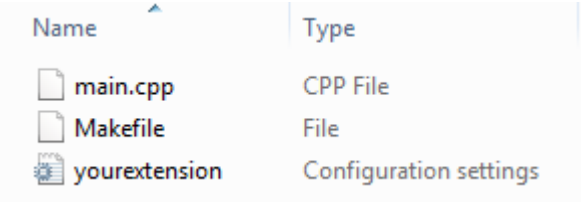
Скрипта генерише фајлове „*imeekstenzije.c*“ и „*php_imeekstenzije.h*“ имена ових фајлова се не мењају. Име функције је потребно декларисати у оквиру конфигурације.

```
55    module_imefunkcije()
```

Слика 51. Декларисање имена функције

Велики проблем код овакве имплементације екстензије је поприлично недокументована Зенд машина и висок ниво комплексности.

Следећи начина је употреба *PHP-CPP* библиотеке. Ова библиотека омогућава писање екстензије употребом *C++* програмског језика. Употребом *PHP-CPP* библиотеке могуће је радити са промењивим, низовима, функцијама, објектима класама једноставно као и у *PHP* скриптама. Такође могу се користити све могућности *C++* програмског језика. Библиотека долази са 3 фајла. Преузимањем компресоване датотеке и њеним распакивањем добијамо основу екстензије.



Слика 52. Садржај библиотеке

Фајл за креирање екстензије „*Makefile*“ садржи инструкције за компајлер. Потребне измене у овом фајлу се тичи измене промењиве име „*NAME*“.

```
21  NAME                =   PhPCodeProtection
```

Слика 53. Измена у фајлу

Потребно је као име конфигурационог фајла уписати назив екстензије, затим у фајл уписати назив екстензије.

PhPCodeProtection.ini

```
1   extension=PhPCodeProtection.so
```

Слика 54. Садржај конфигурационог фајла

Последњи фајл је у основи фајл задужен за имплементацију екстензије. Почетни фајл је празан и потребно је додати функције и класе које ће чинити екстензију.

main.cpp

```cpp
#include <phpcpp.h>

/**
 *  tell the compiler that the get_module is a pure C function
 */
extern "C" {

    /**
     *  Function that is called by PHP right after the PHP process
     *  has started, and that returns an address of an internal PHP
     *  strucure with all the details and features of your extension
     *
     *  @return void*   a pointer to an address that is understood by PHP
     */
    PHPCPP_EXPORT void *get_module()
    {
        // static(!) Php::Extension object that should stay in memory
        // for the entire duration of the process (that's why it's static)
        static Php::Extension extension("yourextension", "1.0");

        // @todo    add your own functions, classes, namespaces to the extension

        // return the extension
        return extension;
    }
}
```

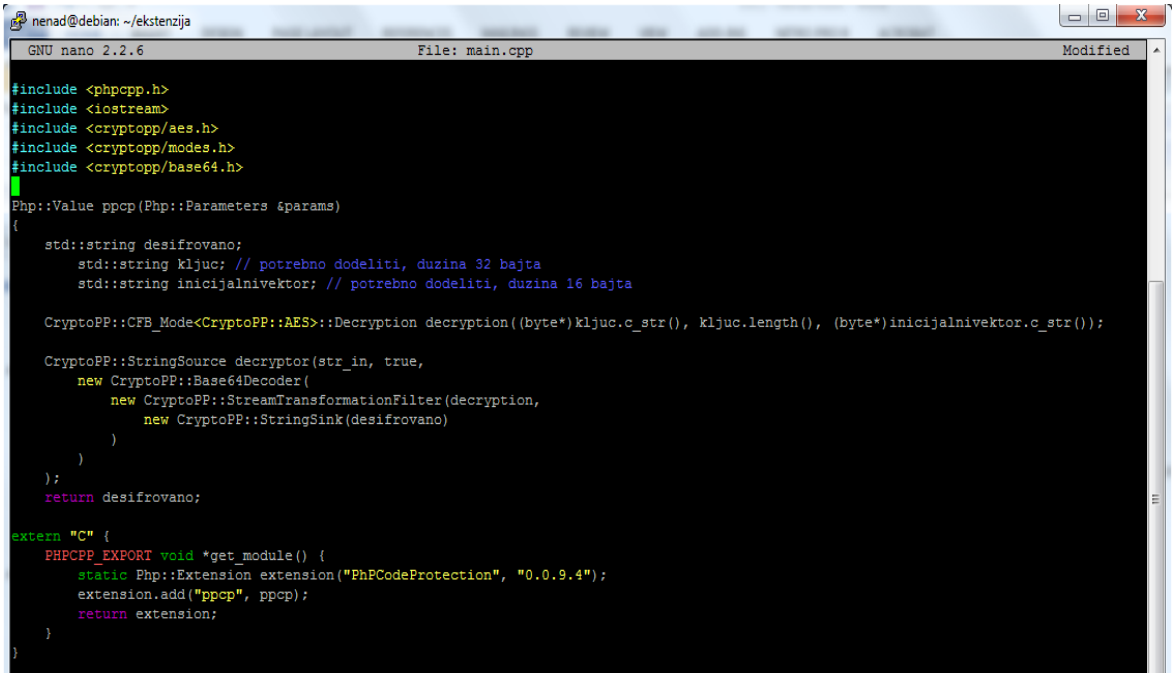
Слика 55. Приказ шаблона основног фајла

У имплементацији је могуће користити стандардне улазно/излазне операторе (<<) као и специјалне функције (*std::endl*). Креатори библиотеке ипак препоручују употребу „*std::cerr*“ као и употребу „*std::cout*“ [74].

Када се PHP користи као модул веб сервера, стандардна дефиниција излаза (*stdout*) се прослеђује на терминал. У комерцијалном окружењу терминал није активан, самим тиме ће прослеђени излаз бити изгубљен. Управо због овога у имплементацији екстензије се не користи „*stdout*“ функција.

PHP-CPP библиотека нуди „*Php::out*“ излаз који се може користити. Ова промењива је део добро познате класе „*std::ostream class*“

Поред библиотеке *PHP-CPP* за екстензију је потребно додатно увести и библиотеке „*Crypto++*“. Ова библиотека је бесплатна и јавно доступна [75]. У шаблону је потребно дефинисати кључ као иницијални вектор.



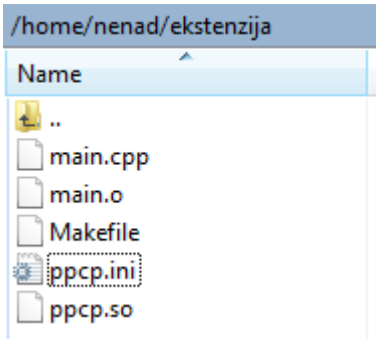
Слика 56. Шаблон екстензије

Након постављања вредности кључа и иницијалног вектора може се извршити компајлирање екстензије.

```
nenad@debian:~/ekstenzija$ make
g++ -Wall -c -O2 -std=c++11 -fpic -o main.o main.cpp
g++ -shared -o ppcp.so main.o -lphpcpp
```

Слика 57. Компајлирање екстензије

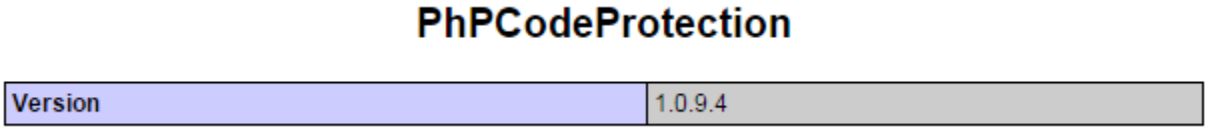
Након компајлирања екстензије у директоријуму добијамо нови фајл са екстензијом „*so*“ овај фајл је екстензија која се дистрибуира у комерцијало окружење.



Слика 58. Директоријум екстензије након компајлирања

У случају покушаја покретања заштићене скрипте а да није извршена имплементација екстензија добиће се порука о употреби непостојеће функције. Поред копирања екстензије на сервер потребно је укључити екстензију у конфигурационом фајлу *PHP*.

Проверу имплементације екстензије можемо извршити прегледом конфигурације *PHP*.



Слика 59. Информација о екстензији унутар PHP информација

Оваква метода креирања екстензије одабрана је због флексибилности и могућности лаке имплементације различитих криптографских алгоритама у екстензију.

У случају постојања потребе да се заштита користи на више сајтова, неопходно је имати више сервера са засебним конфигурацијама да би могли користити исту екстензију која је предвиђена за више сајтова. Једно решење које би се могло применити случају једног сервера и потребе за заштитом више сајтова је динамичко учитавање екстензије. *PHP* садржи уграђену функцију *dl()* која се може користити за учитавање екстензија [76]. Ово омогућава да *PHP* позивом нпр. *dl("pphp.so")* учита екстензију. Ова функција има ограничења због сигурности. Дељени хостинг омогућава да више корисника хостује своје сајтове на истом систему. Овакав систем је најкритичнија локација за злонамерну имплементацију ове функције. Злонамеран корисник може на написати скрипту или програм који ће имати могућност неовлаштеног приступа другим директоријумима.

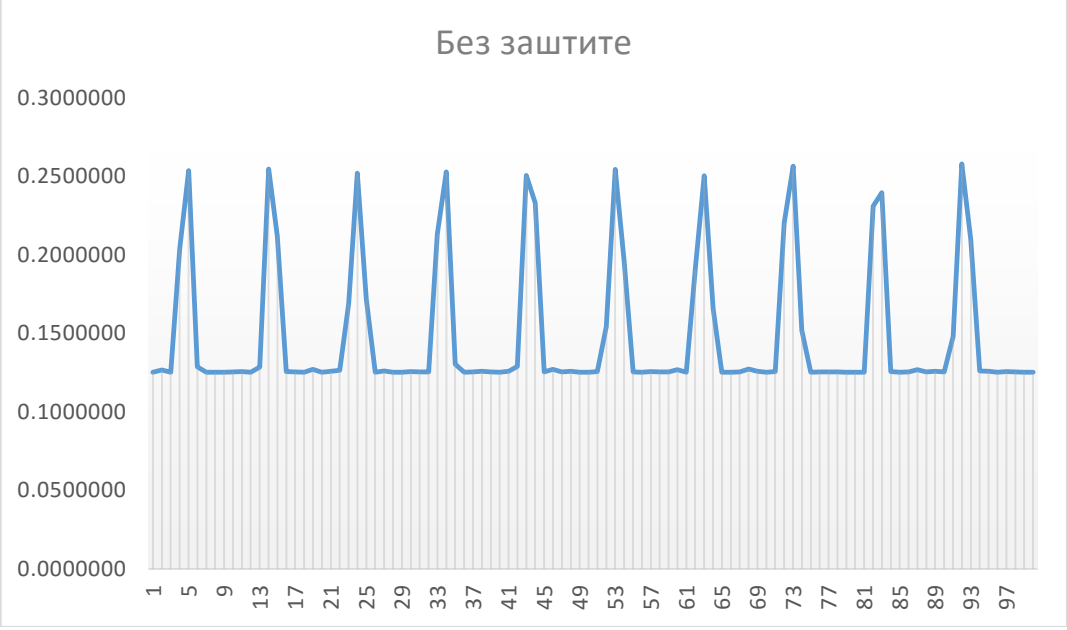
Проблем у понуђеном решењу је постојање могућности да се сними или ухвати садржај излаза из екстензије. Садржај који може бити ухваћен је опкод који је могуће анализирати реверзним инжењерингом. Оваква слабост постоји и у доступним комерцијалним решењима.

Виши ниво сигурности се може добити у случају затвореног интерпретера, чиме се спречава злоупотреба на нивоу опкода.

5 Експериментална анализа

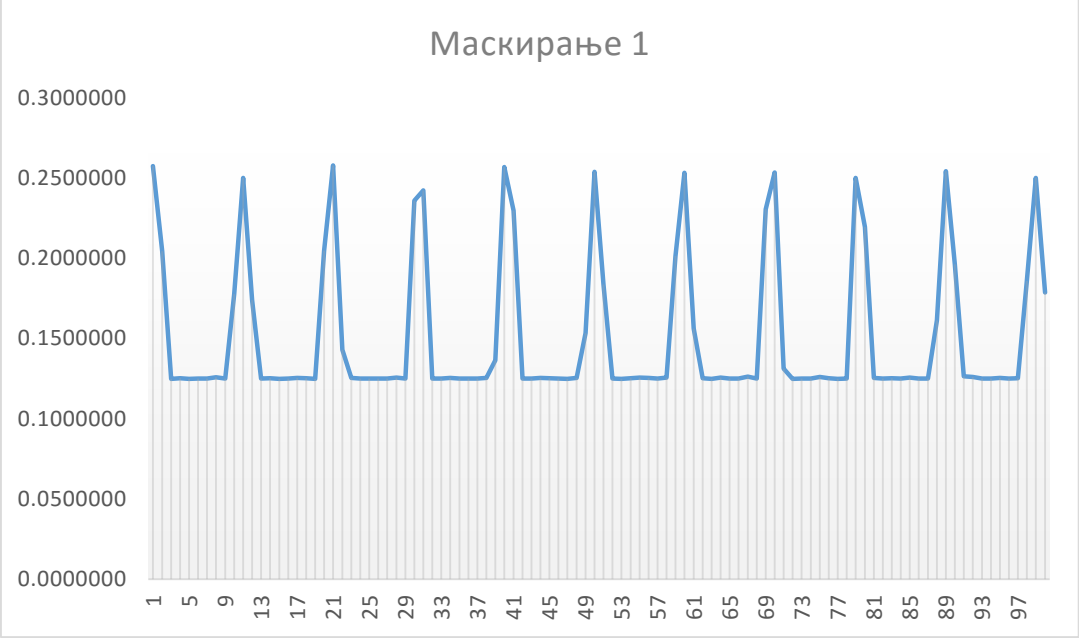
Почетна анализа која је извршена је анализа времена извршавања скрипте заштићене различитим доступним методама. Скрипта је заштићена употребом три методе маскирања изворног кода као и методом шифровања, на крају је урађено тестирање предложеног решења. Код које је тестиран је

```
<?php
$start = microtime(true);

$var1 = 3;
$var2 = 5;
$sum = $var1 + $var2;
echo $sum;

$end = microtime(true);
$time = number_format(($end - $start), 2);

echo 'Stranica je ucitana za ', $time, ' sekundi';
?>
```



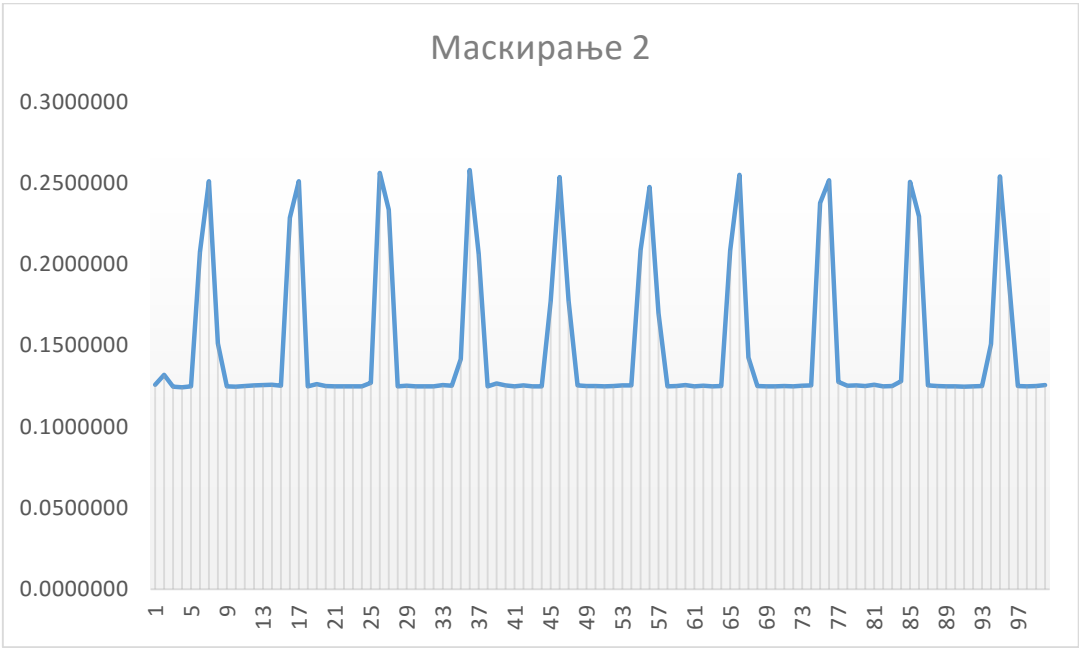
Слика 60. Извршавање скрипте без заштите

На слици је дат графички приказ времена извршавања тестне скрипте на серверу. Слика приказује 100 извршавања скрипте која није заштићена.



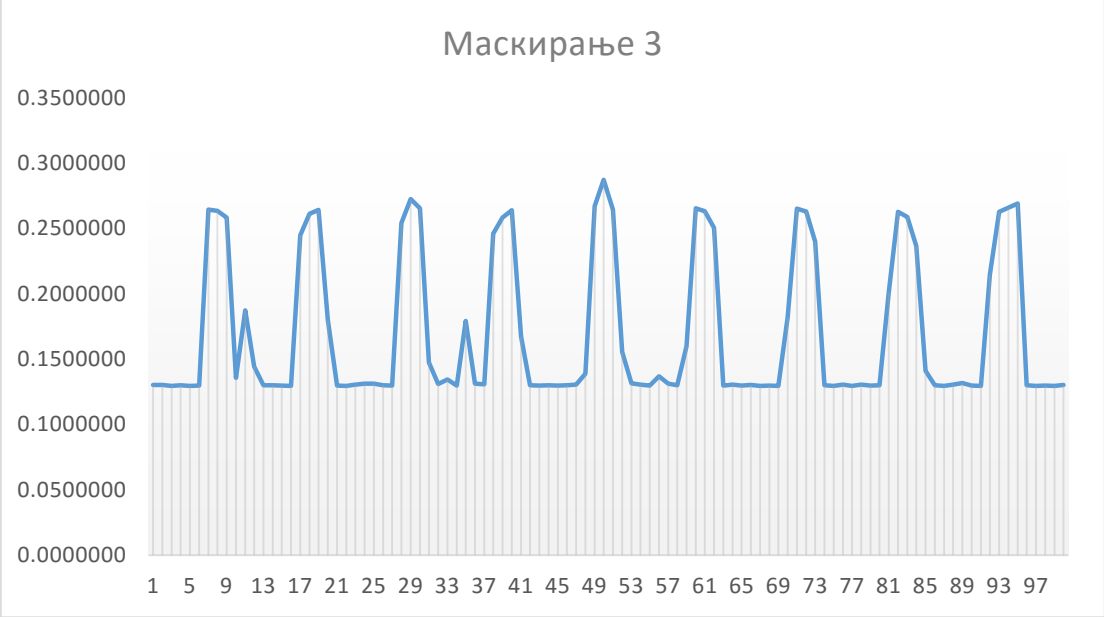
Слика 61. Извршавање скрипте маскираним кодом 1

Слика приказује 100 извршавања скрипте која је заштићена методом маскирања изворног кода јавно доступном методом 1.



Слика 62. Извршавање скрипте маскираним кодом 2

Слика приказује 100 извршавања скрипте која је заштићена методом маскирања изворног кода јавно доступном методом 2.

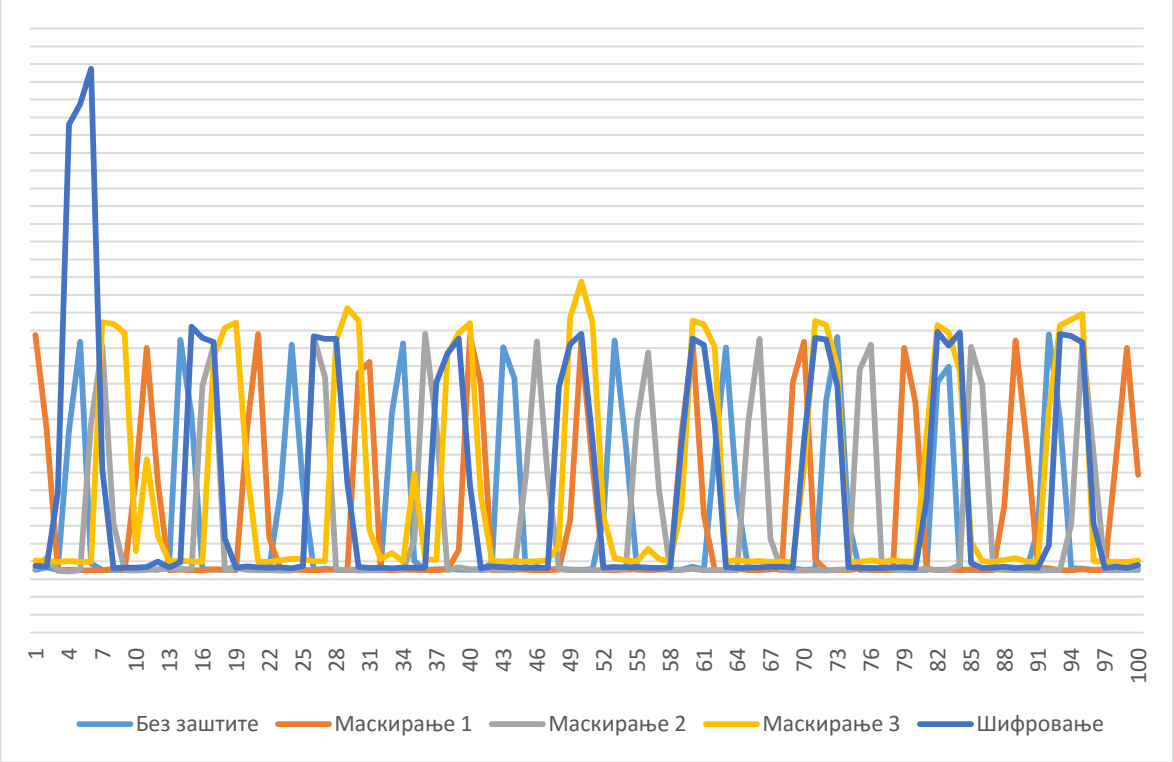


Слика 63. Извршавање скрипте маскираним кодом 3

Слика приказује 100 извршавања скрипте која је заштићена методом маскирања изворног кода јавно доступном методом 3.

Слика 64. Извршавање скрипте са шифрованим кодом

Слика приказује 100 извршавања скрипте која је заштићена методом шифровања изворног кода пробном верзијом комерцијалног решења [45].



Слика 65. Паралелни приказ извршавања скрипте

На слици је приказан резултат мерења 100 извршавања скрипте на серверу. Просечна времена дата си на табели.

Табела 4. Просечна времена извршавања доступних решења

Без заштите	Маскирање 1	Маскирање 2	Маскирање 3	Шифровање
0.1482685	0.1505777	0.14878057	0.170781069	0.169218663

Као што се види на слици време извршавања је приближно, уз мала одступања.

Слика 66. Време извршавања скрипте заштићене предложеним решењем

На слици је графички приказано време извршавања скрипте предложеним решењем. Анализом свих резултата долази се до закључка према ком време извршавања скрипте заштићене предложеним решењем има средњу вредност у поређењу са доступним решењима.

Уз имплементацију реалном систему предложено решење не показује спорије време извршавања у односу на доступна решења.

Додатно мерење је извршено да би се утврдила брзина извршавања уз задате функције заштите. За ту намену је коришћена функција *curl_**.

```
<?php
for ($i=1; $i=100; $i++){
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, "http://localhost/test.php");
curl_setopt($ch, CURLOPT_HEADER, 0);
curl_exec($ch);
curl_close($ch);
$starttime = microtime(1);
$result = curl_exec($ch);
$trajanje = microtime(1) - $starttime;
echo $trajanje <br>;
}
?>
```

Слика 67. Време извршавања скрипте без заштите

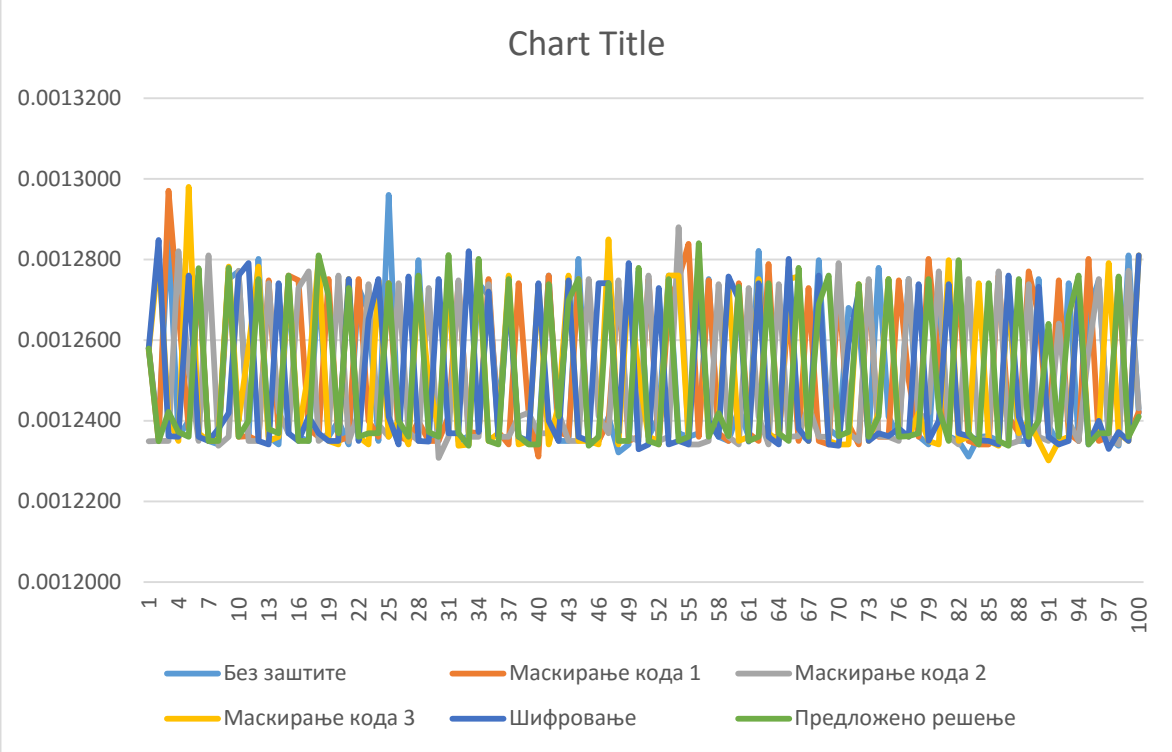
Слика 68. Време извршавања скрипте заштићене маскирањем кода решењем 1

Слика 69. Време извршавања скрипте заштићене маскирањем кода решењем 2

Слика 70. Време извршавања скрипте заштићене маскирањем кода решењем 3

Слика 71. Време извршавања скрипте заштићене шифровањем кода решењем *SourceGardian*

Слика 72. Време извршавања скрипте заштићене предложеним решењем



Слика 73. Паралелни приказ времена извршавања скрипте

Кроз додатно мерење и тестирање долази се до закључка да време извршавања скрипте заштићене доступним решењима и време извршавања скрипте заштићене предложеним решењем немају значајна одступања.

6 Закључак

Предмет истраживања овог рада чине могућности за заштиту програмских решења намењених за извршавање у серверским Веб окружењима. Веб окружење представља, по питању заштите, једно од најизазовнијих извршних окружења с обзиром на то да аутор решења често нема никакву везу, нити контролу над њим. Под заштитом се првенствено подразумевају:

- обезбеђивање да се програмско решење извршава само у свом изворном (неизмењеном), потпуном облику и
- обезбеђивање да се програмско решење извршава само у продукционим окружењима за која је аутор дао сагласност.

Посебан фокус рада чине могућности заштите решења заснованих на интерпретерским језицима. Као пример је коришћен PHP језик. Овакав избор је направљен из неколико разлога. Као прво, PHP је данас изузетно популаран језик и процењује се да је на њему заснована велика већина Веб сајтова. Самим тим, доприноси овог рада директно имају најширу примену гледано у области којом се рад бави. Затим, PHP је изворно интерпретерски језик, али су касније укључени и други концепти, као што су превођење, опкод, JIT и виртуална машина, тако да данас постоји више активних имплементација, углавном са јавно доступним изворним кодом.

Главну особеност овог рада чини сагледавање проблема и решења заштите софтверских решења у Веб окружењима кроз призму стандардног криптолошког модела. Овакав приступ се показао као посебно ефикасан у анализи могућности примене компонената са отвореним изворним кодом у креирању поузданог извршног Веб окружења, односно окружења у којем ће бити могуће извршавање софтверског решења заштићеног у претходно дефинисаном смислу. Применом овог модела проширен је основни контекст анализе, односно она је вршена на различитим нивоима превођења и извршавања изворног кода, укључујући реверзни инжењеринг као средство подразумевано доступно нападачу у покушају нарушавања заштите. С обзиром да у постојећим анализама нисмо наишли на стриктно придржавање овог модела у анализи решења из поменутог домена, овакав приступ представља његов посебан теоријски допринос.

Поред анализе теоријских модела и принципа, у овом раду је, како претходница развоју сопственог модела решења, извршена детаљна анализа постојећих приступа у овој области. Примарно, анализирана су три основна приступа заштити изворног кода - маскирање, шифровање и превођење у језике нижих нивоа. Додатно, кроз описани стандардни криптолошки модел анализирана су постојећа комерцијална решења, односно утврђени су нивои заштите који она нуде. Приликом анализе постојећих решења посебно су обрађени додатни проблеми који она проузрокују, пре свега зависност од произвођача решења, као и потреба за поновним третирањем изворног кода и његовом заменом у извршном окружењу приликом сваког значајнијег унапређивања интерпретера или виртуалне машине. Теоријска компонента доприноса овог дела рада огледа се у стандардизованој анализи постојећих решења, док његову практичну компоненту чине резултати поменуте анализе, односно јасно указивање на недостатке тих решења.

На основу утврђених проблема и недостатака, у раду је дат сопствени модел слободног решења, са подразумевано јавно доступним изворним кодом, чији је циљ да одговори, или испита могућности одговарања на следеће захтеве:

- слободно, отворено решење које елиминише зависност од треће стране од поверења;
- елиминисање додатних проблема која уводе постојећа затворена решења;
- постизање истог или вишег нивоа заштите од онога који нуде постојећа затворена решења;
- илустровање суштинских проблема у остваривању заштите на овом пољу, гледано са криптолошког аспекта, кроз описани стандардни криптолошки модел.

Увођењем захтева за отвореност предложеног решења уведена је потпуно нова димензија у решавању проблема, а остварени резултати, уз ослањање на претходно поменуте резултате вршених анализа, представљају централни научни и практични допринос овог рада.

Даље, у раду су анализиране кључне карактеристике хомоморфне криптографије и рачунарства од поверења. У случају превазилажења постојећих ограничења ових приступа, односно остваривања практичне употребљивости и могућности увођења у

Веб окружење, они су идентификовани као циљна решења за проблем који се обрађује у овом раду. Додатно, на основу суштинских и архитектурних захтева рачунарства од поверења дате су препоруке за развој претходно дефинисаног поузданог извршног Веб окружења. Ове анализе и препоруке представљају посебан допринос у погледу дефинисања даљих праваца истраживања, са циљем остваривања прихватљивог нивоа заштите, чак и код примене најригорознијих криптоаналитичких напада.

7 Литература

[1] “Usage of server-side programming languages for websites,” W3Techs, [Online]. Available: http://w3techs.com/technologies/overview/programming_language/all. [Accessed 28 January 2016].

[2] D. Kahn, The Codebreakers, The Story of Secret Writing, Editora New American Library, 1967.

[3] W. Diffie / E. H. Martin, „New directions in cryptography,“ *Information Theory*, pp. 644-654, 1976.

[4] T. ElGamal, „A public key cryptosystem and a signature scheme based on discrete logarithms,“ *Advances in cryptology*, pp. 10-18, 1985.

[5] R. L. Rivest, A. Shamir / L. Adleman, „A method for obtaining digital signatures and public-key cryptosystems,“ *Communications of the ACM*, т. 21, бр. 2, pp. 120-126, 1978.

[6] A. J. Menezes, P. C. Van Oorschot / S. A. Vanstone, Handbook of applied cryptography, CRC press, 1996.

[7] Federal Information Processing Standards Publication 186-4, “Digital Signature Standard (DSS),” July 2013. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf. [Accessed 12 May 2015].

[8] H. Delfs and H. Knebl, Introduction to cryptography: principles and applications, Springer Science & Business Media, 2007.

[9] R. Wobst, Cryptology Unlocked, John Wiley & Sons, 2007.

[10] M. Veinović / S. Adamović, Kriptologija 1, Beograd: Univerzitet Singidunm, 2013.

[11] "Wikipedia," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Horst_Feistel. [Accessed 3 Avgust 2015].

[12] W. Diffie / H. Martin E., „Special feature exhaustive cryptanalysis of the NBS data encryption standard,“ *Computer,* т. 6, pp. 74-84, 1977.

[13] B. Schneier, Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C, Wiley Computer Publishing, 1996.

[14] P. Kinnucan, „Data encryption gurus: Tuchman and Meyer,“ *CRYPTOLOGIA*, т. 2, бр. 4, pp. 371-381, 1978.

[15] National Institute of Standards and Technology, “Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher,” January 2012. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf. [Accessed May 2015].

[16] National Bureau of Standards, “FIPS PUB 46-3,” *Federal Information Processing Standards Publication,* 1999.

[17] National Institute of Standards and Technology, "ANNOUNCING DEVELOPMENT OF A FEDERAL INFORMATION PROCESSING STANDARD FOR ADVANCED ENCRYPTION STANDARD," 2 January 1997. [Online]. Available: http://csrc.nist.gov/archive/aes/pre-round1/aes_9701.txt. [Accessed 7 Decebar 2014].

[18] P. Bulman, “NIST Announces Encryption Standard Finalists,” 1999. [Online]. Available: http://csrc.nist.gov/archive/aes/round2/AESpressrelease-990809.pdf. [Accessed April 2015].

[19] National Institute of Standards and Technology, "ADVANCED ENCRYPTION STANDARD (AES)," *Federal Information Processing Standards Publication,* no. 197, 2001.

[20] J. Daemen / V. Rijmen, The design of Rijndael: AES-the advanced encryption standard, Springer-Verlag Berlin Heidelberg, 2002.

[21] H. Kruppa / A. S. S. Umar, „Differential and Linear Cryptanalysis in Evaluating AES,“ 30 November 1998. [На мжрежи]. Available: http://www.cs.cmu.edu/~hannes/diffLinAES.pdf. [Последњи приступ 15 Jun 2015].

[22] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *Advances in cryptology,* pp. 10-18, 1985.

[23] M. Robshaw, „Trapdoor One-Way Function,“ *Encyclopedia of Cryptography and Security,* pp. 625-626, 2005.

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

[25] T. Kleinjung, K. Aoki, F. Jens, L. Arjen K., T. Emmanuel, B. Joppe W. / G. Pierrick, „Factorization of a 768-bit RSA modulus,“ *Advances in Cryptology,* pp. 333-350, 2010.

[26] M. Grulović, Osnovi teorije grupa, Novi Sad: Institut za matematiku, 1997.

[27] X. Yi, R. Paulet and E. Bertino, Homomorphic Encryption and Applications, Springer, 2014.

[28] D. David Steven and R. M. Foote, Abstract algebra, Wiley, 2004.

[29] S. M. S. Goldwasser, “Probabilistic encryption,” *Journal of computer and system sciences*, vol. 28, no. 2, pp. 270-299, 1984.

[30] S. M. S. Goldwasser, “Probabilistic encryption & how to play mental poker keeping secret all partial information,” in *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, 1982.

[31] Z. Cao / L. Liu, „Improvement of one quantum encryption scheme,“ у *Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference*, Xiamen, 2010.

[32] G. Kipper, Investigator's guide to steganography, CRC press, 2003.

[33] N. F. Johnson / J. Sushil, „Exploring steganography: Seeing the unseen,“ *Computer*, т. 31, бр. 2, pp. 26-34, 1998.

[34] K. Schott, Schola steganographica, Nuremberg, 1680.

[35] R. Anderson and F. Petitcolas, “On the limits of steganography,” *Selected Areas in Communications, IEEE Journal*, vol. 16, no. 4, pp. 474-481, 1998.

[36] R. Anderson, “Stretching the limits of steganography,” *Information Hiding*, pp. 39-48, 1996.

[37] A. Kerckhoff, "La cryptographie militaire," *Journal des sciences militaires,* pp. 5-38, January 1883.

[38] A. Kerckhoff, "La cryptographie militaire’," *Journal des sciences militaires,* p. 161–191, Februar 1883.

[39] C. Swenson, Modern cryptanalysis: techniques for advanced code breaking, John Wiley & Sons, 2012.

[40] D. Salomon, Assemblers and loaders, Ellis Horwood, 1992.

[41] E. J. Chikofsky and J. H. Cross, “Reverse engineering and design recovery: A taxonomy,” *Software, IEEE,* vol. 7, no. 1, pp. 13 - 17, 1990.

[42] M. Sharif, A. Lanzi, J. Giffin / W. Lee, „Automatic reverse engineering of malware emulators,“ *Security and Privacy IEEE,* pp. 94-109, 2009.

[43] T. Berners-Lee, “The World Wide Web: A very short personal history,” 7 May 1998. [Online]. Available: https://www.w3.org/People/Berners-Lee/ShortHistory.html. [Accessed 2015].

[44] T. Berners-Lee, “World Wide Web,” CERN, 1991. [Online]. Available: http://info.cern.ch/hypertext/WWW/TheProject.html. [Accessed 12 January 2015].

[45] "Zend čuvar," Zend, 2012. [Online]. Available: http://www.zend.com/en/products/zend-guard. [Accessed 20 Maj 2015].

[46] "SourceGuardian," SourceGuardian, 2002. [Online]. Available: http://www.sourceguardian.com/protect_php_features.html. [Accessed Maj 2015].

[47] "ionCube PHP Encoder," ionCube, 2002. [Online]. Available: http://www.ioncube.com/php_encoder.php. [Accessed Јул 2015].

[48] "Decrypt," Decrypt, 2005. [Online]. Available: http://www.decry.pt/company/. [Accessed 5 Август 2015].

[49] "DeZender," DeZender, 2005. [Online]. Available: http://dezender.net/. [Accessed 3 Август 2015].

[50] C. Collberg / T. Clark, „Software watermarking: Models and dynamic embeddings," у *26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1999.

[51] C. Collberg, T. Clark / L. Douglas, „Breaking abstractions and unstructuring data structures," у *IEEE International Conference on Computer Languages*, 1998.

[52] B. Barak, G. Oded, I. Rusell, R. Steven, S. Amit, V. Salil / Y. Ke, „On the (im) possibility of obfuscating programs," *Advances in cryptology—CRYPTO 2001*, pp. 1-18, 2001.

[53] T. Ogiso, S. Yusuke, S. Masakazu / M. Atsuko, „Software obfuscation on a theoretical basis and its implementation," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, pp. 176-186, 2003.

[54] B. Lynn, P. Manoj / S. Amit, „Positive results and techniques for obfuscation," *Advances in Cryptology-EUROCRYPT*, pp. 20-39, 2004.

[55] W. Zhu, T. Clark / W. Fei-Yue, „Applications of homomorphic functions to software obfuscation," *Intelligence and Security Informatics*, pp. 152-153, 2006.

[56] C. S. Collberg / T. Clark, „Watermarking, tamper-proofing, and obfuscation-tools for software protection.," *Software Engineering, IEEE Transactions*, pp. 735-746, 2002.

[57] L. Douglas, „Protecting Java code via code obfuscation,“ *Crossroads,* pp. 21-23, 1998.

[58] V. Paul C, „Revisiting software protection,“ *Information Security,* pp. 1-13, 2003.

[59] B. Schwarz, D. Saumya / G. Andrews, „Disassembly of executable code revisited,“ y *Reverse engineering Ninth working conference IEEE,,* 2002.

[60] S. Schrittwieser, K. Stefan, M. Georg, K. Peter / W. Edgar, „AES-SEC: Improving software obfuscation through hardware-assistance,“ y *Availability, Reliability and Security,* 2014.

[61] P. England, J. D. DeTreville and B. W. Lampson, “Digital rights management operating system”. United States Patent 6,330,670, 11 December 2001.

[62] B. Lampson, A. Martin, M. Burrows and E. Wobber, “Authentication in distributed systems: Theory and practice,” *ACM SIGOPS Operating Systems Review,* vol. 25, no. 5, pp. 165-182, 1991.

[63] J. S. Erickson, “Fair use, DRM, and trusted computing,” *Communications of the ACM,* vol. 46, no. 4, pp. 34-39, 2003.

[64] H. Udell, C. Kappel, W. Ries, S. Baker and G. Sherman, “Self-destructing document and e-mail messaging system”. United States Patent 7,191,219, 13 March 2007.

[65] R. Sandhu, K. Ranganathan / X. Zhang, „Secure information sharing enabled by trusted computing and PEI models,“ y *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, 2006.

[66] A. Carroll, M. Juarez, J. Polk and T. Leininger, “Microsoft Download,” June 2002. [Online]. Available: http://download.microsoft.com/documents/australia/corporateaffairs/palladium_white_paper_public.pdf. [Accessed 5 January 2016].

[67] P. England, L. Butler, M. John, P. Marcus and W. Bryan, “A trusted open platform,” *Computer,* vol. 7, pp. 55-62, 2003.

[68] Microsoft Corporation , “Microsoft Corporation,” 2003. [Online]. Available: http://www.microsoft.com/resources/NGSCB/documents/NGSCBhardware.doc. [Accessed 2015].

[69] S. Ballmer, “Steve Ballmer: Microsoft Management Summit,” Microsoft, 20 April 2005. [Online]. Available: http://news.microsoft.com/2005/04/20/steve-ballmer-microsoft-management-summit/. [Accessed 18 January 2016].

[70] D. Brumley and D. Boneh, “Remote timing attacks are practical,” *Computer Networks*, vol. 48, no. 5, pp. 701-716, 2005.

[71] A. Jevremović, N. Ristić / M. Veinović, „Using cryptology models for protecting PHP source code,“ у *11TH INTERNATIONAL CONFERENCE OF NUMERICAL ANALYSIS AND APPLIED MATHEMATICS*, 2013.

[72] "Pecl," Pecl, 2015. [Online]. Available: https://pecl.php.net/. [Accessed 10 maj 2015].

[73] A. Gutmans, S. B. Stig / R. Derick, PHP 5 Power Programming, Prentice Hall PTR, 2004.

[74] "PHP CPP," Copernica Marketing Software, 2015. [Online]. Available: http://www.phpcpp.com/documentation. [Accessed 2015].

[75] "Kripto++ biblioteka," Cryptopp, 2015. [Online]. Available: http://www.cryptopp.com/. [Accessed 10 Maj 2015].

[76] „dl - Loads a PHP extension,“ PHP, [На мжрежи]. Available: http://php.net/manual/en/function.dl.php. [Последњи приступ 13 February 2016].

[77] National Institute of Standards and Technology, "Announcing the AES," 26 November 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. [Accessed 6 November 2014].

[78] National Institute of Standards and Technology, “Announcing the DATA ENCRYPTION STANDARD,” 25 October 1999. [Online]. Available: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf. [Accessed 12 May 2015].

[79] D. Kahn, The codebreakers, Weidenfeld and Nicolson, 1974.

[80] D. Göhler, “Next-Generation Secure Computing Base,” 2007. [Online]. Available: https://en.wikipedia.org/wiki/File:NGSCB.svg. [Accessed 15 January 2016].

8 Додатак А изворни кодови

8.1 Изворни код екстензије

```
#include <phpcpp.h>
#include <stdlib.h>
#include <iostream>
#include <string>
#include <aes>

Php::Value ppcp(Php::Parameters ¶ms)
{
input kod = params[0];
input par1 = $kljucLog;
output desif;
mcrypt td = mcrypt_module_open("rijndael-128", NULL, "cbc", NULL);
int blocksize = mcrypt_enc_get_block_size(par);
if( buffer_len % blocksize != 0 ){return 1;};
}
return sifrovano;
}
extern "C" {
PHPCPP_EXPORT void *get_module() {
static Php::Extension extension("ppcp", "1.0.9.4");
extension.add("ppcp", ppcp);
return extension;
}
}
```

8.2 *Изворни код библиотеке за AEC*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mcrypt.h>
#include <math.h>
#include <stdint.h>
#include <stdlib.h>

int encrypt(
    void* buffer,
    int buffer_len,
    char* IV,
    char* key,
    int key_len
){
  MCRYPT td = mcrypt_module_open("rijndael-128", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
  if( buffer_len % blocksize != 0 ){return 1;}

  mcrypt_generic_init(td, key, key_len, IV);
  mcrypt_generic(td, buffer, buffer_len);
  mcrypt_generic_deinit (td);
  mcrypt_module_close(td);

  return 0;
}

int decrypt(
    void* buffer,
    int buffer_len,
    char* IV,
    char* key,
    int key_len
){
  MCRYPT td = mcrypt_module_open("rijndael-128", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
  if( buffer_len % blocksize != 0 ){return 1;}

  mcrypt_generic_init(td, key, key_len, IV);
  mdecrypt_generic(td, buffer, buffer_len);
  mcrypt_generic_deinit (td);
  mcrypt_module_close(td);

  return 0;
}
```

8.3 *Изворни код шифрован са SourceGuardian*

```
<?php @"SourceGuardian"; //v10.1.6 evaluation if (!function_exists('sg_load')) { $__v = phpversion(); $__x = explode('.', $__v); $__v2 = $__x[0] . '.' . (int)$__x[1]; $__u = strtolower(substr(php_uname(), 0, 3)); $__ts = (@constant('PHP_ZTS') ||
```

```
@constant('ZEND_THREAD_SAFE') ? 'ts' : ''); $__f = $__f0 = 'ixed.' . $__v2 . $__ts . '.' . $__u; $__ff = $__ff0 = 'ixed.' . $__v2 . '.' . (int)$__x[2] . $__ts . '.' . $__u; $__ed = @ini_get('extension_dir'); $__e = $__e0 = @realpath($__ed); $__dl = function_exists('dl') && function_exists('file_exists') && @ini_get('enable_dl') && !@ini_get('safe_mode'); if ($__dl && $__e && version_compare($__v, '5.2.5', '<') && function_exists('getcwd') && function_exists('dirname')) { $__d = $__d0 = getcwd(); if (@$__d[1] == ':') { $__d = str_replace('\','/',substr($__d,2));$__e=str_replace('\','/',substr($__e,2));}$__e.=($__h=str_repeat('/..',substr_count($__e,'/')));$__f='/ixed/'.$__f0;$__ff='/ixed/'.$__ff0;while(!file_exists($__e.$__d.$__ff) && !file_exists($__e.$__d.$__f) && strlen($__d)>1){$__d=dirname($__d);}if(file_exists($__e.$__d.$__ff)) dl($__h.$__d.$__ff); else if(file_exists($__e.$__d.$__f)) dl($__h.$__d.$__f);}if(!function_exists('sg_load') && $__dl && $__e0){if(file_exists($__e0.'/'.$__ff0)) dl($__ff0); else if(file_exists($__e0.'/'.$__f0)) dl($__f0);}if(!function_exists('sg_load')){$__ixedurl='http://www.sourceguardian.com/loaders/download.php?php_v='.urlencode($__v).'&php_ts='.($__ts?'1':'0').'&php_is='.@constant('PHP_INT_SIZE').'&os_s='.urlencode(php_uname('s')).'&os_r='.urlencode(php_uname('r')).'&os_m='.urlencode(php_uname('m'));$__sapi=php_sapi_name();if(!$__e0) $__e0=$__ed;if(function_exists('php_ini_loaded_file')) $__ini=php_ini_loaded_file(); else $__ini='php.ini';if((substr($__sapi,0,3)=='cgi')||($__sapi=='cli')||($__sapi=='embed')){$__msg=" PHPscript'".__FILE__."'isprotected bySourceGuardian and requiresaSourceGuardianloader'".$__f0."'tobeinstalled . 1) Downloadtherequiredloader'".$__f0."'fromtheSourceGuardiansite: ".$__ixedurl."2) Installtheloaderto";if(isset($__d0)){$__msg.= $__d0 . DIRECTORY_SEPARATOR . 'ixed'; }else{$__msg.= $__e0; if (!$__dl) {$__msg.= " 3) Edit " . $__ini . " and add 'extension=" . $__f0 . "' directive"; }}$__msg.="";}else{$__msg = "<html><body>PHP script '" . __FILE__ . "' is protected by <a href=\"http://www.sourceguardian.com/\">SourceGuardian</a> and requires a SourceGuardian loader '" . $__f0 . "' to be installed.<br><br>1) <a href=\"" . $__ixedurl . "\" target=\"_blank\">Click here</a> to download the required '" . $__f0 . "' loader from the SourceGuardian site<br>2) Install the loader to '; if (isset($__d0)) {$__msg.= $__d0 .
```

DIRECTORY_SEPARATOR . 'ixed'; } **else** {$__msg.= $__e0; **if** (!$__dl) {$__msg.= "
3) Edit " . $__ini . " and add 'extension=" . $__f0 . "' directive
4) Restart the web server"; }}$msg.= "</body></html>"; }**die**($__msg);**exit**();}}**return** sg_load('7295141D8980E6FBAAQAAAAWAAAABHAAAACABAAAAAAAAAD/N9AnpQa/VyWFyv5zt9pZZVklJBOGw3RcGpZLuK6nuvOTCXZdsuul487erNYwwQLxgyuPVbnbbVWrX6XQKHV+yk6DGZqHnnSWmI+m1w+xga7jJwJzUutbfWuMWt2USozsYXi8SwyGQJG16ibY4+uoyTQAAAAYAwAAl/+n0PD3AysYSNvUrMihrXGczkRWUd8HPUcg7Yzfu68PMxWekUpnuFC/N38ZLCNSJb4dIpn0BIzpMPfbwFhJkm3rzyZ5yv6nJhVSSsC/6cUxf4EuhM0L3+QAttJhHDpqZIhdC+TCG5yjVH7rlbfhkZqKC7uMlgOOoeZLv2yfnaUfg1lLlK+Pvb0kGxPzFihhjwq313q7dVgtg/4FUEXTM93dHAHrectgNdyuOtVwJlDFZRym7Ywy7jVDQP7rI2HIN90m4VENWdIhkjmBc3IPHo3FPBe+fVmn/RaAGm1xl0ae4BwwkV0CDuVCObKMNj8H8c1dPQJO5vUOM/d/y6KO+CCrVp4szZJj1mG7f0RP/p3qhnaVlFWo77amBuFLWfnsQu9e8iFKba0N2Bt1yKcQVtvfKNLxngePwQj4jRkWP+Yu9rnzd+4LMXfopsiX4U5V95TONjNNx187ksvvLT5bOtfQrbYJYkbklOVMABh5ksgoLWqjiy8rxf5nz3XNusaVnr/jFQpnPfa52AwLvTNLWpVm+l2DbEeyh5yYyITOALeBWlZbwRZ5G6ZIiewzDEwMh9R6EV+1AVtXvu+CCWJK2eOv8fifCeeTQ7YmL0niOasbYYoTGWHGr/wGZF8RFEOngK8+xVoWq+PV5R5Vjbcbp84/TTktCgwQeGQMP/WcxQit5hLg4iIEggGkPEX+WJjYRnxhTFALTpngRSVBQ1QKIowKPZMyunnmNoVgGujpkCiE5Q4cWt8Y1VzZR253YjA5qwsmsoqvX2RREIhU/QDu9Gcd+b/iEMBdGrOcAE3ytA4AiiiCJmAXMQZuXIIhtCxALZlF+u5AbPOEwHlsRWISFPQ6t2zJaM0G5kdyTLvk8gFuY7bCrHycEwCyf51mNFTd7L/cC5NsczBeT1DAdJvQbXKvtdrL+/aO3mrtMcI22v92rYQ4LMX9HrHjE2D6GiOuSCG+1l/bCKH8IcAEhq34ELiUv3XERquwXssUU7zF3N1ygS1beANCyfr+GdwWdsigVdB/wbgdW2TGO4KCkJA2gobeV9laufCP5NQAAACADAAA5KR8lNpVckzLpjapprdiZSKtB2fuc7unFxl67nk8BIVeWNJPbdEM6+ehssV4GHRIchmm1op/mF0lgPlQz49Rh1e+zrw1+EdeeJUo1AirjMPzwZxBtjlLE3JN+XOTHola+B2pOVOFWVkOOo7D3Mrvu19TU8Bnqf8JNfRnlCwLb/QxOEMpELGI695laSUwJE3fF4/xc4+GzqUzeeeQD3SDdps/z70zXRb8doqJBkFN88U9BtBoHlEoLQUfoS9HFtyXQnnf5PbIpSHWmegNkOcom7e45ta2/2KgRiqwGS1WHslUqRzFhlY2cvzdZVvLMoPsRrVXW0s4w2/T5vudwH0HNGDVyhUZ2mRHDhiVvfzQxdmR0EnCtQnJIe/ti6nrfeZGQPJqOeqvo69v3i2TVMwp/rfQla0m6iSeCoeqza+WXU/7Ibkp/mE8Omc5ElnhhxeLnr57tG9VfhDHpBrb2Ntevx21P/zwFnTHAcLipKUMXNp5QoF6nz0g6yKlKpXGi1OZBUEhgYEbS70wB8I6fFrsYd7gJJpf2k26HWITtYRnnfLWbUAnvb4TbnK/SNzHfEgt8QmUkSfStTW6se4OyrHMv2UhFRpTJkBMTSQSBr68G

HDY8LLiQEUfEybcWT3dJcXP6Uuxod22sMVL7tGJBeiZx8+TFv6IPXFd98fPvta5lsPiZ
LbeiwIW+I0+Te8GATCN+MibfteOMN0Aq8Ejc74oC+32qKeqqd4aqdeXWXgh3KKWqr
A2io54TauF9dQJHGshd9c5VFcck2ULT65W7HevjQwtBG9Wli3OjCjoWMyTw/ALhk6H
cLc81COFbi03H2NpgdAokWOPaFrCBIC0/9U0ZwJ99tmGOKkKf9i4gETKE98XiW6w
WpHUkORt9uUjElScSkMwBFqosrcAsu3NlkiNbvVTuMs27fHa8gVP05UgpV+9r8UATS
tPnBHWshiXfIOYtyiaQvCZPxPf9XoVs+FCcfvcsn5N3nIQFXKwAp1vi3uYrigCOWypg
H2Mdkze+Ksi9u8HdgJlJCGCAdrYfdYTLy7MeMgRNNc/nK+lATvAnrzYAAAAQAwA
AEHJwVlY7a1CJKj6/7JxB+GtGqboDDBNBgOdbRQk3Y0SE9u4hp5v1aoHQw+TkrE5Y
BctYwWgq2EKRnZlCgiSyiY+opuwEBa7TfdPlxS6f9DojtsCwCktf855L2OyS+fbE46DZa
oHeGsN7DWKnFTGLikUQ/3TAPdsFsyFPimeHTafHHhGHz+QTIB6sVACiYozwi5yrOS
ivrCxVoS9vDKF0tnc9Bp6WoUn+W2UXf6kNRt+7GJfGZzQhXNJ0MdREVpkSF2KOff
Hiy7zmTjX8s94Kakl7lc/M7I+wW9F5U/jNCo7qAGrkS7iKvah7h2ZuOFOBMebo/MiWtd
/mEbxvFVVJf8drjFPz73fI0Mujs3smlIDzEK+R44FSMgQiKmWJaIJOooT+/bxPkpOMpA
GbIa7tVMF3t+ooIlzsXiBJ1dAU+MgjJ3S/Cb7CvcjltxnvmYEMH6/i0DSRziWfsqSwj70W
vtbV9O16aqWCXVpCGQfwK+1dUZ7bYBRrpiCgAgajNoA8Vv1WMNEtYzgs9yOArHa
I9GZadBZLzbza3ONFls2one4Y98XME20t/9o+39GYOkOhVmzredqPGgy9RRIKAm3qT
AvfKRtZqz/4cg65R5J15zsbj14zYQwcrog7svmUpATVHJ5ybJcpbO98/YniibB5dkgOMM
BUw+AclodRT133hqlFlzV4sEZGTHbvtL37cRLG/YRlUw2NnkEnzUxQTBQINnrMmym
YiDZEQjOz2SWw5altNJpMcSzGsd+DXFv6ComHRXuxUsN+qtjNoMJT5wXEIqGOnJO
vXcfelbYejl109RI6cbU4A9vzHqXT5OGcKWyELJD1z3kya8vqHxHHMXlnJFpA4uH0F7
Kw9Tfq01e1Ggxzsk1BVBf6fnqoxeGvMdDmJaG9i4PDst1bdHb36Xt5hK7XGHS994dC1
q6y/IyfKH8UckDy9np+6q3hbivo7J+vyK5U+aeoTV05tVGFj8DMOP9pQUu9nDvRIxf84
gFujLcDkq4MhN0EQQazhkPwXADtSxhBhYyI9zRwHvTFHrLZbDcAAAAIAwAAF38f
k41cCBeHkzDOmZdISy4/JMxDJYdij1Zy5PkfAH+ooeUSAHsBzaQ94G4PfZro1ISM72d
aBQlu0sUQ6Le7EzKDwprHuYHWT/+ryQEU+CH6pCI7mdYjNewqKZ0v084s+2LGbLK
BxnZWsPrLOz4M4AWH4O05qf0IlbBXBV6nREFe4Xvzk8gYe0jt/kplAH/PAYLGE6Hq
FQTV+oi+hOJlWAP2vt1zezm3WgDpC4aBBy0NV1KpWp/gXS5dWiKaMCe9b7MRDL9
POkGQjOkKiec053vCMYhf7KsYcl/ED0KCmDALCmsjKF+4mzVvksrGcXR00WOqihH
3TYD8HyVuOLy3Qc6BJg+2XDzFBhvPvxNhGMlhWRH1EF/J63pixkbT/iTMfxjK7+8oK
IdWae1IW91XAhKqeFYcRAD4swANioGqN4+2wyvsjokAlY4a54rVsI7J0gWMAFGo/m
gnW7n6egZ742WRjlM9nUBG+Fb6lYyZB44tWmn88mciE0F8EgF3G5FzEv1DhLSOYM
yiqRDxBvZrN09QcFjQBWeupE+KaGlhMyRE98YP/c7SfmKbgnAM+RFbQWeYCZeeD

```
xpnHv0HVuLd+4SlqRegw28Ph+lvtKUSPH6XVFdFElrOVU0b+2LXj9FwDBr4Gdd4cLj/
8vb8G4Nj6iQoOczmf/zGIHy/eSNGOqlFKZSyLKkkEgRYd8C069QI3bfnf/iH+MVs/m1m
HtclpyxESxtZ4Y2PtC8dX3ZfcwafKWUGfD0lLofMAPHocC9VJQclAXEDqVQAvkN1F
W9aWxm4poXW+jmVPp+oNsyXHnVbeLeoaXObJOUkrys/jmUhrSiGg7kR8hkkjiNPalc
VSAm3cXwHCuaB0kZp+nPbzg3BKhyAhAEl7YVpM6+m5M15Ukh1sCDKeU8ImpKEs
J/1naGmP5xaZCWw+Ur4nyAKqlmhahqAtHXVJXJLPcpO+5vn1EduYJxUqH57tVl6Q8S
royBCpCeD1l6KB+vhTEVOhLf3Ufe4OhhNp7uO51hyKHHc/LHXzDLhfNw4AAAACA
MAAE+rVI0tnRG//XfswcfkU3gYaRxBrgA3iaMAjFZji1xZogY8/UyMNHdLrnHAdr6Hv
OgrL2aJltiFEpUwPwtYxUPNBOqcVZAEeBzbGCbH8kjd9xBqRdDTmV67+FFJlZ2pc2V
EKKYgKK+rr9V2yi0LbbjNK5zhCVBJglaqZOGkB9yMloM6j9XSPLn5dmrhv2rj4QpOPj
VsrWYgmI20ZHLpZ1ilWKV1W6vcWfTz5mtGODKr0FOxQ3t0QGxTncaVhImfwR5+/o
rOYA645R766yGpsf0VwfMiADCQ1ZsjA8a2l0uC4cPER+buGRy4ydvEzlKLfMnijjvY/7
ZPKeVEBsNbn1aUOHyMNXG/D4PTSsP5RL2vkR1kpTNOTzqVMVhrBI2d+eiUdu8Lm
as7vJMkvjM1q6hffaJgW/QSSgcfMF+rE8giWISW3vUOBJK7gSF9DJBP3vduhWOMDftz
RarRETd/SO1Zm1I335SJ/9gEGE+BrBjrn98D25OgYGedrasHtUaEljoSa02v0mjetKevjni
wX2mxI0dPuZJXztcD5V+ejRoRVhjMrGZOD4GJuL7h1kYPRK6rrEHW4zxcegb2P+Q+3
7iKYCFtAUHz/kccDvcwF5Wj9XRqZJ3QMd5uIM0bvRYkZ+tEhTdHn/hmPEYurs2wDZ
pNQwlBppr79THRbdAx2Zq/RTE8dIngqmIQSDczD8ZtgNMKsh/b36BZ8yvgu3ruDJ8110
MS5VqF1obzD2C4n7WBAJ1T6x7fESbDPXO9lDV3ns1ia4JvB3f2q46NOrWPO7EBkhpI
s0xvbuY1t2eeh5NtyT8KlZHKHGHIZ36CMiVYAih8qIFzeiomWzcx3yBZjfP9IKV8xqK
QFzAuez9wjxanM68JH5aivBf8h7q1jAIYnVk5Pmuwr+WCVM+1LtaTkO6SDv9Im2tTxJl
t1CE0Y6afiyoFDFG7BypUBU58In+kHuCFxFrncLJVHU/HO8Vj+sfuHlW9bJcV0oOuFI
w7sX2f7e6UieShgnY1Fzaue5s3DgD5uKklKiP2AAAAAA==''); ?>
```

Изворни кор маскиран решењем 1

```
<?php
$g6a15a39="\x62\x61\163\x65\66\64\x5f\x64\145\x63\157\x64\145";@eval($g6a15a39(
"Ly9OS05Pdy81TFF6bUEvaG1icUF0MTh3MlFMUEZLYWkrMUZSeWlQeVBBdWZV
Sjd0R1gxWGpvL3R2R0plbzBoZ1c1ZFUzUXdBNDJCNVN0bzRjT0d1ODZIQWhrelhPR
jZLR0orYTRCK1RRT1Q1Z053V1BxN0JTUlVET1FxdElHczZuR08yeTNhaGtSTlY3VW
YrMFJmY3hUcVAzaS9TM1BadnJNbXpPVzBFYXBqL3F4UW4zY3VuZlk1YmpKaUZu
NXh5UTRMN3AvcDdDYXYycTloTWRvb09rWUxGR1Z3V3h2VGNCOTZOQkc3S25U
T3RjQ0s3WElDclVXbmVudEowWWtqZWN0bEVBeDlCVC9FRlEvdDd0ZnVWOWgrQ
UNMYlE1TnBqY2p3Mm02RDFZMUJLS3RlYlFzbEhCL0dMZG1OWDZEUnlGeTMwK
```

0lVR3VmajFuMkU5bzc3NE9VaFczOW03S21xbWtvMHdSSHBTVWIrRFFiUW9iTU1U
Qkt0VlVNUlByb05xK2R1dFVVK0x6NGI0Y1VSZWtvZXlYc3Znb3R2L05zaXNQNGR
vdVlMK0VsQnFkdVFiQSt6UHVwWGVzQkEwRERBZlVIUTBIeTBKYmcveEZKaUJV
UlQ0M0ozSitXVmRpUElNSkhWSWpVQ3Z5T3BvNytTMExhR2o9PTozcXAwNXMwN
AokbGMyZGEwN2E9IlwxNjIiOyRhMjg3ZmI1ZT0iXDE0NSI7JGY4YmU5OGZkPSJce
DcwIjskaTYwYzhjY2M9IlwxNDYiOyRsMDAxZWM3Yj0iXDE2MyI7JGc2YTE1YTM5
PSJceDYyIjskYjU5Yzg3Y2M9Ilx4NzMiOyRvY2Q4MWEwMD0iXDE0NyI7JHk2ZDM2
MjgzPSJceDczIjskbGMyZGEwN2EuPSJcMTQ1IjskbDAwMWVjN2IuPSJcMTY0IjskZjh
iZTk4ZmQuPSJcMTYyIjskaTYwYzhjY2MuPSJcMTUxIjskYjU5Yzg3Y2MuPSJcMTY0I
jskYTI4N2ZiNWUuPSJceDc4Ijskb2NkODFhMDAuPSJceDdhIjskZzZhMTVhMzkuPSJc
MTQxIjskeTZkMzYyODMuPSJcMTUwIjskeTZkMzYyODMuPSJcMTQxIjskYjU5Yzg3
Y2MuPSJcMTYyIjskbDAwMWVjN2IuPSJceDcyIjskZzZhMTVhMzkuPSJceDczIjskaTY
wYzhjY2MuPSJceDZjIjskbGMyZGEwN2EuPSJcMTYzIjskb2NkODFhMDAuPSJcMTU
xIjskYTI4N2ZiNWUuPSJcMTYwIjskZjhiZTk4ZmQuPSJceDY1IjskYTI4N2ZiNWUuPSJ
cMTU0IjskYjU5Yzg3Y2MuPSJcMTQzIjskb2NkODFhMDAuPSJceDZlIjskbGMyZGEw
N2EuPSJceDY1IjskZzZhMTVhMzkuPSJceDY1IjskaTYwYzhjY2MuPSJcMTQ1IjskbDA
wMWVjN2IuPSJceDVmIjskZjhiZTk4ZmQuPSJcMTQ3IjskeTZkMzYyODMuPSJceDMx
IjskbDAwMWVjN2IuPSJcMTYyIjskZjhiZTk4ZmQuPSJcMTM3IjskbGMyZGEwN2EuP
SJceDc0IjskYjU5Yzg3Y2MuPSJcMTU1IjskYTI4N2ZiNWUuPSJcMTU3IjskZzZhMTVh
MzkuPSJceDM2IjskaTYwYzhjY2MuPSJceDVmIjskb2NkODFhMDAuPSJcMTQ2IjskYj
U5Yzg3Y2MuPSJcMTYwIjskb2NkODFhMDAuPSJceDZjIjskYTI4N2ZiNWUuPSJceDY
0IjskbDAwMWVjN2IuPSJcMTU3IjskaTYwYzhjY2MuPSJceDY3IjskZzZhMTVhMzkuP
SJcNjQiOyRmOGJlOThmZC49Ilx4NzIiOyRvY2Q4MWEwMC49Ilx4NjEiOyRpNjBjOG
NjYy49Ilx4NjUiOyRhMjg3ZmI1ZS49Ilx4NjUiOyRmOGJlOThmZC49IlwxNDUiOyRsM
DAxZWM3Yi49Ilx4NzQiOyRnNmExNWEzOS49Ilx4NWYiOyRmOGJlOThmZC49Ilx4
NzAiOyRvY2Q4MWEwMC49IlwxNjQiOyRnNmExNWEzOS49Ilx4NjQiOyRpNjBjOG
NjYy49IlwxNjQiOyRsMDAxZWM3Yi49Ilw2MSI7JGc2YTE1YTM5Lj0iXHg2NSI7JG9j
ZDgxYTAwLj0iXDE0NSI7JGwwMDFlYzdiLj0iXDYzIjskZjhiZTk4ZmQuPSJceDZjIjska
TYwYzhjY2MuPSJcMTM3IjskZjhiZTk4ZmQuPSJceDYxIjskaTYwYzhjY2MuPSJcMTQ
zIjskZzZhMTVhMzkuPSJcMTQzIjskZjhiZTk4ZmQuPSJceDYzIjskZzZhMTVhMzkuPSJ
cMTU3IjskaTYwYzhjY2MuPSJcMTU3IjskZzZhMTVhMzkuPSJcMTQ0IjskZjhiZTk4Zm
QuPSJceDY1IjskaTYwYzhjY2MuPSJcMTU2IjskaTYwYzhjY2MuPSJcMTY0IjskZzZhM
TVhMzkuPSJceDY1IjskaTYwYzhjY2MuPSJceDY1IjskaTYwYzhjY2MuPSJcMTU2Ijska
TYwYzhjY2MuPSJcMTY0IjskaTYwYzhjY2MuPSJceDczIjskdmJjZjNmODg9JGEyODd
mYjVlKCJceDI4IixfX0ZJTEVfXyk7QGV2YWwoJGI1OWM4N2NjKCR5NmQzNjI4My
gkZjhiZTk4ZmQoIlw1N1x4NWNceDI4XDEzNFx4MjJceDJlXDUyXDEzNFx4MjJceDVj
XDUxXHgyZiIsIlx4MjhceDIyXDQyXDUxIiwkZjhiZTk4ZmQoIlw1N1x4ZFx4N2NceGF
ceDJmIiwiIiwkaTYwYzhjY2MoJGxjMmRhMDdhKCR2YmNmM2Y4OCkpKSkpLCJce
DYxXHgzN1w2NVx4NjZceDM0XHgzM1x4MzhceDM3XHg2NVwxNDZceDMxXDYz
XDE0MVx4NjNcMTQ2XHgzNlw2NFx4MzFceDM1XDcxXHgzNFwxNDJcMTQ1XHgz
MVw2MFx4NjNceDM1XHgzOFwxNDVceDMxXHgzNlx4NjZcNzBcMTQ1XHg2Nlx4
NjZceDMxXHg2M1x4MzRcNjYiKT8kb2NkODFhMDAoJGc2YTE1YTM5KCRsMDAx
ZWM3YigiQ0lLS2VkRVhSaWxLYjN6TFNEOTRjOVNYdjZydWJLVEEwM2FPejZMa3
VKMisvd1ZtMnlnRmR1RU15RVRjSHpveXc0Y1hIUVlPMnM5K3NFOEhxdWE2c3FR
NGdNQ3NPMDVxRWFsd0JVMXU0ZCtDamU5RTh2OVRQc1llMTQ4dWxreDBETVB
ZTlBLYm81R1RldGliTXFGU1lsQS9LN21qSzRWZWJSWkRnWndiOVlzdmNzRzdOW
TJILzZ3OFBuby8rWDVSWENtZTEvK1hZcnkrL2RpbDg4cy9SLzc1WTlVQ1krVTJzT0
wxNW1nQWM2OWlUeXpKSktWQ1NLUi9YNkpEM3VuanlWZnNCNVc4Vk90SFpuM

HY1dVgrNGw1cU42QU9IZ1NVNVN2cUxEd25QRUFYOG5KbUU2ZzB6WEFNelV3M
lU3US8yeHl2QXRKdUhCcjlKSnI0R3h2YjREaFN3enphU1NkOVVENEJQb3BSSnF5bV
dtc1RTa1BMWnpQWVdxZHFTN0Ftb2EyTW9raHZKRkVhMU9BNHYyaHVCTUExcCt
uN2w4V2prd0taM0JOUE9RQWpiK0FxT1pDeUtZTml4UkFBWC9ETlkzV1NBVzhsTm
QyQlFWRmJTZ2ZsYzVMdHNpS1QyMTFycFhGczRmMDhVbllXYmpEZ3R6RnQreEd
nYXUvbW90NnFLSldTZTFYQWIzRHI4RlE1Y0hjd1dIWWlPbjZUVEd0ZHhHQUpmR0
9QSHU5QTQxbDMwYXR1SjB5aWhabENEMmlMSlQrTVllY1pXV2FzYTBtQmFhNX
ZNOWNKWjRMdTg1dlR3ZE8yMXdWcDArdlczMUtva09EaTl3T2wzUFd6Yk1MT1pD
U29wOGxUdjlFYkZobkNXbTJmbldOWk9nc0hNajBpU1FwcXJjT1hEanA0ZGZsUXJ3T
01lWWhXKzhlS1lRMWRNLzhEVEk3KzlOUlR3VFRNSXc0d28ydzRtVW1DQVhJMjEz
T25kd0Y5RW9Od0RCM0t3dVArYnZwbEZEcTIyZmo1N3hQbm9sb0N1cFFybzVDTmp
2bTFLY3RHbyt0TkkrajJhYXFsenl5aU1LUCs4R0xremlmeHNiZkVMU0FiQ25WUEJIU
DdpD1A1ODFKR3AyN3BubjhJbVJZTjBUbFlFaVlUZ2pwVlgwc2JyYkNtY3BTemJRcG
R1L3Y5eUd2UE9LMEJ2SFB5bUY4MldhdGo3dXZkeDNxZThxc2pCUngwdnA5Y28vW
GRka1NsaDQrV1ArUTNDWFhRdUFwRTlvN2doQ3N4ZitUQjN5N01UdVRKNHVJUz
NEMTZZR2tHckRPZHJra3pSUmJTR3Y2MGY5MUJQR3luQzNnQzhoUU94bG9BVW5
idXhObWtkZWFrQTBqSUM2RlVjdmVKbjJLZnZPeC9jL05iNXFMQk03bit5VHVlU0xL
TmNnMk5xSzZuRDNsZjd6N1Z1N0N1RzF5V294SGdoYzFJc05WUTdMUnFOWmR0a
U1KbmNheGFDYUtaYUkxaVY1dElpcjk1SHBCUWxZS1l1MDVHdmhEUWpyNUhNb0
5BRDNxbjJsU2pHS1FjNDBDV1BKUWt6L1pKV2pSSGhHR0hoZmw2SmxGdWpmaW
RCNjAyWENkMmdJUEpINUdadG11WjcyY093N0s4UnR5M2hickhKekRLdjc2L0YwO
TJwWDVycjlLaEYyTTRlWGExV2FqTm1uL3JZV1ltY3VPbzlUa1BUTGNmMVV2OHF
OZnhEODl0RHFMZldEb0hGSm5NQk80L2ZXdlRlZTY1K28zbXhhVHhYZzlhVnhBckt
SUzg3MWJzU0taTlpKdjFRZlk2Z3ladmh3NlI4dFVsWGhPTDYxdlhnRWp0c2oxdTBiSF
h2eXpFK0p6aktXN1hlVGVibHB1bThxbmI0aWFwOHlVdGtyVTlIQ2J3cGo4RDJQMG5
WaEswTEYwckNSbWdMUGcveUlRU3VETUd4bThBcDJEM1NyOG1RWHJUcTVZM0
5SaEJ3QmpDc3d1eXdlUTZHSFpaa2UvMWhFTGk5Q0xQRkttUXE4NmNsYmx3Nk9L
MUN5QitoRmVidkhQNkkxYml6eVV2dzVsbXRLTnFnL1A4a21kRDRJSFFlNjRjdklzZk
Vabk1sMXJtdlJqUXJ1VXRjKy92c0kvYlBrUXY1WXY2UEZ0dnZEV2hBK2xMZWFFe
XE3c1g2dlFTZTVNREJ6LzlYL09oRzZHUnNMUzF4L0ZkQkE1OUYrdk5Jc3hPanpGcU
lEazlYZDRrSHhoVXVvUkJnczNVUzlPTHhkTFA0cjk5VVl0cTYzemNlVmZMdXAzbD
dIdDVWN2pBTHF6WWVXcWI5QUFONEZNU1FFQVFYbzJBSVhJQlB6ZHQ4RkdXe
UtPM1JBMDgxOGI2WGpYdVp0dHREbm5GY094czFtd25uU3BHZS9ZVUdSRURaUG
R0QzZJelNjRmxyYjl3SjE1MVFGTWlhRFZDdHZuVEpPSVlNaDU5ZzJKYXF3cXpPd
WNQYW92S3QzNUhJNFM4bVhpeWlBNkpjRWROMFdTMy9mTkFMYndoWVBNV3
UwbFNMSlJ0aVNsZWZLeEJVUTNsSUh4U0ZnTk5JWGM2R0V6Y3F4dWN4ek5QUm
41NWM4YVhPbU05OXhKaDVRUVAzMHhFNExlYjJXbnVVRVFvSGp2WW9NSTh2
OUVrdWo3Qld2UTdRYXkwRCtZS3FaUFJjWEdhODRsa3lHQWR0aGJnOE5GTmJESD
gvUEUrbzNuQ0VrWHhYRjNFeGgwS0pjTGV4N1FnSXZJd0VxS0RiOEgxSEhrckQ3Uz
ZqMDJlcy8zYUplLytORD09IikpKTokb2NkODFhMDAoJGc2YTE1YTM5KCRsMDAxZ
WM3YigiQ0g3VXdoQUxSaWxLRHU5NmJaWndFTC9UTnhDaUVMY1RBWHRZaUZx
U283NStJTGhxRkZOREZPWkUrbmlOVmt2WVJCYi9LOThVc2k4Ti9iTWs1WkNMOT
RVdFU2UXNPNU0vcmlFL1prdVNpelRaK1FRNk5zbzE1OXJEY1BEcERxUVVOUkwv
UCtZK0JMTi9rOXhVVWpaOCtvL2NDNFZKdGhOZlZyWHNFQlhHRVVqSkM4enN5T
y8rcHMrTXNHRFV0YW05K0dpb2JpbzNpbVQvcy8zbThyOS9hSzUvZm9YZU1sYWcy
VGxlY2Z2eW9MOU03VWRaRzdsSVBlaVhIcXYzVUdlNjNRMmoycjA4VWxiODNnS
HVLTzZJTnZEQm9qekNvSnJ3UXorZHVHRjcrdmFnK3FEbFV5U1FyMHM2dFk1NUlT
cjVwNHRuQjRYbkY3cnZMWWJTVExsRjVHRW5QMXI0YnpvRjdPVmF3Tzd2Y2pG

```
MXRhcFlqU0p1cStMdWpJaTh0aFlXTmpMZzZZalVKbkJ0NzdOUTN2MDdJbWVpaTlT
a0ltWlBQUzZzTllQUHd2RVVETlpQZGx2SlFGUUhONlZaME4xR3IwYytVOENCK1Jy
czNDUndWdTZ0R3UzWG5zWFNSd3N3eUhEaHNlZkMxeW5QcHQ5c1VjMU8zYzRxL
25tSitjTGN1cVV1aE0xSXc5VG1PdTQwTVdTaGZjS0tHZVA2Zmc2WGd2K3dSZDVW
NEROUmg2OTU0MzIyS0s1blpmLzF6YVZVeTNpcUUzaWpjeHBaenNyODhXSlBUQU
tMNzhCbzJZTHlNNUhIc3FKUThFVnQvSEpGb3Y5NDdqZFNSdHZ6NEsyQVlRWGRC
Q0VwNElvd3B0eEYyNERkbFM2SW5BWGdjL29BNU9jMWZLMEdId3NrMkZBQ09n
YXRTUDAxZ2FZT2hGdHBmTitqYmVQSkR1YXdSUEN1VmZMaENpZUlBZGFJOGI
0MDhpekZCNXFlMWNXUWg2RmdtQ1JCeXFIdGNpMEh2dTM2ZVNRdURIR0Zqd0V
DcG1lMUZiOVpxY1o1Q0xnZUVlOWdvOXZUaVh4anZvb2ZFcU4yUVl4dWNCa3NM
VnA5ZTREbWJ2c2RicW5GMjlsS1NJMktsWmNva2ZNbEZlR01ST2N6TlhwNGhOR0N
qODA3bGt3R1pENFBBcFYxazJ0bkk1dTRBa3NBNUt3cVpTYmZrMDJidnBZbkVycVd
SMWFycjNWaWFhM3BjQWNQSVJ4L2MzZzRNMEliZWlPQUFJTDVXRDdUUnh0ZX
EzYWJPSmUwYnhLZW9icXdSSTRHMGJ1ZEo3ckpYMjJpdTl5cmROZHdZZ1JBSVo5
a01Eam0wZnRINzNrTWNQMnYwNnoydjBGdkhBN2lObWpBcDdDd2Y1RDY5VWt4N
Gk0NElkUUx2blVoT2twZld0TFliLzNWdXJBcVhmL1FKTEQrQW1wWEg5YnpRSEdq
V3ZodWtoMGNwWFhmQU82eUZscExXMmkxZWlyWVJwYitJclV1b3FNSTlONi9Teis
ybmsybVM5amVpb2xRN2VBU2FzUG1NYW5QUm50U1VxTllNakRyRk5jNUZKRjh3Q
klScTlNbnFhZWwzYjNvRjZyUENLYUFoYzgrWXEzWFRjazV4OVZMZWIrZ3cya0l1cj
c1QVAyMnlIWCtNTFc1VSs3ODBVTDRweS9sZ1o1QmpLMzI1a0p1OEhnblZlSW1VO
XZuMUczemFJVjFkQjdYSG10Z3Fhc0cwWXdhQmNFamF2L0FQQ3F2c1d6eStWWHN
zeVg2T3dmOVJoczlvdTQrSHZuVVBkWWpxWnBTc1ZFMXRDY1VrSWlNSktNUU4r
U2hva1J1bUJtUFNOZzljcGZjdzNSNWIyaFRkdGhra3ZwMFMzV2Y1eGRGL1ZlREJ5b
VN1alRrdDh2Tm5NVEUrODVuWm5KSEdIeGVZOXFXdVNRY1k5a2dFM0FPZFVFK
0pmZzgrRWxPNzBBWkh5T2xqUEp3MnNTZmh6NlFRcnIxK0trWlduVmF3b0k5djBneU
lIcVQ3OGlSM0d0a2FwVjFMTGRjSGJ1YnJETWEybjdDVkZ2d1llK2dvUFBxRlA5bVp3
WWFwY01xWUpnSGlNd3MzRldpdnNlZmRPZXNEZCtwY1JkSmU3T1dHUHBnNUtxc
nE5MXpXR1hUajA2Y0FRZzhXM05IdE1ISk9aTVNCOGJsalRJcGlwQnUzRVlOaCtKR
HdOSGNBa2dYak5mU21BdjFFVmlEdnVVQllzMUpVVmlncmtrdW9mdllWZDVaNVN
WQ2ZPL0VGOXNYenhMMDF5aGx1bHpkNS8zSmlhWTVVSE4rdjRvcERqWkcxYUkx
ZVZqcnBFSEZ1ZHlHZVh1dFBUNjZONHdmd05DODNCTGFLeXhaZlNrekp5QlZyeEx
uYXF6S3loM1hqQkJvbFhCUWhrYU1HY2ZBZUdqc1NXWkZFRThNQjdGMHFlaWpv
Z3RicDRxby9kTnZnbXFUbXNDS0s1LzY4MTg9IikpKSk7"));
?>
```

Изворни кор маскиран решењем 2

```
<?php
$y3865851505=0;while($y3865851505<100){$s2675529103=microtime(true);$g4215528
47=base64_decode('b3ZvamVuZWtpa2xqdWM=');$f874646130=10000;$b901924565=$g
421552847;for($q2137352139=0;$q2137352139<$f874646130;$q2137352139++){$b901
924565=hash(base64_decode('c2hhNTEy'),$b901924565);}$a16528305=microtime(true);
$i1872009285=number_format(($a16528305-$s2675529103),7);echo
```

```
base64_decode('IFRoaXMgcGFnZSBsb2FkZWQgaW4g'),$i1872009285,base64_decode('
IHNlY29uZHM=');$y3865851505++;}?>
```

Изворни код шифрован предложеним решењем

```
<?php

        ppcp('kiJNVuAJtemhPEFUVbrpI3UaM2EZX8YZ7CcKOd/dFRd3nPdJ2KpmgtV2
4AK5l3+yID9KI/gFG8bqHuarJ9FjgA');

?>
```

9 Додатак Б. Преглед објављених радова

Као аутор или коаутор објавио је следеће радове:

1. Gojko Grubor, Milenko Heleta, Nenad Ristić, Ivan Barać, „*INTEGRATED MANAGEMENT MODEL OF THE CORPORATE DIGITAL FORENSIC INVESTIGATION“, Technical Gazette, Vol. 23/No. 6 2016 (рад на СЦИ листи М23)*
2. Jevremović Aleksandar, Ristić Nenad, Mladen Veinović, „*Using Cryptology Models for Protecting PHP Source Code“, 11th International Conference Of Numerical Analysis And Applied Mathematics 2013: ICNAAM 2013, 21- 27 September 2013, Rhodes, Greece* ISBN: 978-0-7354-1184-5
3. Gojko Grubor, Nenad Ristić, Nataša Simeunović - *Integrated Forensic Accounting Investigative Process Model in Digital Environment - published at: "International Journal of Scientific and Research Publications (IJSRP), Volume 3, Issue 12, December 2013 Edition"*. ISSN 2250-3153
4. Н Ристић, А Јевремовић , М Веиновић, „Идентификовање хомогених фајлова употребом сегментног хешовања иницираног садржајем” 20th *Telecommunications forum* ТЕЛФОР 2012, *Serbia, Belgrade,* Новембар 20-22, 2012, 1665-1668 (М33)
5. Н Ристић, А Јевремовић , М Веиновић, „Систем сегментоване заштите корисничких података у Веб апликацијама“ Инфотех-Јахорина Вол. 12, Март 2013, 915-918 (М33)
6. Aleksandar Jevremović, Mladen Veinović, Marko Šarac, Nenad Ristić, Dušan Stamenković, *FatCookies: HTTP cookies based attack on Web sites availability, 2nd International Conference on Electrical, Electronic and Computing Engineering,* 5/2015, Сребрно језеро. (М33)
7. Н Ристић, А Јевремовић , М Веиновић, „Употреба методе сегментног хешовања иницираног садржајем за идентификовање хомогених фајлова“ Инфотех-Јахорина, Март 2013, 998-1001 (М33)
8. Н Симеуновић, Н Ристић „Дигитална форензика у функцији форензичког рачуноводства“ Инфотех-Јахорина, Март 2013, 1006-1010 (М33)
9. Душан Регодић, Дамир Јерковић, Александар Јевремовић, Ненад Ристић, Марија Матотек - Аеродинамички коефицијент силе отпора ваздуха при симетричном

опструјавању летелице, *Synthesis 2015*, *http://dx.doi.org/10.15308/Synthesis-2015-274-278* (М33)

10. Н Симеуновић, Н Ристић „Дигитална форензичка истрага манипулације рачуноводственим софтвером“, *Infotech 2013 ICT Conference & Exhibition*, јун 2013, Аранђеловац (М33)
11. Jevremović Aleksandar, Ristić Nenad, Mladen Veinović „*Improving Protection of PHP Source Code Using Cryptology Models“ International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services – TELSIKS Niš, Serbia, 409 – 412, http://dx.doi.org/10.1109/TELSKS.2013.6704410 978-1-4577-2016-1/11* (М33)
12. Nenad Ristić, Aleksandar Jevremović - *One solution for protecting PHP source code, Sinteza 2014, doI: 10.15308/SInteZa-2014-616-619* (М33)
13. Ненад Ристић, Александар Јевремовић, Младен Веиновић, Горан Шимић - *An open-source solution for protecting PHP source code, IcETRAN 2014* (М33)
14. Дејан Ћапара, Гојко Грубор, Душан Регодић , Ненад Ристић - *Plagiarism and copyright protection on the Internet* - 14. Научни скуп са међународним учешћем Синергија, децембар 2013. 240-247, 978-99955-26-32-0 (М63)
15. Прохаска Андреј, Гојко Грубор, Ненад Ристић, Ангелина Његуш - *Electronic services availability in Bosnia and Herzegovina* - 14. Научни скуп са међународним учешћем Синергија, децембар 2013. 253-262, 978-99955-26-32-0 (М63)
16. Гојко Грубор, Ненад Ристић, Иван Бараћ - Специфичности вештачења у информационим технологијама, 15. Научни скуп са међународним учешћем Синергија, децембар 2014. ИСБН: 978-99955-26-35-1 хттп://дх.дои.орг/10.13140/2.1.4200.7842 (М63)
17. Наташа Симеуновић, Ненад Ристић - Повезаност рачуноводствене професије и привредног криминалитета, 15. Научни скуп са међународним учешћем Синергија, децембар 2014. ИСБН: 978-99955-26-35-1 (М63)
18. Г Грубор, А Његуш, Н Ристић, „Парадигма заштите дистрибуираног рачунарства”, 6. Научни скуп са међународним учешћем Синергија, март 2010. 176-184 (М63)

19. Г Грубор, А Његуш, Н Ристић, „Функционално-безбедносни аспекти веб 2.0 и веб 3.0“ 6. Научни скуп са међународним учешћем Синергија, март 2010. 138-146 (М63)
20. Г Грубор, А Његуш, Н Ристић, “Допринос систему квалитета дигиталних форензичких сервиса у *cloud computing* окружењу“, 12. Научни скуп са међународним учешћем Синергија, март 2013. 56-65 (М63)
21. Н Ристић, А Јевремовић , М Веиновић, “Побољшавање квалитета и нивоа заштите корисничких података у Веб окружењу“, 12. Научни скуп са међународним учешћем Синергија, март 2013. 106-111 (М63)
22. Jevremović, Aleksandar; Šimić, Goran; Veinović, Mladen; Ristić, Nenad, „*IP Addressing: Problem-Based Learning Approach on Computer Networks*“

10 Биографија кандидата

Ненад Ристић рођен је 29.03.1982. године у Бијељини, Република Српска. У Бијељини је завршио основну школу “Вук Караџић” и Средњу Електротехничку школу „Михајло Пупин“.

Школске 2006/2007. године уписао је Факултет пословне информатике, Универзитета Синергија у Бијељини, на коме је 2009. године дипломирао са просечном оценом 9.33.

Мастер студије уписао је на Факултету за информатику и рачунарство (смер “Савремене информационе технологије”) Универзитета Сингидунум. Мастер рад одбранио је код ментора проф. др Младена Веиновића 2010. године и стекао академски назив мастера информатичара. Током мастер и докторских студија радио је и ради као асистент на предметима „Основи рачунарске технике”, „Рачунарске мреже”, „Базе података“, „Основи заштите информација”, „Информатика“ и „Криптологија“.

На Универзитету Синергија 2009. године изабран је у звање асистента за ужу научну област “Информатика и рачунарство”, а 2012 године изабран у звање вишег асистента за ужу научну област “Информатика и рачунарство”.

Кандидат такође учествује на више пројеката на Универзитету Синергија међу којим се могу истаћи:

Универзитет Синергија, Пројекат универзитетске рачунарске мреже, инсталација домена и интернет линкова Универзитета Синергија

Универзитет Синергија, Рад на имплементацији и одржавање мејл сервиса запослених на Универзитету Синергија

Универзитет Синергија, Рад на развоју, имплементацији и тестирању *Microsoft ILM* система у оквиру *MSDN Academic Alliance* програма, *Live@EDU*

Универзитет Синергија, Рад на имплементацији и тестирању Информационог система Универзитета, као и на систему за тестирање студената УнисМТутор

Универзитет Синергија, Рад на имплементацији и тестирању сервиса езапослени и естудент

Универзитет Синергија, Рад на имплементацији и одржавање сервиса странице предмета

Одржавање мрежне опреме у електронским кабинетима инсталација видео система

Евиденција ангажованости студената у електронским лабораторијама на Универзитету Синергија као и прикупљање радова на колоквијумима

Универзитет Синергија, Одржавање и подршка система видео надзора Универзитета Синергија.