2 Криптографія

2. 1. Класифікація криптоалгоритмов

Сама криптографія не є вищим ступенем класифікації суміжних з нею дисциплін. Навпаки, криптографія спільно з криптоаналізом (метою якого є протистояння методам криптографії) складають комплексну науку – кріптологию.
Необхідно відзначити, що в російськомовних текстах по даному предмету зустрічаються різні вживання основних термінів, таких як "криптографія", "тайнопис" і деяких інших. Більш того, і по класифікації криптоалгоритмов можна зустріти різні думки. У зв'язку з цим автор не претендує на те, що його варіант використовування подібних термінів є єдино вірним.
Відносно криптоалгоритмов існує декілька схем класифікації, кожна з яких заснована на групі характерних ознак. Таким чином, один і той же алгоритм "проходить" відразу за декількома схемами, опиняючись в кожній з них в якій-небудь з підгруп.
Основною схемою класифікації всіх криптоалгоритмов є наступна:

1. Тайнопис.
   Відправник і одержувач проводять над повідомленням перетворення, відомі тільки їм двом. Стороннім особам невідомий сам алгоритм шифрування. Деякі фахівці вважають, що тайнопис не є криптографією взагалі, і автор знаходить це абсолютно справедливим.
2. Криптографія з ключем.
   Алгоритм дії на передавані дані відомий всім стороннім особам, але він залежить від деякого параметра – "ключа", яким володіють тільки відправник і одержувач.
   1. Симетричні криптоалгоритмы.
      Для зашифровуваної і розшифровки повідомлення використовується один і той же блок інформації (ключ).
   2. **Асиметричні криптоалгоритмы.**
      Алгоритм такий, що для зашифровуваної повідомлення використовується один ("відкритий") ключ, відомий всім охочим, а для розшифровки – інший ("закритий"), існуючий тільки у одержувача.

Весь подальший матеріал буде присвячений криптографії з ключем, оскільки більшість фахівців саме по відношенню до цих криптоалгоритмам використовує термін криптографія, що цілком виправдане. Так, наприклад, будь-який криптоалгоритм з ключем можна перетворити на тайнопис, просто "зашивши" в початковому коді програми деякий фіксований ключ. Зворотне ж перетворення практично неможливе.
Залежно від характеру дій, вироблюваних над даними, алгоритми підрозділяються на:

1. Перестановочниє
   Блоки інформації (байти, біті, крупніші одиниці) не змінюються самі по собі, але змінюється їх порядок проходження, що робить інформацію неприступної стороннему спостерігачу.
2. Підстановлювальні
   Самі блоки інформації змінюються по законах криптоалгоритма. Переважна більшість сучасних алгоритмів належить цій групі.

Помітьте: будь-які криптографічні перетворення не збільшують об'єм інформації, а лише змінюють її уявлення. Тому, якщо програма шифрування значно (більш, ніж на довжину

заголовка) збільшує об'єм вихідного файлу, то в її основі лежить неоптимальний, а можливо і взагалі некоректний криптоалгоритм. Зменшення об'єму закодованого файлу можливе тільки за наявності вбудованого алгоритму архівації в криптосистемі і за умови стисливості інформації (так, наприклад, архіви, музичні файли формату MP3, відеозображення формату JPEG стискатися більш ніж на 2-4% не будуть).
Залежно від розміру блоку інформації криптоалгоритмы діляться на:

1. Потокові шифри.
   Одиницею кодування є один біт. Результат кодування не залежить від того, що пройшло раніше вхідного потоку. Схема застосовується в системах передачі потоків інформації, тобто в тих випадках, коли передача інформації починається і закінчується в довільні моменти часу і може випадково уриватися. Найпоширенішими представителямі потокових шифрів є скремблери.
2. Блокові шифри
   Одиницею кодування є блок з декількох байтів (в даний час 4-32). Результат кодування залежить від всіх початкових байтів цього блоку. Схема застосовується при пакетній передачі інформації і кодуванні файлів.

Симетричні криптоалгоритмы

Скремблери

Скремблерами називаються програмні або апаратні реалізації алгоритму, що дозволяє шифрувати побітно безперервні потоки інформації. Сам скремблер представляє з себе набір біт, що змінюються на кожному кроці по певному алгоритму. Після виконання кожного чергового кроку на його виході з'являється шифруючий біт – або 0, або 1, який накладається на поточний біт інформаційного потоку операцією XOR.
Останнім часом сфера вживання ськремблірующих алгоритмів значно скоротилася. Це пояснюється в першу чергу зниженням об'ємів побітной послідовної передачі інформації, для захисту якої були розроблені дані алгоритми. Практично повсюдно в сучасних системах застосовуються мережі з комутацією пакетів, для підтримки конфіденційності якої використовуються блокові шифри. А їх криптостойкость перевершує, і деколи досить значно, криптостойкость скремблерів.
Суть ськремблірованія полягає в побітном зміні проходячого через систему потоку даних. Практично єдиною операцією, що використовується в скремблерах є XOR – що "побітное виключає АБО". Паралельно проходженню інформаційного потоку в скремблері за певним правилом генерується потік біт – кодуючий потік. Як пряме, так і зворотне шифрування здійснюється накладенням по XOR кодуючій послідовності на початкову.
Генерація кодуючої послідовності біт проводиться циклічно з невеликого початкового об'єму інформації – ключа по наступному алгоритму. З поточного набору біт вибираються значення певних розрядів і складаються по XOR між собою. Всі розряди зсовуються на 1 біт, а тільки що набуте значення ("0" або "1") поміщається в наймолодший розряд, що звільнився. Значення, що знаходилося в найстаршому розряді до зсуву, додається в кодуючу послідовність, стаючи черговим її бітом (див. рис.1).

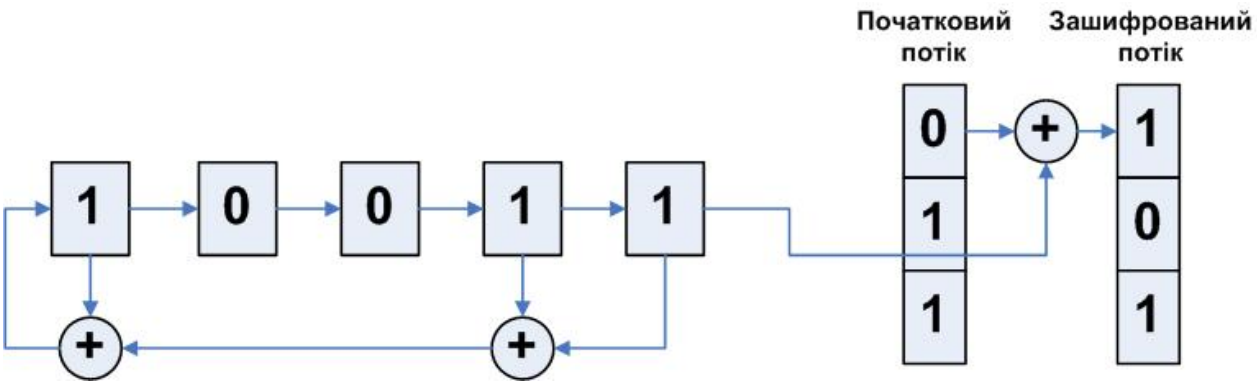


Рис.1.

З теорії передачі даних криптографія запозичувала для запису подібних схем двійкову систему запису. По ній зображений на малюнку скремблер записується комбінацією "100112" – одиниці відповідають розрядам, з яких знімаються біти для формування зворотного зв'язку. Розглянемо приклад кодування інформаційної послідовності 0101112 скремблером 1012 з початковим ключем 1102.

```
скремблер  код.бит   инф.бит   рез-т
 1 1 0  _
  \ \   \_
 1 1 1  _ \_
  \ \   \_  0  XOR  0   =  0
 0 1 1  _ \_
  \ \   \_  1  XOR  1   =  0
 1 0 1    \_
  \ \       1  XOR  0   =  1
```

і т.д.

Як бачимо, пристрій скремблера гранично просто. Його реалізація можлива як на електронній, так і на електричній базі, що і забезпечило його широке вживання в польових умовах. Більш того, той факт, що кожний біт вихідної послідовності залежить тільки від одного вхідного біта, ще більш зміцнило положення скремблерів в захисті потокової передачі даних. Це пов'язано з неминуче виникаючими в каналі передачі перешкодами, які можуть спотворити в цьому випадку тільки ті біти, на які вони доводяться, а не пов'язану з ними групу, байт, як це має місце в блокових шифрах.

Декодування заськремблірованних послідовностей відбувається по тій же самій схемі, що і кодування. Саме для цього в алгоритмах застосовується результуюче кодування по тому, що "виключає АБО" – схема, однозначно відновлена при раськодіруванії без яких-небудь додаткових обчислювальних витрат. Проведемо декодування одержаного фрагмента.

Як Ви можете здогадатися, головна проблема шифрів на основі скремблерів - синхронізація передаючого (кодуючого) і приймаючого (декодуючого) пристроїв. При пропуску або помилковому вставлянні хоча б одного біта вся передавана інформація необоротно втрачається. Тому, в системах шифрування на основі скремблерів дуже велика увага надається методам синхронізації. На практиці для цих цілей звичайно застосовується комбінація двох методів: а) додавання в потік інформації синхронізуючих бітів, наперед відомих приймальній стороні, що дозволяє їй при незнаходженні такого біта активно почати пошук синхронізації з відправником, і б) використовування високоточних генераторів тимчасових імпульсів, що дозволяє в моменти втрати синхронізації проводити декодування бітів інформації, що приймаються, "по пам'яті" без синхронізації.

Число біт, охоплених зворотним зв'язком, тобто розрядність пристрою пам'яті для біт, що породжують кодуючу послідовність, називається розрядністю скремблера. Зображений вище скремблер має розрядність 5. Відносно параметрів криптостойкости дана величина повністю

ідентична довжині ключа блокових шифрів, який буде проаналізований далі. На даному ж етапі важливо відзначити, що чим більше розрядність скремблера, тим вище криптостойкость системи, заснованої на його використовуванні.
При достатньо довгій роботі скремблера неминуче виникає його зациклення. Після виконання певного числа тактів в осередках скремблера створиться комбінація біт, яка в ньому вже одного разу виявлялася, і з цієї миті кодуюча послідовність почне циклічно повторюватися з фіксованим періодом. Дана проблема неустраніма за своєю природою, оскільки в N розрядах скремблера не може перебувати більше 2N комбінацій біт, і, отже, максимум, через, 2N-1 циклів повтор комбінації обов'язково відбудеться. Комбінація "всі нулі" зразу ж виключається з ланцюжка графа станів скремблера – вона приводить скремблер до такого ж положення "всі нулі". Це указує ще і на те, що ключ "всі нулі" непридатний для скремблера. Кожний біт, що генерується при зсуві, залежить тільки від декількох біт береженої в даний момент скремблером комбінації. Тому після повторення деякої ситуації, що одного разу вже зустрічалася в скремблері, всі наступні за нею повторюватимуть в точності ланцюжок, що вже пройшов раніше в скремблері.
_Можливі різні типи графів стану скремблера. На малюнку 2 приведені зразкові варіанти для 3-розрядного скремблера. У разі "А" окрім завжди присутнього циклу "000">>"000" ми бачимо ще два цикли – з 3-мя станами і 4-мя. У разі "Б" ми бачимо ланцюжок, який сходиться до циклу з 3-х станів і вже ніколи звідти не виходить. І нарешті, у разі "В" всі можливі стани окрім нульового, об'єднані в один замкнутий цикл. Очевидно, що саме в цьому випадку, коли всі 2N-1 станів системи утворюють цикл, період повторення вихідних комбінацій максимальний, а кореляція між довжиною циклу і початковим станом скремблера (ключем), яка привела б до появи слабкіших ключів, відсутня.

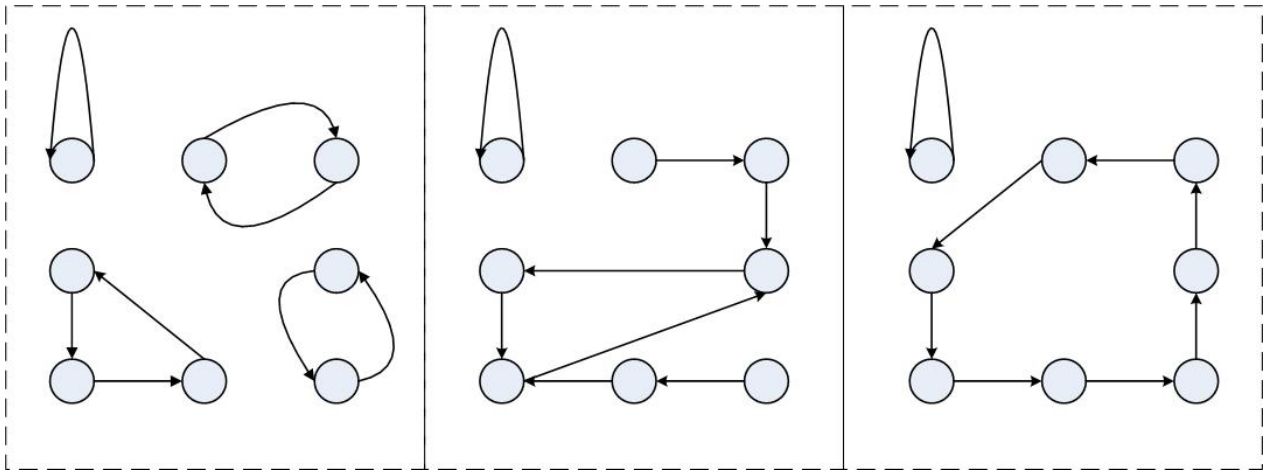


Рис.2.

І ось тут математика піднесла прикладній науці, якій є криптографія, черговий подарунок. Слідством однієї з теорем доводиться (в термінах стосовно ськремблірованію), що для скремблера будь-якої розрядності N завжди існує такий вибір охоплюваних зворотним зв'язком розрядів, що послідовність біт, що генерується ними, матиме період, рівний 2N-1 бітам. Так, наприклад, в 8-бітовому скремблері, при обхваті 0-го, 1-го, 6-го і 7-го розрядів генерації 255 біт послідовно проходять всі числа від 1 до 255, не повторюючись жодного разу.
Схеми з вибраними по даному закону зворотними зв'язками називаються генераторами послідовностей найбільшої довжини (ПНД), і саме вони використовуються в ськремблірующей апаратурі. З безлічі генераторів ПНД заданої розрядності за доби, коли вони реалізовувалися на електричній або мінімальній електронній базі вибиралися ті, у яких

число розрядів, що беруть участь в створенні чергового біта, було мінімальним. Звичайно генератора ПНД вдавалося досягти за 3 або 4 зв'язки. Сама ж розрядність скремблерів перевищувала 30 біт, що давало можливість передавати до 240 біт = 100 Мбайт інформації без побоювання початку повторення кодуючої послідовності.
ПНД нерозривно пов'язані з математичною теорією поліномів, що не приводяться. Виявляється, достатньо щоб поліном ступеня N не був представимо по модулю 2 у вигляді твору ніяких інших поліномів, для того, щоб скремблер, побудований на його основі, створював ПНД. Наприклад, єдиним поліномом ступеня 3, що не приводиться, є x3+x+1, в двійковому вигляді він записується як 10112 (одиниці відповідають присутнім розрядам). Скремблери на основі поліномів, що не приводяться, утворюються відкиданням найстаршого розряду (він завжди присутній, а отже, несе інформацію тільки про ступінь полінома), так на основі вказаного полінома, ми можемо створити скремблер 0112 з періодом зациклення 7(=23-1). Природно, що на практиці застосовуються поліноми значно вищих порядків. А таблиці поліномів будь-яких порядків, що не приводяться, можна завжди знайти в спеціалізованих математичних довідниках.
_Істотним недоліком ськремблірующих алгоритмів є їх нестійка до фальсифікації. Докладніше дана проблема розглянута на наступній лекції, стосовно створення цілих криптосистем.

Блокові шифри

2.2.2.1. Общие сведения о блочных шифрах (19 кб)
На сьогоднішній день розроблено достатньо багато стійких блокових шифрів. Практично всі алгоритми використовують для перетворень певний набір бієктівних (оборотних) математичних перетворень
2.2.2.2. Сеть Фейштеля
Мережею Фейштеля називається метод оборотних перетворень тексту, при якому значення, обчислене від однієї з частин тексту, накладається на інші частини. Часто структура мережі виконується таким чином, що для шифрування і дешифрованія використовується один і той же алгоритм – відмінність полягає тільки в порядку використовування матеріалу ключа.
2.2.2.3. Блочный шифр TEA
Блоковий алгоритм TEA приведений як приклад одного з найпростіших в реалізації стійких криптоалгоритмов.
2.2.2.4. AES : стандарт блочных шифров США с 2000 года
У 1998 році був оголошений відкритий конкурс на криптостандарт США на декілька перших десятиліть XXI століття. Переможцем конкурсу був визнаний бельгійський блоковий шифр Rijndael. Швидше за все він стане стандартом де-факто блокового шифрування у всьому світі.

Загальні відомості про блокові шифри

Блокові шифри шифрують цілі блоки інформації (від 4 до 32 байт) як єдине ціле – це значно збільшує стійкість перетворень до атаки повним перебором і дозволяє використовувати різні математичні і алгоритмічні перетворення.

Характерною особливістю блокових криптоалгоритмов є той факт, що в ході своєї роботи вони проводять перетворення блоку вхідної інформації фіксованої довжини і одержують

результуючий блок того ж об'єму, але неприступний для прочитання стороннім особам, що не володіють ключем. Таким чином, схему роботи блокового шифру можна описати функціями Z=EnCrypt(X,Key) і X=DeCrypt(Z,Key)
Ключ Key є параметром блокового криптоалгоритма і є деяким блоком двійкової інформації фіксованого розміру. Початковий (X) і зашифрований (Z) блоки даних також мають фіксовану розрядність, рівну між собою, але необов'язково рівну довжині ключа.
Блокові шифри є основою, на якій реалізовані практично всі криптосистеми. Методика створення ланцюжків із зашифрованих блоковими алгоритмами байт дозволяє шифрувати ними пакети інформації необмеженої довжини. Така властивість блокових шифрів, як швидкість роботи, використовується асиметричними криптоалгоритмами, повільними за своєю природою. Відсутність статистичної кореляції між бітами вихідного потоку блокового шифру використовується для обчислення контрольних сум пакетів даних і в хешуванні паролів.
Наступні розробки всесвітньо визнані стійкими алгоритмами і публікацій про універсальні методи їх злому в засобах масової інформації на момент створення матеріалу не зустрічалося.

Назва алгоритму	**Автор**	**Розмір блоку**	**Довжина ключа**
IDEA	Xuejia Lia and James Massey	64 біти	128 біт
CAST128		64 біти	128 біт
BlowFish	Bruce Schneier	64 біти	128 – 448 біт
ГОСТ	НДІ ***	64 біти	256 біт
TwoFish	Bruce Schneier	128 біт	128 – 256 біт
MARS	Корпорація IBM	128 біт	128 – 1048 біт

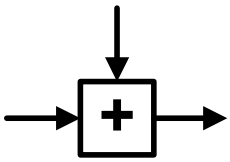
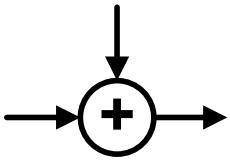
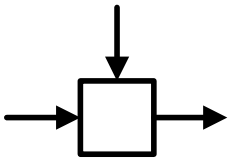
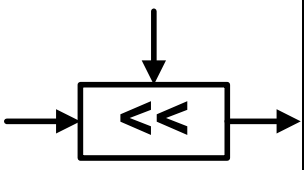
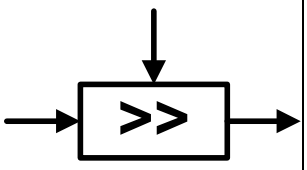
Криптоалгоритм іменується ідеально стійким, якщо прочитати зашифрований блок даних можна тільки перебравши всі можливі ключі, до тих пір, поки повідомлення не виявиться осмисленим. Оскільки по теорії вірогідності шуканий ключ буде знайдений з вірогідністю 1/2 після перебору половини всіх ключів, то на злом ідеально стійкого криптоалгоритма з ключем довжини N потрібно в середньому 2N-1 перевірок. Таким чином, в загальному випадку стійкість блокового шифру залежить тільки від довжини ключа і зростає експоненціально з її зростанням. Навіть припустивши, що перебір ключів проводиться на спеціально створеній багатопроцесорній системі, в якій завдяки діагональному паралелізму на перевірку 1 ключа йде тільки 1 такт, то на злом 128 бітового ключа сучасній техніці потрібно не менше 1021 років. Природно, все сказане відноситься тільки до ідеально стійких шифрів, якими, наприклад, з великою часткою упевненості є приведені в таблиці вище алгоритми.
_Окрім цієї умови до ідеально стійких криптоалгоритмам застосовується ще одна дуже важлива вимога, якій вони повинні обов'язково відповідати. При відомих початковому і зашифрованому значеннях блоку ключ, яким проведено це перетворення, можна взнати також тільки повним перебором. Ситуації, в яких сторонньому спостерігачу відома частина початкового тексту зустрічаються повсюдно. Це можуть бути стандартні написи в електронних бланках, фіксовані заголовки форматів файлів, досить що часто зустрічаються в тексті довгі слова або послідовності байт. В світлі цієї проблеми описана вище вимога не є нічим надмірним і також строго виконується стійкими криптоалгоритмами, як і перше.
Таким чином, на функцію стійкого блокового шифру Z=EnCrypt(X,Key) накладаються наступні умови:

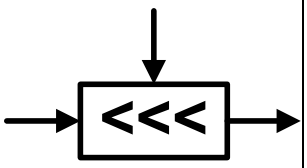
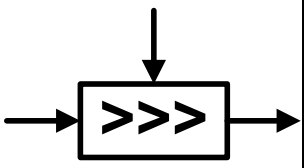
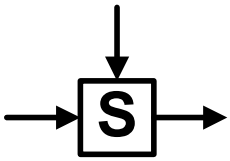
1. Функція EnCrypt повинна бути оборотною.

2. Не повинне існувати інших методів прочитання повідомлення X по відомому блоку Z, окрім як повним перебором ключів Key.
3. Не повинне існувати інших методів визначення яким ключем Key було проведено перетворення відомого повідомлення X в повідомлення Z, окрім як повним перебором ключів.

Давайте розглянемо методи, за допомогою яких розробники блокових криптоалгоритмов добиваються одночасного виконання цих трьох умов з дуже великою часткою достовірності. Всі дії, вироблювані над даними блоковим криптоалгоритмом, засновані на тому факті, що перетворюваний блок може бути представлений у вигляді цілого ненегативного числа з діапазону, відповідного його розрядності. Так, наприклад, 32-бітовий блок даних можна інтерпретувати як число з діапазону 0..4'294'967'295. Крім того, блок, розрядність якого звичайно є "ступенем двійки", можна трактувати як декілька незалежних ненегативних чисел з меншого діапазону (розглянутий вище 32-бітовий блок можна також представити у вигляді 2 незалежних чисел з діапазону 0..65535 або у вигляді 4 незалежних чисел з діапазону 0..255). Над цими числами блоковим криптоалгоритмом і проводяться по певній схемі наступні дії (зліва дані умовні позначення цих операцій на графічних схемах алгоритмів) :

Бієктівниє математичні функції

	Додавання	X'=X+V
	Виключаюче АБО	X'=X XOR V
	Множення по модулю 2^N	X'=(X*V) mod (2^N)
	Арифметичний зсув вліво	X'=X SHL V
	Арифметичний зсув вправо	X'=X SHR V

	Циклічний зсув вліво	X'=X ROL V
	Циклічний зсув вправо	X'=X ROR V
	Табличні підстановки S-box	X'=Table[X,V]

Як параметр V для будь-якого з цих перетворень може використовуватися:

1. фіксоване число (наприклад, X'=X+125)
2. число, одержуване з ключа (наприклад, X'=X+F(Key))
3. число, одержуване з незалежної частини блоку (наприклад, X2'=X2+F(X1))

Останній варіант використовується в схемі, названій на ім'я її творця мережею Фейштеля (німий. Feistel).
Послідовність виконуваних над блоком операцій, комбінації перерахованих вище варіантів V і самі функції F і складають "ноу-хау" кожного конкретного блокового криптоалгоритма.
Розмір блоків і довжина ключа сучасних (1999 рік) алгоритмів були нами розглянуті раніше. Один-два разу на рік дослідницькі центри миру публікують черговий блоковий шифр, який під лютою атакою криптоаналітиків або придбаває за декілька років статус стійкого криптоалгоритма, або (що відбувається невимірний частіше) безславно йде в історію криптографії.
Характерною ознакою блокових алгоритмів є багатократне і непряме використовування матеріалу ключа. Це диктується в першу чергу вимогою неможливості зворотного декодування відносно ключа при відомих початковому і зашифрованому текстах. Для вирішення цієї задачі в приведених вище перетвореннях найчастіше використовується не саме значення ключа або його частини, а деяка, іноді необоротна (небієктівная) функція від матеріалу ключа. Більш того, в подібних перетвореннях один і той же блок або елемент ключа використовується багато разів. Це дозволяє при виконанні умови оборотності функції щодо величини X зробити функцію необоротної щодо ключа Key.
Оскільки операція зашифровуваної або розшифровки окремого блоку в процесі кодування пакету інформації виконується багато разів (іноді до сотень тисяч раз), а значення ключа і, отже, функцій Vi(Key) залишається незмінним, то іноді стає доцільно наперед однократно обчислити дані значення і берегти їх в оперативній пам'яті спільно з ключем. Оскільки ці значення залежать тільки від ключа, то оїн в криптографії називаються матеріалом ключа.
Необхідно відзначити, що дана операція жодним чином не змінює ні довжину ключа, ні криптостойкость алгоритму в цілому. Тут відбувається лише оптимізація швидкості

обчислень шляхом кешированія (англ. caching) проміжних результатів. Описані дії зустрічаються практично в багатьох блокових криптоалгоритмах і носять назву розширення ключа (англ. key scheduling)

Мережа Фейштеля

Мережа Фейштеля є подальшою модифікацією описаного вище методу змішування поточної частини шифрованого блоку з результатом деякої функції, обчисленої від іншої незалежної частини того ж блоку. Ця методика набула широке поширення, оскільки забезпечує виконання вимоги про багатократне використовування ключа і матеріалу початкового блоку інформації.
Класична мережа Фейштеля має наступну структуру:

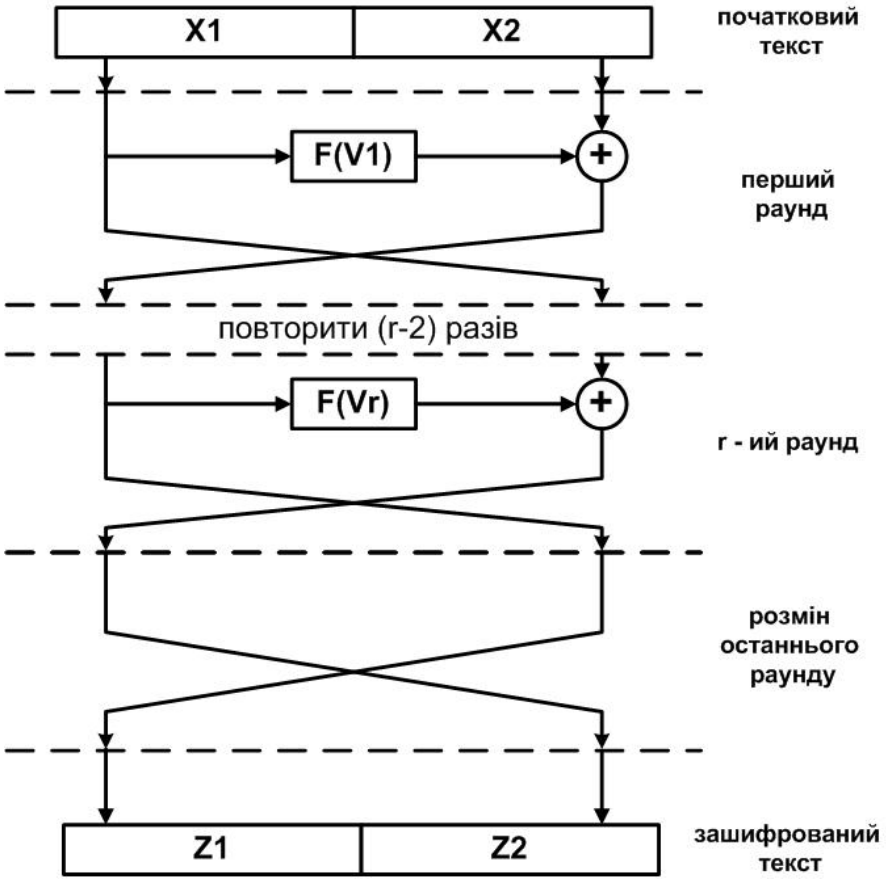


Рис.1.

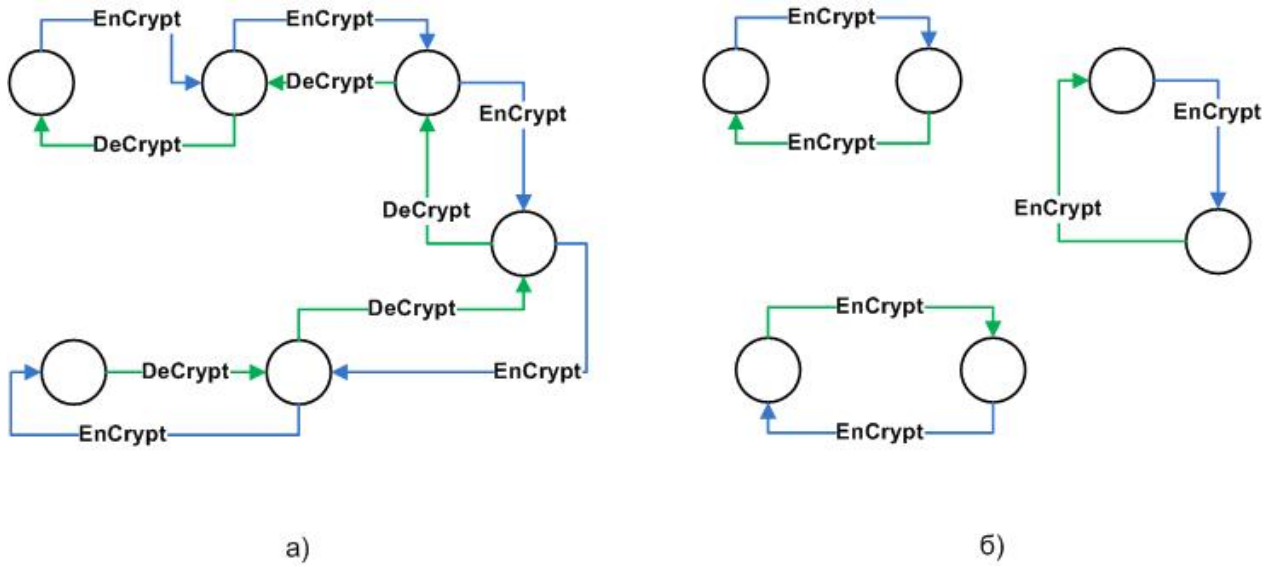
Незалежні потоки інформації, породжені з початкового блоку, називаються гілками мережі. В класичній схемі їх дві. Величини Vi іменуються параметрами мережі, звично це функції від матеріалу ключа. Функція F називається створюючою. Дія, що складається з однократного обчислення функції твірної і подальшого накладення її результату на іншу гілку з обміном їх місцями, називається циклом або раундом (англ. round) мережі Фейштеля. Оптимальне число раундів До – від 8 до 32. Важливе те, що збільшення кількості раундів значно збільшує кріптоскойстость будь-якого блокового шифру до криптоаналізу. Можливо, ця особливість і вплинула на так активне розповсюдження мережі Фейштеля – адже при виявленні, скажімо, якого-небудь слабкого місця в алгоритмі, майже завжди достатньо збільшити кількість

раундів на 4-8, не переписуючи сам алгоритм. Часто кількість раундів не фіксується розробниками алгоритму, а лише указуються розумні межі (обов'язково нижній, і не завжди – верхній) цього параметра.

Зразу ж виникає питання, – чи є дана схема оборотною ? Очевидно, так. Мережа Фейштеля володіє тією властивістю, що навіть якщо як функція твірної F буде використано необоротне перетворення, то і в цьому випадку весь ланцюжок буде відновлений. Це відбувається унаслідок того, що для зворотного перетворення мережі Фейштеля не потрібно обчислювати функцію F-1.

Більш того, як неважко помітити, мережа Фейштеля симетрична. Використовування операції XOR, оборотної своїм же повтором, і інверсія останнього обміну гілок роблять можливим раськодірованіє блоку тією ж мережею Фейштеля, але з інверсним порядком параметрів Vi. Помітимо, що для оборотності мережі Фейштеля не має значення чи є число раундів парним або непарним числом. В більшості реалізацій схеми, в яких обидва вищеперелічені умови (операція XOR і знищення останнього обміну) збережені, пряме і зворотне перетворення проводяться однією і тією ж процедурою, якій як параметр передається вектор величин Vi або в початковому, або в інверсному порядку.

З незначними доробками мережу Фейштеля можна зробити і абсолютно симетричною, тобто виконуючої функції шифрування і дешифрованія одним і тим же набором операцій. Математичною мовою це записується як "Функція EnCrypt тотожно рівна функції DeCrypt". Якщо ми розглянемо граф станів криптоалгоритма, на якому крапками наголошено на блоках вхідної і вихідної інформації, то при якомусь фіксованому ключі для класичної мережі Фейштеля ми матимемо картину, зображену на рис.2а, а в другому випадку кожна пара крапок одержить унікальний зв'язок, як зображено на мал. 2б. Модифікація мережі Фейштеля, що володіє подібними властивостями приведена на малюнку 3. Як бачимо, основна її хитрість в повторному використовуванні даних ключа в зворотному порядку в другій половині циклу. Необхідно помітити, проте, що саме через цю недостатньо дослідженої специфіки такої схеми (тобто потенційної можливості ослаблення зашифрованого тексту зворотними перетвореннями) її використовують в криптоалгоритмах з великою обережністю.



а) б)

Рис.2.

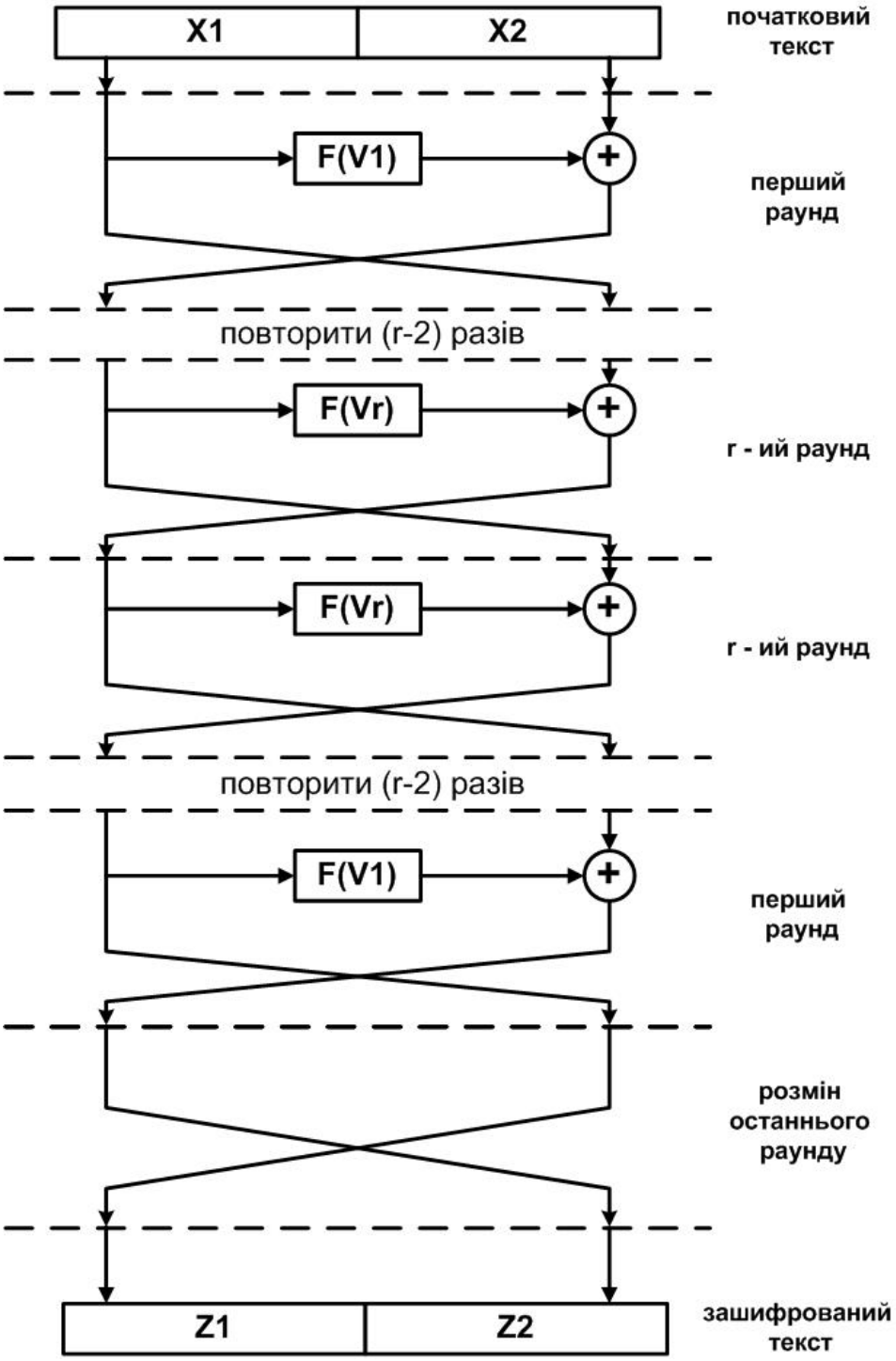
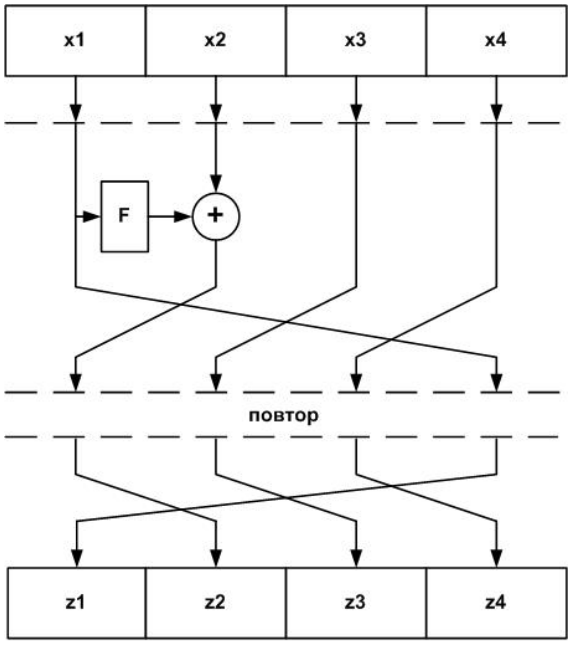


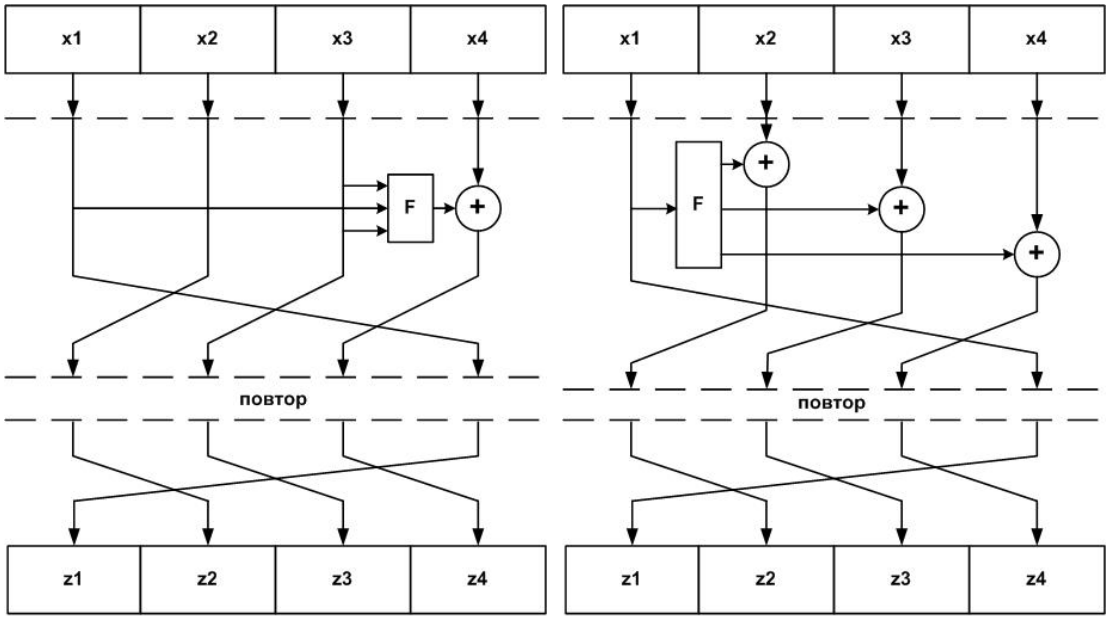
Рис.3.

А ось модифікацію мережі Фейштеля для більшого числа гілок застосовують набагато частіше. Це в першу чергу пов'язано з тим, що при великих розмірах кодованих блоків (128 і більш битий) стає незручно працювати з математичними функціями по модулю 64 і вище. Як відомо, основні одиниці інформації оброблювані процесорами на сьогоднішній день – це байт і подвійне машинне слово 32 біти. Тому все частіше і частіше в блокових криптоалгоритмах зустрічається мережа Фейштеля з 4-мя гілками. Найпростіший принцип її модифікації зображений на малюнку 4а. Для швидшого перемішування інформації між гілками (а це

основна проблема мережі Фейштеля з великою кількістю гілок) застосовуються дві модифіковані схеми, звані "type-2" і "type-3". Вони зображені на малюнках 4б і 4в.



а)



б) в)

Рис.4.

Мережа Фейштеля надійно зарекомендувала себе як криптостойкая схема твору перетворень, і її можна знайти практично в будь-якому сучасному блоковому шифрі. Незначні модифікації торкаються звичайно додаткових початкових і крайових перетворень (англомовний термін –

whitening) над шифрованим блоком. Подібні перетворення, виконувані звично також або що "виключає АБО" або складанням мають метою підвищити початкову рандомізацію вхідного тексту. Таким чином, криптостойкость блокового шифру, що використовує мережу Фейштеля, визначається на 95% функцією F і правилом обчислення Vi з ключа. Ці функції і є об'єктом всі нових і нових досліджень фахівців в області криптографії.

Блоковий шифр TEA

Розглянемо один з найпростіших в реалізації, але визнаний стійких криптоалгоритмов – TEA (Tiny Encryption Algorithm).
Параметри алгоритму :
Розмір блоку – 64 біти.
Довжина ключа – 128 біт.
В алгоритмі використана мережа Фейштеля з двома гілками в 32 біти кожна.
Функція твірної F обратіма.Мережа Фейштеля несиметрична через використовування як операція накладення не виключає "АБО", а арифметичного складання.
Нижче приведений код криптоалгоритма на мові програмування PASCAL.

```
type TLong2=array[0.. 1] longint;
   TLong2x2=array[0.. 1] TLong2;
const Delta=$9E3779B9;
var   key:TLong2x2;
procedure EnCryptRouting(var data);
var y,z,sum:longint; a:byte;
begin
y:=TLong2(data)[0];z:=TLong2(data)[1];sum:=0;
for a:=0 to 31 do
  begin
  inc(sum,Delta);
  inc(y,((z shl 4)+key[0,0]) xor (z+sum) xor ((z shr 5)+key[0,1]));
  inc(z,((y shl 4)+key[1,0]) xor (y+sum) xor ((y shr 5)+key[1,1]));
  end;
TLong2(data)[0]:=y;TLong2(data)[1]:=z
end;
```

Схема роботи алгоритму приведена на малюнку 1.

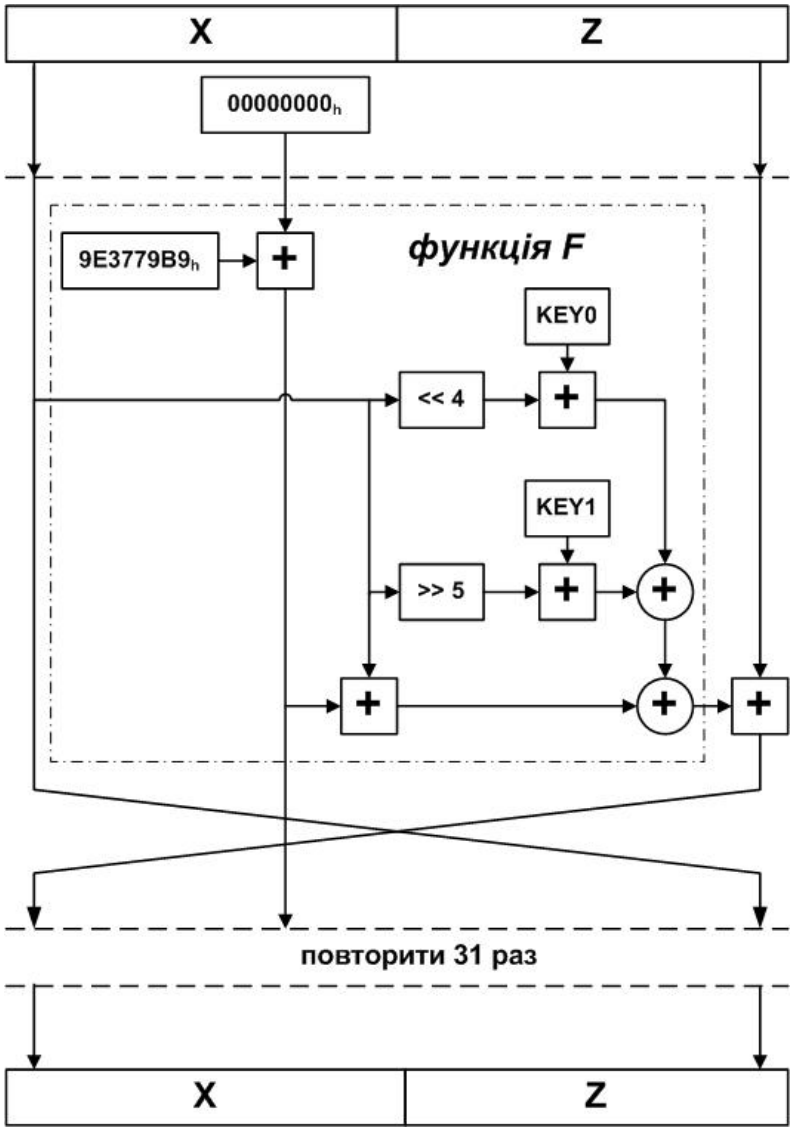


Рис.1.

Відмінною рисою криптоалгоритма TEA є його розмір. Простота операцій, відсутність табличних підстановок і оптимізація під 32-розрядну архітектуру процесорів дозволяє реалізувати його на мові ASSEMBLER в гранично малому об'ємі коду. Недоліком алгоритму є деяка повільність, викликана необхідністю повторювати цикл Фейштеля 32 разу (це необхідно для ретельного "перемішування даних" через відсутність табличних підстановок).

AES : стандарт блокових шифрів США з 2000 року

2.2.2.4.1. Общие сведения о конкурсе AES

Вимоги до конкурсантів AES, коротка інформація про Національний Інститут Стандартизації США. На конкурс було подано 15 заявок, перший етап відбору пройшли тільки 5 претендентів – фіналістів AES.

2.2.2.4.2. Финалист AES – шифр MARS

Розробка корпорації IBM, заснована на класичній мережі Фейштеля і численних математичних операціях.

2.2.2.4.3. Финалист AES – шифр RC6
Модифікація широко відомого блокового шифру RC5. Використовує тільки основні математичні перетворення, бітові зсуви і, як функція перемішування, операцію T(X)=X*(X+1).
2.2.2.4.4. Финалист AES – шифр Serpent
Шифр використовує тільки операції табличних підстановок, що виключає "АБО" і бітових зсувів в ретельно підібраній черговості.
2.2.2.4.5. Финалист AES – шифр TwoFish
Достатньо складна в реалізації розробка компанії Counterpane Security Systems, що утілила багато цікавих ідей з алгоритму попередника BlowFish.
2.2.2.4.6. Победитель AES – шифр Rijndael
Блоковий шифр, реалізований не за принципом мережі Фейштеля, і що має надійну математичну базу перетворень. Структура алгоритму дозволяє досягти дуже високого ступеня оптимізації на персональних ЕОМ і серверах, і в той же час не дуже уповільнює виконання на електронних пристроях з обмеженими ресурсами.

Загальні відомості про конкурс AES

У 80-х роках в США був прийнятий стандарт симетричного криптоалгоритма для внутрішнього вживання DES (Data Encryption Standard), який набув достатньо широке поширення свого часу. Проте на даний момент цей стандарт повністю неприйнятний для використовування з двох причин : 1) основний – довжина його ключа складає 56 біт, що надзвичайно мале на сучасному етапі розвитку ЕОМ, 2) другорядної – при розробці алгоритм був орієнтований на апаратну реалізацію, тобто містив операції, виконувані на мікропроцесорах за непріємлімо великий час (наприклад, такі як перестановка біт усередині машинного слова по певній схемі).
Все це сподвігло Американський інститут стандартизації NIST – National Institute of Standards & Technology на оголошення в 1997 році конкурсу на новий стандарт симетричного криптоалгоритма. На цей раз вже були враховані основні промахи шифру-попередника, а до розробки були підключені найкрупніші центри по кріптологиі зі всього світу. Тим самим, переможець цього змагання, названого AES – Advanced Encryption Standard, стане де-факто світовим криптостандартом на найближчі 10-20 років.
Вимоги, пред'явлені до кандідітам на AES в 1998 році, були гранично прості :

1. алгоритм повинен бути симетричним,
2. алгоритм повинен бути блоковим шифром,
3. алгоритм повинен мати довжину блоку 128 біт, і підтримувати три довжини ключа : 128, 192 і 256 біт.

Додатково кандидатам рекомендувалося:

1. використовувати операції, легко реалізовувані як апаратний (в мікрочіпах), так і програмно (на персональних комп'ютерах і серверах),
2. орієнтуватися на 32-розрядні процесори,
3. не ускладнювати без необхідності структуру шифру для того, щоб всі зацікавлені сторони були в змозі самостійно провести незалежний криптоаналіз алгоритму і переконатися, що в ньому не закладено яких-небудь недокументованих можливостей.

На першому етапі в оргкомітет змагання поступило 15 заявок з абсолютно різних кутів миру. Протягом 2 років фахівці комітету, досліджуючи самостійно, і вивчаючи публікації інших дослідників, вибрали 5 кращих представників, що пройшли в "фінал" змагання.

Алгоритм	**Творець**	**Країна**	**Швидкодія (asm, 200МГц)**
MARS	IBM	US	8 Мбайт/з
RC6	R.Rivest & Co	US	12 Мбайт/з
Rijndael	V.Rijmen & J.Daemen	BE	7 Мбайт/з
Serpent	*Universities*	IS, UK, NO	2 Мбайт/з
TwoFish	B.Schneier & Co	US	11 Мбайт/з

Всі ці алгоритми були визнані достатньо стійкими і успішно протистоячими всім широко відомим методам криптоаналізу.
2 жовтня 2000 року NIST оголосив про свій вибір – переможцем конкурсу став бельгійський алгоритм RIJNDAEL. З цієї миті з алгоритму-переможця зняті всі патентні обмеження – його можна буде використовувати в будь-якій криптопрограмме без відрахування яких-небудь засобів творцю.
Нижче ми розглянемо основні (робітництво) частини алгоритмів переможців першого етапу. Об'єм лекції не дозволяє привести для кожного алгоритму методи створення S-box'ов (таблиць для табличних підстановок) і методи розширення матеріалу ключа. Повний опис всі 15 алгоритмів претендентів на AES, включаючи дослідження по їх криптостойкости можна знайти на сервері інституту NIST, вказаному вище.

Фіналіст AES – шифр MARS

Шифр складається з трьох видів операцій, які повторюються спочатку в прямому, а потім в інверсному порядку. На першому кроці йде класичне вхідне забілення : до всіх байтів початкового тексту додаються байти з матеріалу ключа.
Другий етап : пряме перемішування, однотипна операція, що має структуру мережі Фейштеля повторюється 8 разів. Проте, на цьому етапі не проводиться додавання матеріалу ключа. Мета даного перетворення – ретельна рандомізація даних і підвищення стійкості шифру до деяких видів атак (рис.1).
Третій етап : власне шифрування. В ньому використовується мережа Фейштеля треьего типу з 4 гілками, тобто значення трьох функцій, обчислених від однієї гілки накладаються відповідно на три інших, потім йде перестановка машинних слів. Ця операція також повторюється 8 разів (рис.1). Саме на цьому етапі відбувається змішування тексту з основною (більшої) частиною матеріалу ключа. Самі функції, що накладаються на гілці, зображені на рис.2. Як бачимо, в алгоритмі MARS використані практично всі види операцій, вживаних в криптографічних перетвореннях : складання, що "виключає АБО", зсув на фіксоване число біт, зсув на змінне число біт, множення і табличні підстановки.
У другій частині операції шифрування повторюються ті ж операції, але в зворотному порядку : спочатку шифрування, потім перемішування, і, нарешті, забілення. При цьому в другі варіанти всіх операцій внесені деякі зміни так, щоб криптоалгоритм в цілому став абсолютно симетричним. Тобто, в алгоритмі MARS для будь-кого X виконується вираз EnCrypt(EnCrypt(X))=X

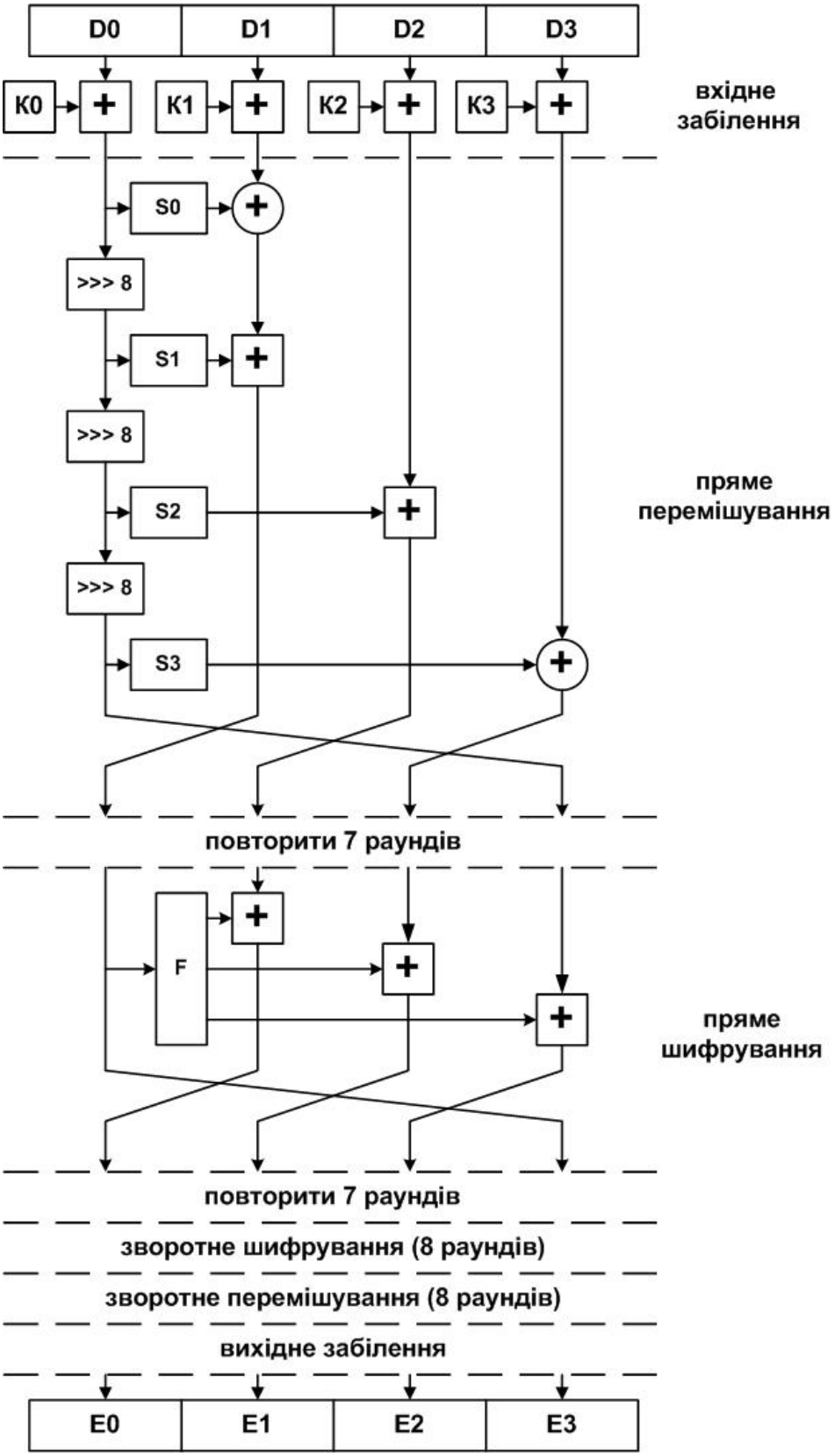

D0
D1
D2
D3
K0
K1
K2
K3
вхідне забілення
S0
>>> 8
S1
>>> 8
S2
>>> 8
S3
пряме перемішування
повторити 7 раундів
F
пряме шифрування
повторити 7 раундів
зворотне шифрування (8 раундів)
зворотне перемішування (8 раундів)
вихідне забілення
E0
E1
E2
E3

Рис.1.

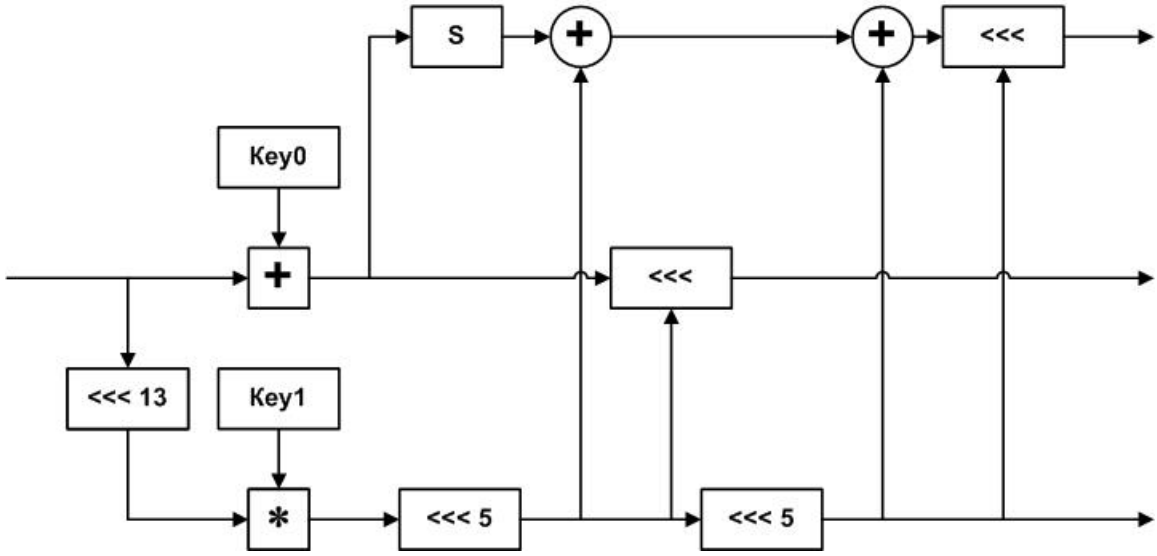


Рис.2.

Фіналіст AES – шифр RC6

Алгоритм є продовженням криптоалгоритма RC5, розробленого Рональдом Рівестом (англ. Ron Rivest) – дуже відомою особою в світі криптографії. RC5 був трохи змінений для того, щоб відповідати вимогам AES по довжині ключа і розміру блоку. При цьому алгоритм став ще швидшим, а його ядро, успадковуване від RC5, має солідний запас досліджень, проведених задовго до оголошення конкурсу AES.
Алгоритм є мережею Фейштеля з 4 гілками змішаного типу : в ньому два парні блоки використовуються для одночасної зміни вмісту двох непарних блоків. Потім проводиться звичний для мережі Фейштеля зсув на одне машинне слово, що міняє парні і непарні блоки місцями. Сам алгоритм гранично простий і зображений на малюнку 1. Розробники рекомендують використовувати при шифруванні 20 раундів мережі, хоча у принципі їх кількість не регламентується. При 20 повторах операції шифрування алгоритм має найвищу швидкість серед 5 фіналістів AES.

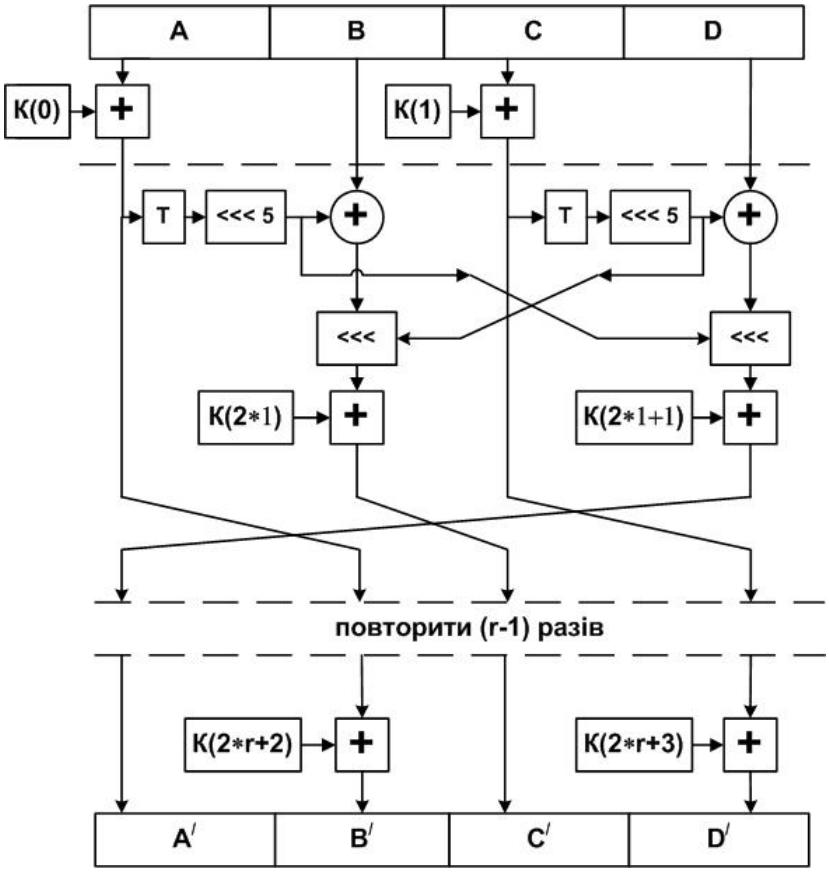


Рис.1.

Перетворення T(x) дуже просто : T(X)=(X*(X+1)) mod 2N. Воно використовується як нелінійне перетворення з хорошими показниками перемішування бітового значення вхідної величини.

Фіналіст AES – шифр Serpent

Алгоритм розроблений групою учених з декількох дослідницьких центрів миру. Алгоритм є мереж Фейштеля для чотирьох гілок змішаного типу : 2 парні гілки змінюють совместо значення непарних, потім міняються місцями. Як криптопреобразований використовуються що тільки виключає "АБО", табличні підстановки і бітові зсуви. Алгоритм складається з 32 раундів. Самі раунди складені таким чином, що додавання до гілками матеріалу ключа на першому і останньому раундах утворює вхідне і вихідне забілення.

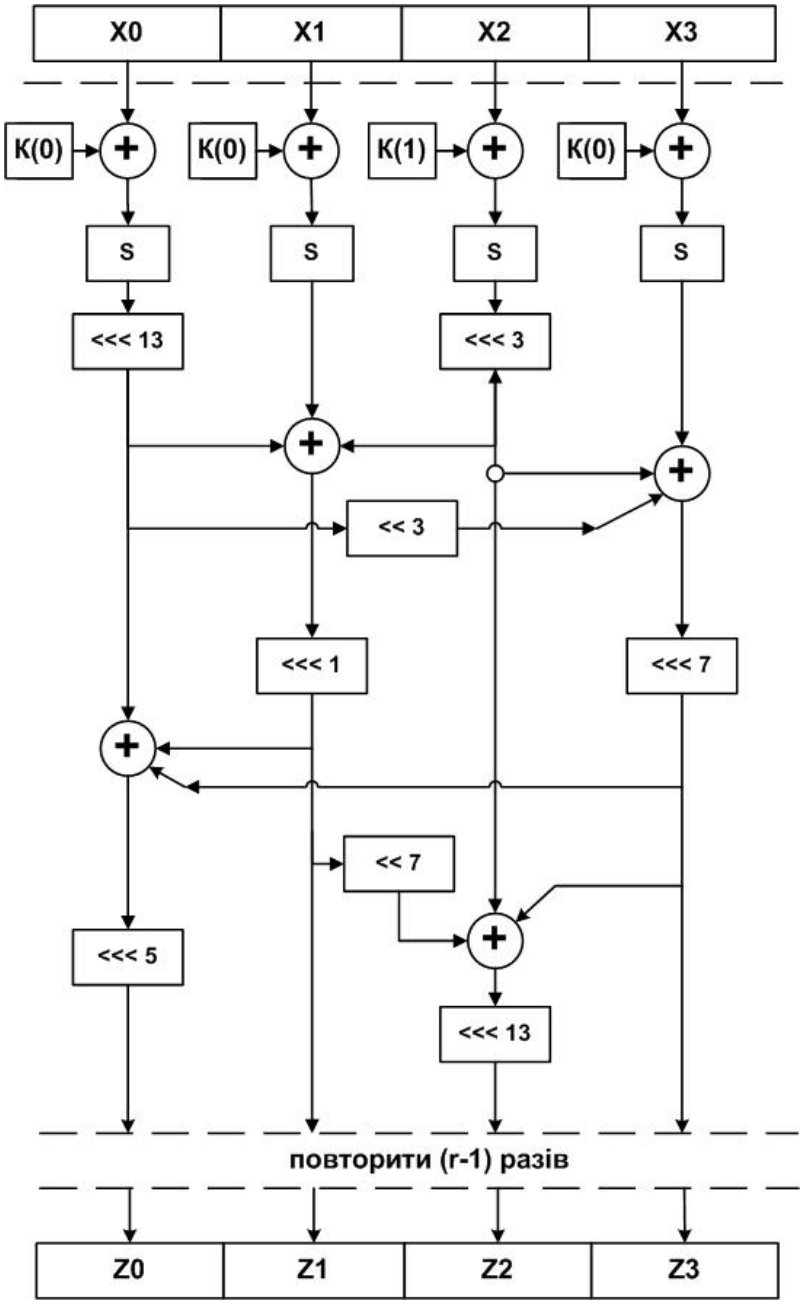


Рис.1.

Фіналіст AES – шифр TwoFish

Алгоритм розроблений командою Counterpain Security Systems, очолюваної Брюсом Шнайером (англ. Bruce Schneier). Попередня програмна розробка цієї фірми, BlowFish, що називалася, була і дотепер є визнаним криптостойким алгоритмом.

У алгоритмі TwoFish розробники залишили деякі вдалі рішення з проекту-попередника, окрім цього провели ретельні дослідження по перемішуванню даних в мережі Фейштеля. Алгоритм є мережею Фейштеля змішаного типу: перша і друга гілка на непарних раундах проводять модифікацію третьої і четвертої, на парних раундах ситуація міняється на протилежну. В

алгоритмі використовується криптопреобразование Адамара (англ. Pseudo-Hadamar Transform) – оборотне арифметичне складання першого потоку з другим, а потім другого з першим.
Єдиним наріканням, що поступило на адресу TwoFish від незалежних дослідників, є той факт, що при розширенні матеріалу ключа в алгоритмі використовується сам же алгоритм. Подвійне вживання блокового шифру досить сильно ускладнює його аналіз на предмет наявності слабких ключів або недокументованих замаскованих зв'язків між вхідними і вихідними даними.

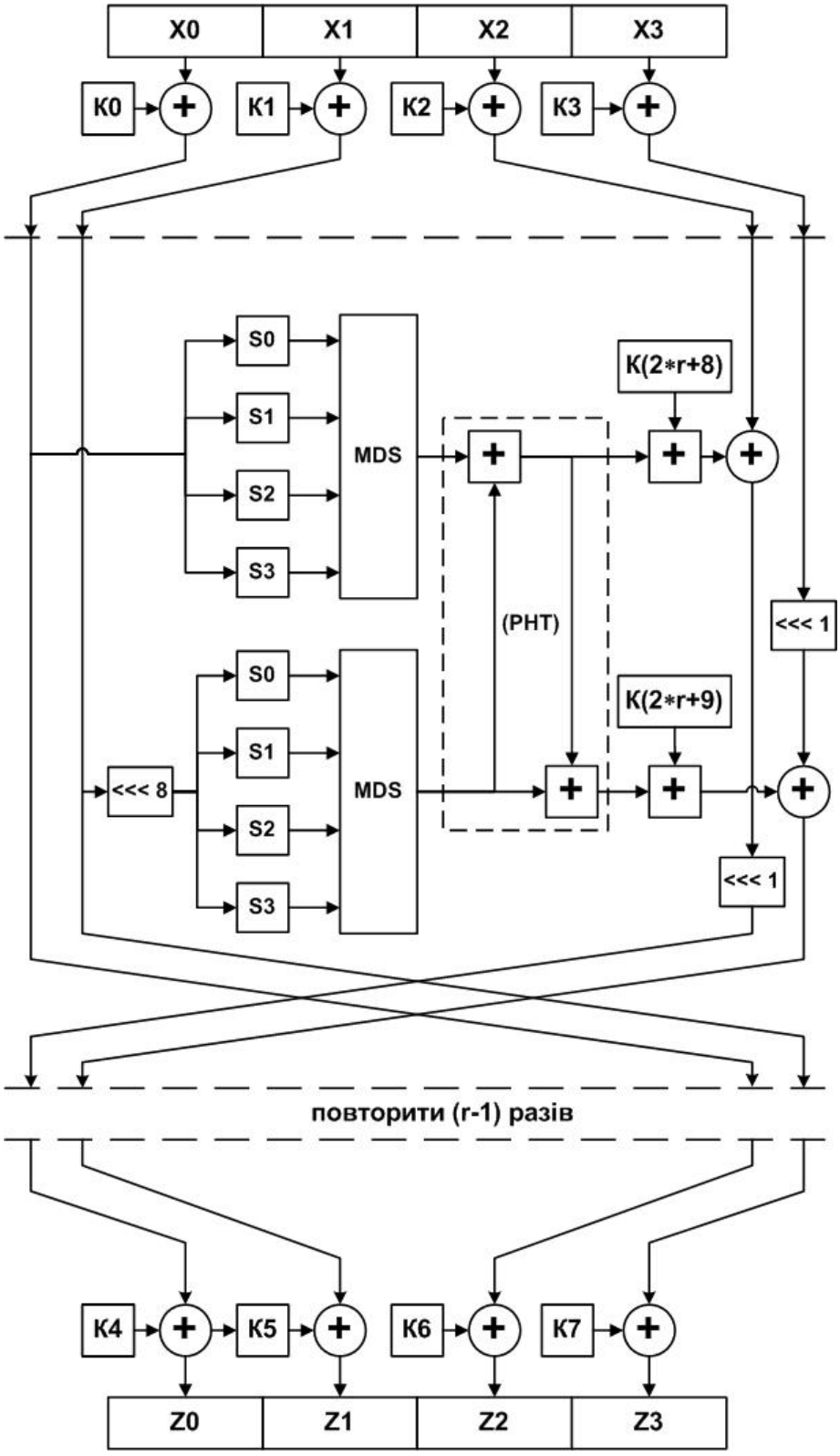


Рис.1.

Переможець AES – шифр Rijndael

Даний алгоритм розроблений двома фахівцями з криптографії з Бельгії. Він є нетрадиційним блоковим шифром, оскільки не використовує мережу Фейштеля для криптопреобразований. Алгоритм представляє кожний блок кодованих даних у вигляді двовимірного масиву байт розміром 4x4, 4x6 або 4x8 залежно від встановленої довжини блоку. Далі на відповідних етапах перетворення проводяться або над незалежними стовпцями, або над незалежними рядками, або взагалі над окремими байтами в таблиці.
Всі перетворення в шифрі мають строге математичне обгрунтовування. Сама структура і послідовність операцій дозволяють виконувати даний алгоритм ефективно як на 8-бітових так і на 32-бітових процесорах. В структурі алгоритму закладена можливість паралельного виконання деяких операцій, що на багатопроцесорних робочих станціях може ще підняти швидкість шифрування в 4 рази.
Алгоритм складається з деякої кількості раундів (від 10 до 14 – це залежить від розміру блоку і довжини ключа), в яких послідовно виконуються наступні операції :

- ByteSub – таблична підстановка 8x8 біт (рис.1),
-

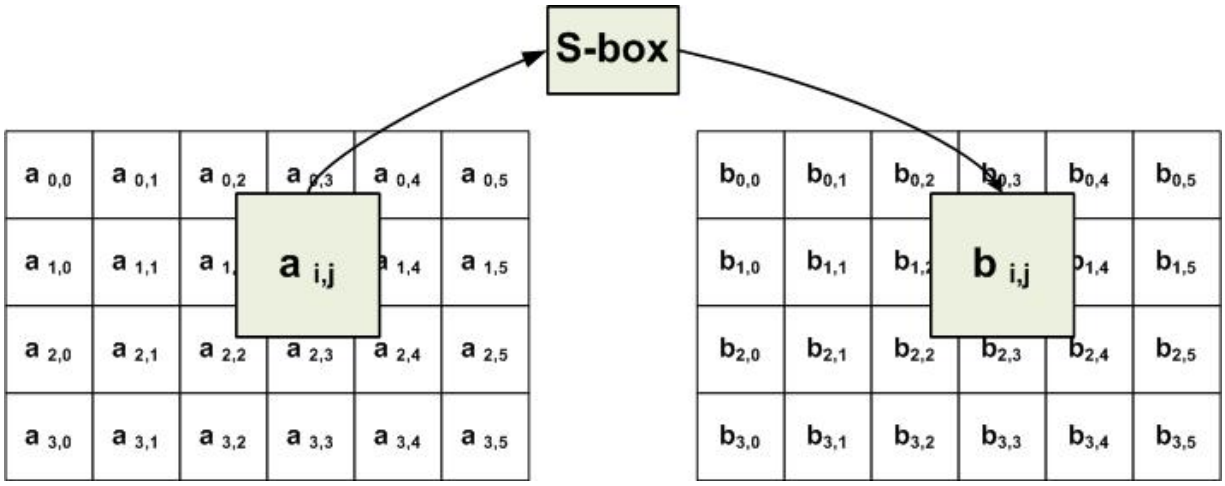


Рис.1.

- ShiftRow – зсув рядків в двовимірному масиві на різні зсуви (рис.2),

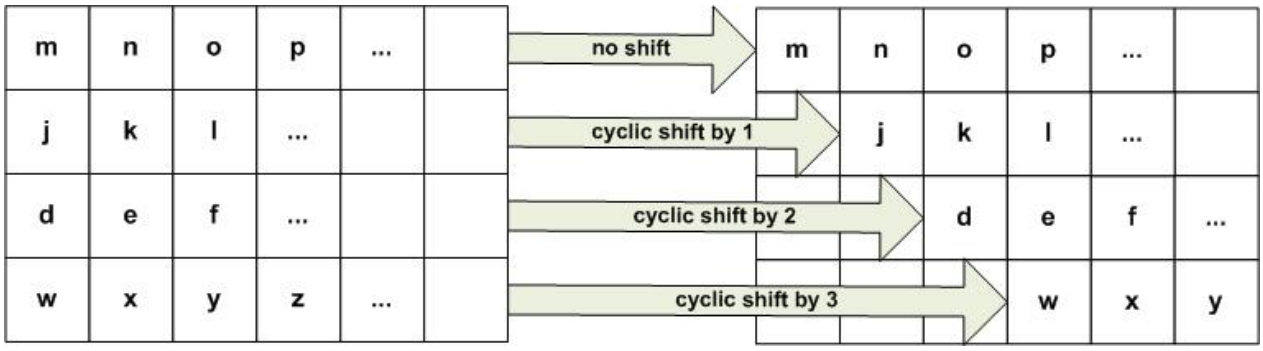


Рис.2.

- MixColumn – математичне перетворення, що перемішує дані усередині стовпця (рис.3),
-

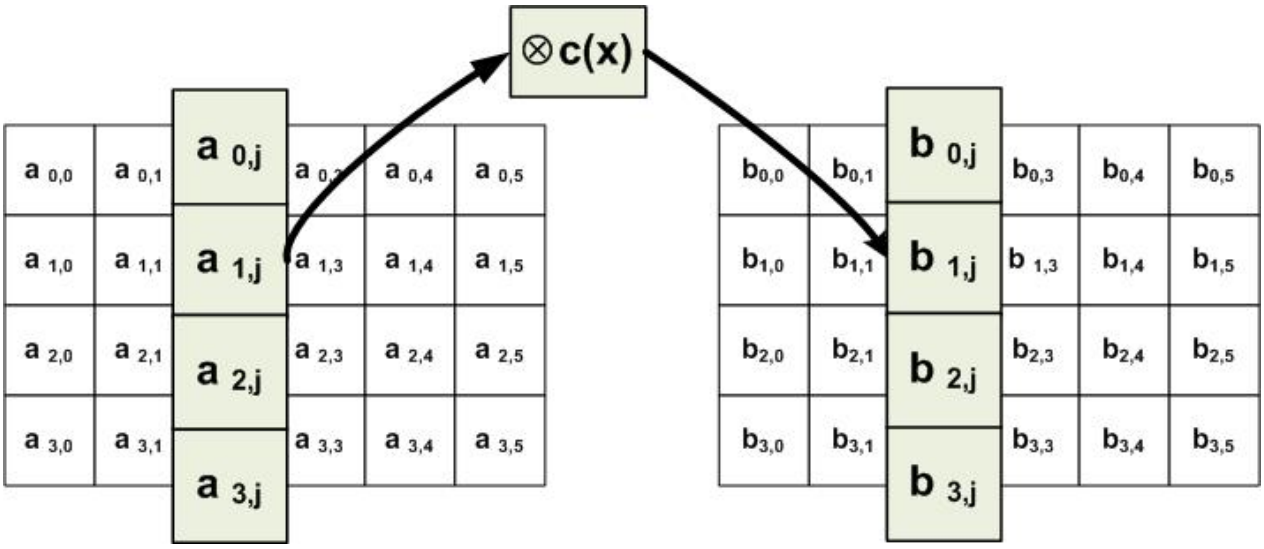


Рис.3.

- AddRoundKey – додавання матеріалу ключа операцією XOR (рис.4).
-

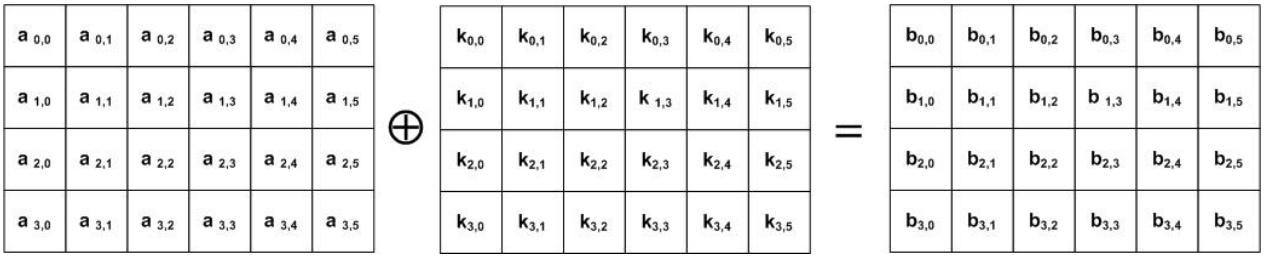


Рис.4.

У останньому раунді операція перемішування стовпців відсутня, що робить всю послідовність операцій симетричної.

Симетричні криптосистеми

2.3.1. Функции криптосистем

Короткий опис того, для чого все ж таки призначені криптосистеми.

2.3.2. Алгоритмы создания цепочек

Алгоритми створення ланцюжків покликані рандомізіровать вхідний потік криптоалгоритма, а також забезпечувати кратність його розміру довжині блоку криптоалгоритма.

2.3.3. Методы рандомизации сообщений

Для того, щоб одні і ті ж дані, передавані по мережі багатократно, шифрувалися кожного разу по-новому, необхідне внесення в процес шифрування по спеціальних схемах якої-небудь випадкової інформації.

2.3.4. Архивация

Архівація (стиснення даних) – є процес представлення інформації в іншому вигляді (перекодіровання) з потенційним зменшенням об'єму, що вимагається для її зберігання. Існує безліч класів різних алгоритмів стиснення даних, кожний з яких орієнтований на свою область вживання.

2.3.5. Хеширование паролей

Для того, щоб не примушувати людини запам'ятовувати ключ – довгу послідовність цифр, були розроблені методи перетворення рядка символів будь-якої довжини (так званого пароля) в блок байт наперед заданого розміру (ключ). На алгоритми, що використовуються в цих методах, накладаються вимоги, порівнянні з вимогами на самі криптоалгоритмы.

2.3.6. Транспортное кодирование

У деяких системах передачі інформації вимагається, щоб потік містив тільки певні символи ASCII кодування. Проте, вихідний потік криптоалгоритма має дуже високу рандомізацію і в ньому зустрічаються з рівною вірогідністю всі 256 символів. Для подолання цієї проблеми використовується транспортне кодування.

2.3.7. Общая схема симметричной криптосистемы

Схема об'єднання всіх розглянутих вище методів.

Функції криптосистем

Всі дослідження, які ми проводили на попередніх лекціях, торкалися тільки криптоалгоритмов, тобто методів перетворення невеликого блоку даних (від 4 до 32 байт) в закодований вигляд залежно від заданого двійкового ключа. Криптоалгоритмы поза сумнівом є "серцем" криптографічних систем, але, як ми зараз побачимо, їх безпосереднє вживання без яких-небудь модифікацій для кодування великих об'ємів даних насправді не дуже пріємлімо. Всі недоліки безпосереднього вживання криптоалгоритмов усуваються в криптосистемах. Криптосистема – це завершена комплексна модель, здатна проводити двосторонні криптопреобразования над даними довільного об'єму і підтверджувати час відправки повідомлення, володіюча механізмом перетворення паролів і ключів і системою транспортного кодування. Таким чином, криптосистема виконує три основні функції:

1. посилення захищеності даних,
2. полегшення роботи з криптоалгоритмом з боку людини
3. забезпечення сумісності потоку даних з іншим програмним забезпеченням.

Конкретна програмна реалізація криптосистеми називається криптопакетом.

Алгоритми створення ланцюжків

Перша задача, з якою ми зіткнемося при шифруванні даних криптоалгоритмом, – це дані з довжиною, нерівній довжині 1 блоку криптоалгоритма. Ця ситуація матиме місце практично завжди.

Перше питання:

– Що можна зробити, якщо ми хочемо зашифрувати 24 байти тексту, якщо використовується криптоалгоритм з довжиною блоку 8 байт?

– Послідовно зашифрувати три рази по 8 байт і скласти їх у вихідний файл так, як вони

лежали в початковому.
– А якщо даних багато і деякі блоки по 8 байт повторюються, це значить, що у вихідному файлі ці ж блоки будуть зашифровані однаково - це дуже погано.
Друге питання :
– А що коли даних не 24, а 21 байт.
Не шифрувати останні 5 байт або чимось заповнювати ще 3 байти, – а потім при дешифруванні їх викидати.
– Перший варіант взагалі нікуди не годиться, а другий застосовується, але ніж заповнювати ?
Для вирішення цих проблем і були введені в криптосистеми алгоритми створення ланцюжків (англ. chaining modes). Найпростіший метод ми вже у принципі описали. Це метод ECB (Electronic Code Book). Шифрований файл тимчасово розділяється на блоки, рівні блокам алгоритму, кожний з них шифрується незалежно, а потім із зашифрованих пакетів даних компонується в тій же послідовності файл, який відтепер надійно захищений криптоалгоритмом. Назву алгоритм одержав через те, що через свою простоту він широко застосовувався в простих портативних пристроях для шифрування – електронних шифрокніжках. Схема даного методу приведена на рис.1.

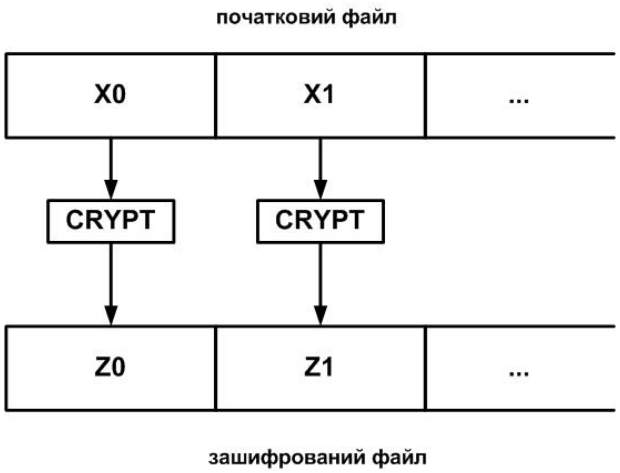


Рис.1.

У тому випадку, коли довжина пакету інформації, що пересилається, не кратна довжині блоку криптоалгоритма можливо розширення останнього (неповного) блоку байт до необхідної довжини або за допомогою генератора псевдовипадкових чисел, що не завжди безпечне відносно криптостойкости, або за допомогою хеш-суммы передаваного тексту. Другий варіант більш переважний, оскільки хеш-сума володіє кращими статистичними показниками, а її апріорна популярність стороннему особі рівносильна знанню ним всього передаваного тексту.
Вказаним вище недоліком цієї схеми є те, що при повторі в початковому тексті однакових символів протягом більш, ніж 2*N байтів (де N – розмір блоку криптоалгоритма), у вихідному файлі присутні однакові зашифровані блоки. Тому, для "могутнішого" захисту великих пакетів інформації за допомогою блокових шифрів застосовуються декілька оборотних схем "створення ланцюжків". Всі вони майже рівнозначні по криптостойкости, кожна має деякі переваги і недоліки, залежні від виду початкового тексту.
Всі схеми створення ланцюжків засновані на ідеї залежності результуючого зашифровуваного блоку від попередніх, або від позиції його в початковому файлі. Це досягається за допомогою блоку "пам'яті" – пакету інформації довжини, рівній довжині блоку алгоритму. Блок пам'яті (до нього застосовують термін IV – англ. Initial Vector) обчислюється за певним принципом зі всіх минулих шифрування блоків, а потім накладається за

допомогою якої-небудь оборотної функції (звичне XOR) на оброблюваний текст на одній із стадій шифрування. В процесі раскодірованія на приймальній стороні операція створення IV повторюється на основі прийнятого і розшифрованого тексту, унаслідок чого алгоритми створення ланцюжків повністю обратіми. Початковий файл зашифрований файл
Два найпоширеніших алгоритму створення ланцюжків – CBC і CFB. Їх структура приведена на рис.2 і рис.3. Метод CBC одержав назву від англійської абревіатури Cipher Block Chaining – об'єднання в ланцюжок блоків шифру, а метод CFB – від Cipher FeedBack – зворотний зв'язок по шифроблоку.

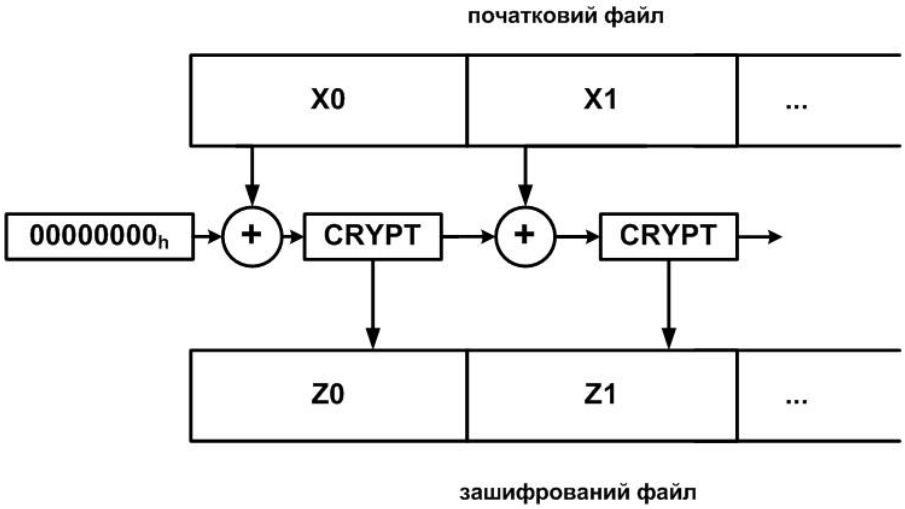


Рис.2.

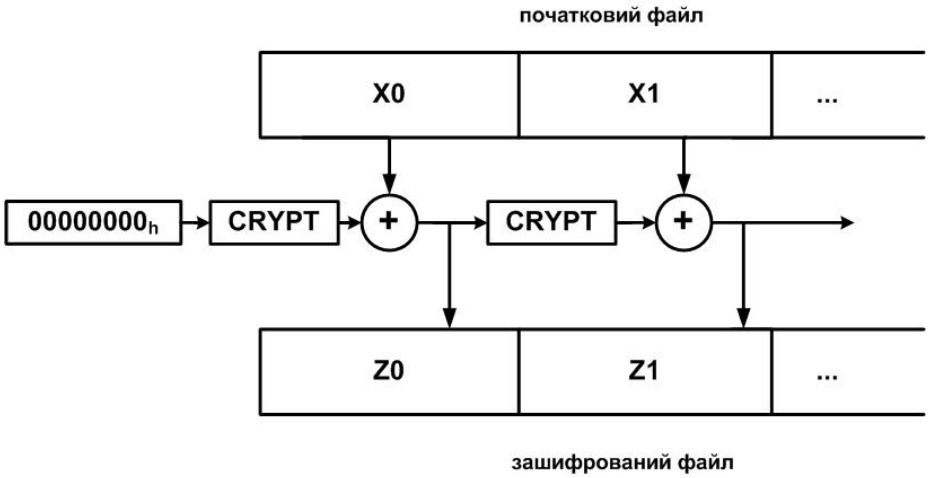


Рис.3.

Ще один метод OFB (англ. Output FeedBack – зворотний зв'язок по виходу) має дещо іншу структуру (вона зображена на рис.4.) : в ньому значення накладається на шифрований блок не залежить від попередніх блоків, а тільки від позиції шифрованого блоку (в цьому значенні він повністю відповідає скремблерам), і через це він не поширює перешкоди на подальші блоки. Очевидно, що всі алгоритми створення ланцюжків однозначно відновлені. Практичні алгоритми створення і декодування ланцюжків будуть розроблені на практичному занятті.

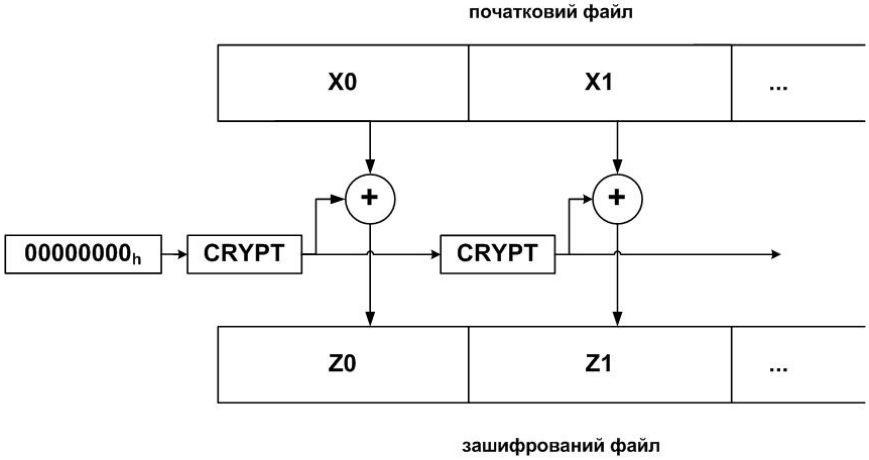


Рис.4.

Порівняємо характеристики методів створення ланцюжків у вигляді таблиці.

Метод	**Шифрування блоку залежить від**	**Спотворення одного біта при передачі**	**Чи кодується некратне блоку число байт без доповнення?**	**На вихід криптосистеми поступає**
ECB	поточного блоку	псує весь поточний блок	ні	вихід криптоалгоритма
CBC	всіх попередніх блоків	псує весь поточний і всі подальші блоки	ні	вихід криптоалгоритма
CFB	всіх попередніх блоків	псує один біт поточного блоку і всі подальші блоки	так	XOR маска з початковим текстом
OFB	позиції блоку у файлі	псує тільки один біт поточного блоку	так	XOR маска з початковим текстом

Методи рандомізації повідомлень

2.3.3.1. Обзор методик рандомизации сообщений

Дві основні методики внесення випадковості в процес шифрування представляють з себе : а) внесення випадкових біт в сам шифрований файл з ігноруванням їх на дешифрующей стороні, би) шифрування початкового файлу випадковим ключем.

2.3.3.2. Генераторы случайных и псевдослучайных последовательностей

Генератори випадкових послідовностей виконують велику роль в сучасній криптографії. У тому випадку, коли послідовність, що генерується, заснована тільки на стані ЕОМ, вона називається псевдовипадковою. Дійсно випадковими є тільки деякі фізичні процеси і людський чинник.

Огляд методик рандомізації повідомлень

Наступним удосконаленням, направленим на підвищення стійкості всієї системи в цілому є створення ключів сеансу. Ця операція необхідна в тих випадках, коли проводиться часте шифрування схожих блоків даних одним і тим же ключем. Наприклад, це має місце при передачі інформації або команд в автоматизованих системах управління, в банківських операціях і багатьох інших випадках передачі інформації, що має певний наперед відомий формат.
У цьому випадку необхідне введення якої-небудь випадкової величини в процес шифрування. Це можна зробити декількома способами:

1. записом в початок файлу даних псевдовипадкової послідовності байт наперед обумовленої довжини з відкиданням її при дешифруванії – цей метод працюватиме тільки при вживанні алгоритмів створення ланцюжків з пам'яттю (CBC,CFB,OFB),
2. вживанням модифікованих алгоритмів створення ланцюжків, які при шифруванні кожного блоку змішують з ним або а) фіксовану випадкову величину, прикріплену до початку зашифрованого файлу, або б) значення (значення), обчислювані за допомогою того ж шифру і ключа від наперед обумовленої величини,
3. створенням спеціально для кожного файлу абсолютно випадкового ключа, так званого ключа сеансу, яким і шифрується весь файл (сам же ключ сеансу шифрується первинним ключем, званим в цьому випадку майстер-ключем і поміщається на початку зашифрованого файлу).

Всі схеми принципово не мають очевидних недоліків, але через більшу опрацьованість останнього методу звичайно застосовується саме він.

Генератори випадкових і псевдовипадкових послідовностей

Найбільша проблема всіх методів рандомізації повідомлень – це породження дійсно випадкової послідовності біт. Річ у тому, що генератори випадкових послідовностей, що використовуються для загальних цілей, наприклад, в мовах програмування, є насправді псевдовипадковими генераторами. Річ у тому, що у принципі існує кінцева, а не нескінченна безліч станів ЕОМ, і, як би складно не формувалося в алгоритмі число, воно все одно має відносно небагато біт інформаційної насиченості.
Давайте розглянемо проблему створення випадкових і псевдовипадкових чисел детальніше. Найчастіше в прикладних задачах результат формують з лічильника тиків – системного годинника. В цьому випадку дані про поточну годину несуть приблизно 16 біт інформації, значення лічильника тиків – ще 16 біт. Це дає нам 32 біти інформації – як ви пам'ятаєте, на сьогоднішній день межею стійкої криптографії є значення в 40 біт, при реальних довжинах ключів в 128 біт. Природно, подібний метод украй недостатньо. Йдемо далі, до 32 біт можна додати ще 16 біт з надшвидкого таймера, що працює на частоті 1,2 Мгц в комп'ютерах архітектури IBM PC AT і цього ще недостатньо. Крім того, навіть якщо ми зможемо набрати довжину ключа в 128 біт (що дуже сумнівне), вона нестиме псевдовипадковий характер, оскільки заснована на стані тільки даної ЕОМ на момент початку шифрування. Джерелами по-справжньому випадкових величин можуть бути тільки зовнішні об'єкти, наприклад, людина.
Два найчастіше вживаних методу створення випадкових послідовностей за допомогою людини засновані на введенні з клавіатури. В обох випадках користувача просять, не замислюючись, понабирати на клавіатурі безглузді поєднання букв.

По першому методу над самими введеними значеннями проводяться дії, що підвищують випадковість вихідного потоку. Так, наприклад, обов'язково віддаляються верхні 3 біти введеного ASCII символу, часто віддаляються ще один верхня і ще одна нижня біти. Потім, об'єм одержаної послідовності зменшується ще в три рази накладенням першого і другого біта, на третій, операцією XOR. Це, у принципі, генерує достатньо випадкову послідовність біт.
По другому методу на введені символи алгоритм не звертає ніякої уваги, зате конспектує інтервали часу, через які відбулися натиснення. Запис моментів проводиться по відліках швидкого системного таймера (частота 1,2 Мгц) або внутрішньому лічильнику процесора, що з'явився в процесорах, починаючи з Intel Pentium (частота відповідає частоті процесора). Оскільки верхні і молодші біти мають певну кореляцію між символами (перші через фізичні характеристики людини, другі через особливості операційної системи), то вони відкидаються (звичайно віддаляються 0-8 старших біта і 4-10 молодших).
Як варіанти, що більш рідко зустрічаються, можна зустріти 1) комбінацію обох клавіатурних методів і 2) метод, заснований на маніпуляторі "миша" - він виділяє випадкову інформацію із зсувів користувачем покажчика миші.
У могутніх криптосистемах військового вживання використовуються дійсно випадкові генератори чисел, засновані на фізичних процесах. Вони є платнею, або зовнішніми пристроями, що підключаються до ЕОМ через порт введення-висновку. Два основні джерела білого Гауссовського шуму – високоточне вимірювання теплових флуктуацій і запис радіоефіру на частоті, вільній від радіомовлення.

Архівація

2.3.4.1. Общие принципы архивации. Классификация методов
Існують дві великі групи алгоритмів архівації : стиснення без втрат бієктівно перекодує інформацію по інших законах, тобто можливе абсолютно ідентичне її відновлення; стиснення з втратами необоротно видаляє з інформації деякі відомості, що роблять якнайменший вплив на значення повідомлення.
2.3.4.2. Алгоритм Хаффмана
Алгоритм стиснення орієнтований на неосмислені послідовності символів якого-небудь алфавіту. Необхідною умовою для стиснення є різна вірогідність появи цих символів (і ніж відмінність у вірогідності ощутімєє, тим більше ступінь стиснення).
2.3.4.3. Алгоритм Лемпеля-Зива
А цей алгоритм стиснення заснований навпаки на кореляціях між розташованими поряд символами алфавіту (словами, що управляють послідовностями, заголовками файлів фіксованої структури)

Загальні принципи архівації. Класифікація методів

Наступною великою темою є архівація даних. Як Вам відомо, пригнічуюче більшість сучасних форматів запису даних містять їх у вигляді, зручному для швидкого маніпулювання, для зручного прочитання користувачами. При цьому дані займають об'єм більший, ніж це дійсно потрібне для їх зберігання. Алгоритми, які усувають надмірність запису даних, називаються алгоритмами стиснення даних, або алгоритмами архівації. В даний час існує величезна безліч програм для стиснення даних, заснованих на декількох основних способах.

Навіщо ж потрібна архівація в криптографії? Річ у тому, що в сучасному криптоаналізі, тобто науці про протистояння криптографії, з очевидністю доведено, що вірогідність злому криптосхемы за наявності кореляції між блоками вхідної інформації значно вище, ніж за відсутності такої. А алгоритми стиснення даних за визначенням і мають своєю основною задачею усунення надмірності, тобто кореляцій між даними у вхідному тексті.
Всі алгоритми стиснення даних якісно діляться на 1) алгоритми стиснення без втрат, при використовуванні яких дані на приймальні відновлюються без щонайменших змін, і 2) алгоритми стиснення з втратами, які видаляють з потоку даних інформацію, що трохи впливає на суть даних, або взагалі несприйману людиною (такі алгоритми зараз розроблені тільки для аудіо- і відео- зображень). В криптосистемах, природно, використовується тільки перша група алгоритмів.
Існує два основні методи архівації без втрат:

- алгоритм Хаффмана (англ. Huffman), орієнтований на стиснення послідовностей байт, не зв'язаних між собою,
- алгоритм Лемпеля-Зіва (англ. Lempel, Ziv), орієнтований на стиснення будь-яких видів текстів, тобто використовуючий факт неодноразового повторення "слів" – послідовностей байт.

Практично всі популярні програми архівації без втрат (ARJ, RAR, ZIP і т.п.) використовують об'єднання цих двох методів – алгоритм LZH.

Алгоритм Хаффмана

Алгоритм заснований на тому факті, що деякі символи із стандартного 256-символьного набору в довільному тексті можуть зустрічатися частіше за середній період повтору, а інші, відповідно, – рідше. Отже, якщо для запису поширених символів використовувати короткі послідовності біт, завдовжки менше 8, а для запису рідкісних символів – довгі, то сумарний об'єм файлу зменшиться.
Хаффман запропонував дуже простий алгоритм визначення того, який символ необхідно кодувати яким кодом для отримання файлу з довжиною, дуже близькою до його ентропії (тобто інформаційної насиченості). Хай у нас є список всіх символів, що зустрічаються в початковому тексті, причому відома кількість появ кожного символу в ньому. Випишемо їх вертикально в ряд у вигляді осередків майбутнього графа по правому краю листу (мал. 1а). Виберемо два символи з якнайменшою кількістю повторень в тексті (якщо три або більше число символів мають однакові значення, вибираємо будь-кого два з них). Проведемо від них лінії вліво до нової вершини графа і запишемо в неї значення, рівне сумі частот повторення кожного з об'єднуваних символів (рис.2б). Відтепер не братимемо до уваги при пошуку якнайменших частот повторення два об'єднані вузли (для цього зітремо числа в цих двох вершинах), але розглядатимемо нову вершину як повноцінний осередок з частотою появи, рівній сумі частот появи двох вершин, що з'єдналися. Повторюватимемо операцію об'єднання вершин до тих пір, поки не дійдемо однієї вершини з числом (рис.2в і 2г). Для перевірки: очевидно, що в ній буде записана довжина кодованого файлу. Тепер розставимо на двох ребрах графа, витікаючих з кожної вершини, біти 0 і 1 довільно – наприклад, на кожному верхньому ребрі 0, а на кожному нижньому – 1. Тепер для визначення коду кожної конкретної букви необхідно просто пройти від вершини дерева до неї, виписуючи нулі і одиниці по маршруту проходження. Для малюнка 4.5 символ "А" одержує код "000", символ "Б" – код "01", символ "К" – код "001", а символ "О" – код "1".

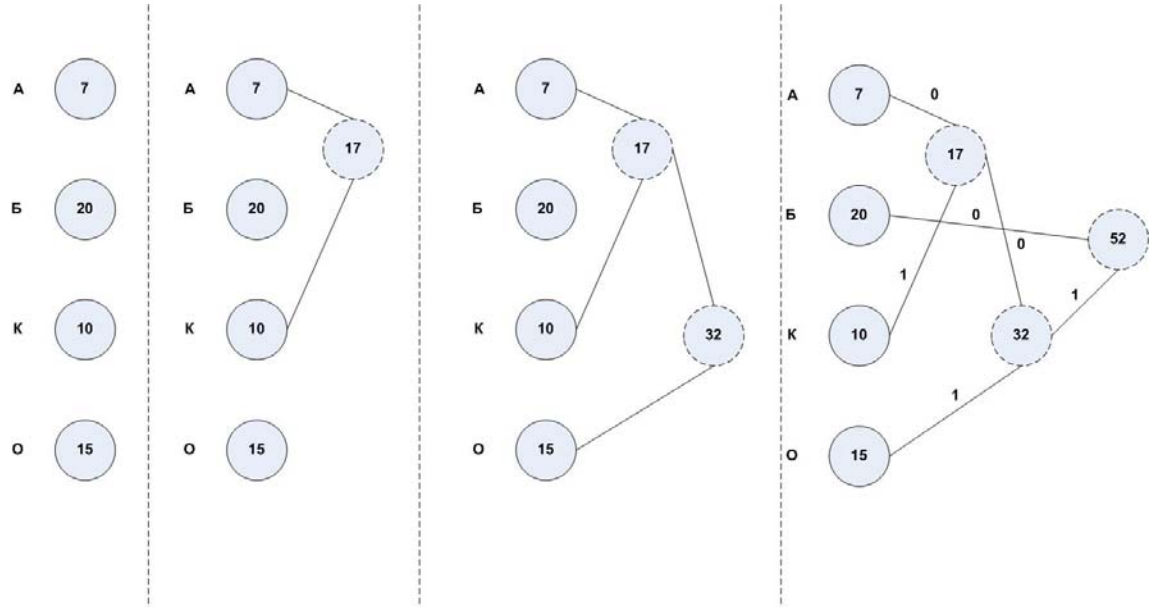


Рис.1.

У теорії кодування інформації показується, що код Хаффмана є префіксним, тобто код ніякого символу не є початком коду якого-небудь іншого символу. Перевірте це на нашому прикладі. А з цього виходить, що код Хаффмана однозначно відновлений одержувачем, навіть якщо не повідомляється довжина коду кожного переданого символу. Одержувачу пересилають тільки дерево Хаффмана в компактному вигляді, а потім вхідна послідовність кодів символів декодується їм самостійно без якої-небудь додаткової інформації. Наприклад, при прийомі "0100010100001" їм спочатку відділяється перший символ "Би" : "01-00010100001", потім знову починаючи з вершини дерева – "А" "01-000-10100001", потім аналогічно декодується весь запис "01-000-1-01-000-01" "БАОБАБ".

Алгоритм Лемпеля-Зіва

Класичний алгоритм Лемпеля-Зіва – LZ77, названий так по року своєї публікації, гранично простий. Він формулюється таким чином : "якщо в минулому раніше вихідному потоці вже зустрічалася подібна послідовність байт, причому запис про її довжину і зсув від поточної позиції коротший ніж сама ця послідовність, то у вихідний файл записується посилання (зсув, довжина), а не сама послідовність". Так фраза "КОЛОКОЛ_ОКОЛО_КОЛОКОЛЬНІ" закодується як "КОЛО(-4,3)_(-5,4)О_(-14,7) ЬНІ".

Поширений метод стиснення RLE (англ. Run Length Encoding), який полягає в записі замість послідовності однакових символів одного символу і їх кількості, є підкласом даного алгоритму. Розглянемо, наприклад, послідовність "АААААА". За допомогою алгоритму RLE вона буде закодована як "(А,7)", в той же час її можна достатньо добре стиснути і за допомогою алгоритму LZ77 : "А(-1,6)". Дійсно, ступінь стиснення саме такої послідовності їм гірший (приблизно на 30-40%), але сам по собі алгоритм LZ77 більш універсальний, і може набагато краще обробляти послідовності взагалі нестискувані методом RLE.

–

Хешування паролів

Від методів, що підвищують криптостойкость системи в цілому, перейдемо до блоку хешування паролів – методу, що дозволяє користувачам запам'ятовувати не 128 байт, тобто 256 шістнадцяткових цифр ключа, а деякий осмислений вираз, слово або послідовність символів, що називається паролем. Дійсно, при розробці будь-якого криптоалгоритма слід враховувати, що в половині випадків кінцевим користувачем системи є людина, а не автоматична система. Це ставить питання про те, зручно, і чи взагалі реально людині запам'ятати 128-бітовий ключ (32 шістнадцяткові цифри). Насправді межа запомінаємості лежить на межі 8-12 подібних символів, а, отже, якщо ми примушуватимемо користувача оперувати саме ключем, тим самим ми практично вимусимо його до запису ключа на якому-небудь листку паперу або електронному носії, наприклад, в текстовому файлі. Це, природно, різко знижує захищеність системи.
Для вирішення цієї проблеми були розроблені методи, що перетворюють вимовний, осмислений рядок довільної довжини – пароль, у вказаний ключ наперед заданої довжини. В переважній більшості випадків для цієї операції використовуються так звані хеш-функции (від англ. hashing – дрібна нарізка і перемішування). Хеш-функцией називається таке математичне або алгоритмічне перетворення заданого блоку даних, яке володіє наступними властивостями:

1. хеш-функція має нескінченну область визначення,
2. хеш-функція має кінцеву область значень,
3. вона необратіма,
4. зміна вхідного потоку інформації на один біт міняє близько половини всіх біт вихідного потоку, тобто результату хеш-функции.

Ці властивості дозволяють подавати на вхід хеш-функции паролі, тобто текстові рядки довільної довжини на будь-якій національній мові і, обмеживши область значень функції діапазоном 0..2N-1, де N – довжина ключа в бітах, набувати на виході достатньо рівномірно розподілені по області значення блоки інформації – ключі.
Неважко помітити, що вимоги, подібні 3 і 4 пунктам вимог до хеш-функции, виконують блокові шифри. Це указує на один з можливих шляхів реалізації стійких хеш-функций – проведення блокових криптопреобразований над матеріалом рядка-пароля. Цей метод і використовується в різних варіаціях практично у всіх сучасних криптосистемах. Матеріал рядка-пароля багато разів послідовно використовується як ключ для шифрування деякого наперед відомого блоку даних – на виході виходить зашифрований блок інформації, однозначно залежний тільки від пароля і при цьому має достатньо хороші статистичні характеристики. Такий блок або декілька таких блоків і використовуються як ключ для подальших криптопреобразований.
Характер вживання блокового шифру для хешування визначається відношенням розміру блоку криптоалгоритма і розрядності необхідного хеш-результата, що використовується. Якщо вказані вище величини співпадають, то використовується схема одноцепочечного блокового шифрування. Первинне значення хеш-результата H0 встановлюється рівним 0, весь рядок-пароль розбивається на блоки байт, рівні по довжині ключу блокового шифру, що використовується для хешування, потім проводяться перетворення по реккурентной формулі:
$H_j = H_{j-1}$ XOR EnCrypt(Hj-1,PSWj),
де EnCrypt(X,Key) – блоковий шифр, що використовується (рис.1).
Останнє значення Hk використовується як шуканий результат.

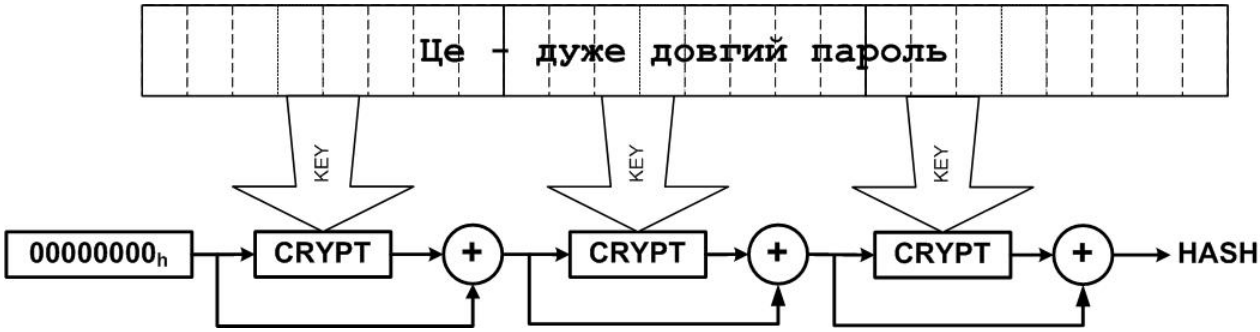


Рис.1.

У тому випадку, коли довжина ключа рівно в два рази перевершує довжину блоку, а подібна залежність досить часто зустрічається в блокових шифрах, використовується схема, що нагадує мережу Фейштеля. Характерним недоліком і приведеної вище формули, і хеш-функции, заснованої на мережі Фейштеля, є велика ресурсоємкость відносно пароля. Для проведення тільки одного перетворення, наприклад, блоковим шифром з ключем завдовжки 128 битий використовується 16 байт рядка-пароля, а сама довжина пароля рідко перевищує 32 символи. Отже, при обчисленні хеш-функции над паролем будуть проведено максимум 2 "повноцінні" криптопреобразования.

Рішення цієї проблеми можна досягти двома шляхами : 1) заздалегідь "розмножити" рядок-пароль, наприклад, записавши її багато разів послідовно до досягнення довжини, скажімо, в 256 символів; 2) модифікувати схему використовування криптоалгоритма так, щоб матеріал рядка-пароля "повільніше" витрачався при обчисленні ключа.

По другому шляху пішли дослідники Девіс і Майер, що запропонували алгоритм також на основі блокового шифру, але використовуючий матеріал рядка-пароля багатократно і невеликими порціями. В ньому є видимим елементи обох приведених вище схем, але криптостойкость цього алгоритму підтверджена численними реалізаціями в різних криптосистемах. Алгоритм одержав назву "Tandem DM" (рис.2):

```
G0=0; H0=0 ;
FOR J = 1 TO N DO
  BEGIN
  TMP=EnCrypt(H,[G,PSWj]); H'=H XOR TMP;
  TMP=EnCrypt(G,[PSWj,TMP]); G'=G XOR TMP;
  END;
Key=[Gk,Hk]
```

Квадратними дужками (X16=[A8,B8]) тут позначено просте об'єднання (склеювання) двох блоків інформації рівної величини в один – подвоєної розрядності. А як процедура EnCrypt(X,Key) знову може бути вибраний будь-який стійкий блоковий шифр. Як видно з формул, даний алгоритм орієнтований на те, що довжина ключа двократно перевищує розмір блоку криптоалгоритма. А характерною особливістю схеми є той факт, що рядок пароля прочитується блоками по половині довжини ключа, і кожний блок використовується в створенні хеш-результата двічі. Таким чином, при довжині пароля в 20 символів і необхідність створення 128 бітового ключа внутрішній цикл хеш-функции повториться 3 рази.

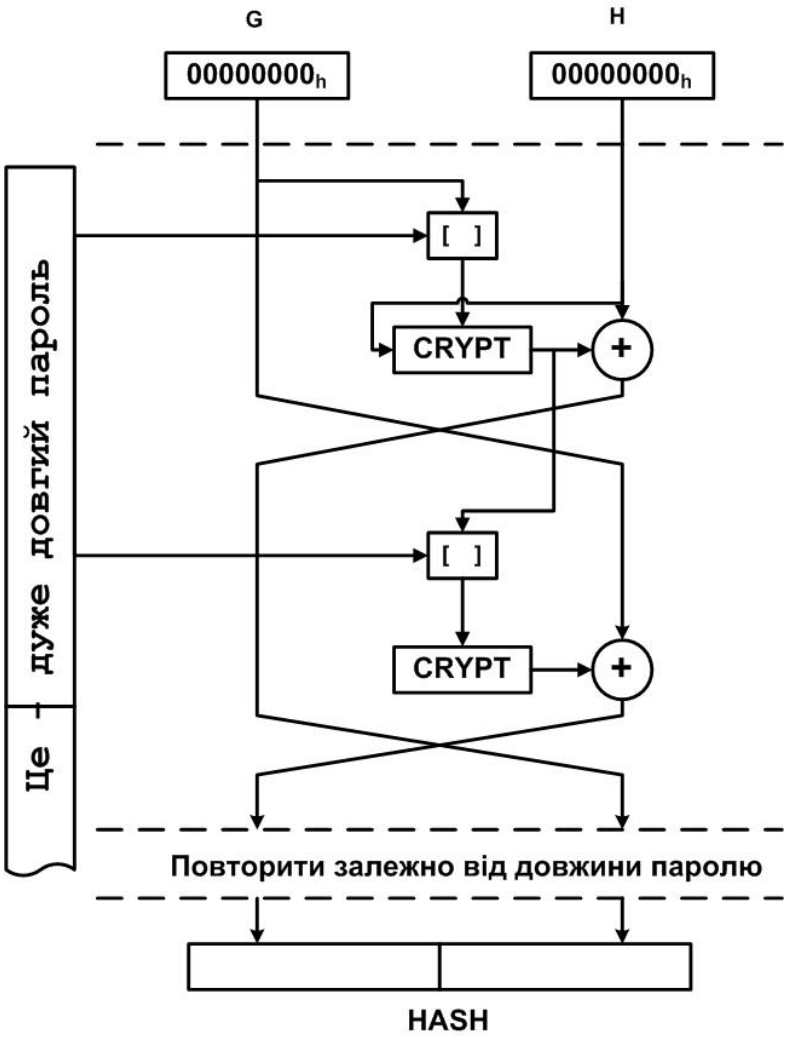


Рис.2.

Транспортне кодування

Оскільки системи шифрування даних часто використовуються для кодування текстової інформації : листування, рахунків, платежів електронної комерції, і при цьому криптосистема повинна бути абсолютно прозорою для користувача, то над вихідним потоком криптосистеми часто проводиться транспортне кодування, тобто додаткове кодування (не шифрування !) інформації виняткове для забезпечення сумісності з протоколами передачі даних.

Вся річ у тому, що на виході криптосистеми байт може приймати всі 256 можливих значень, незалежно від того чи був вхідний потік текстовою інформацією чи ні. А при передачі поштових повідомлень багато систем орієнтовано на те, що допустимі значення байтів тексту лежать у вужчому діапазоні : всі цифри, розділові знаки, алфавіт латиниці плюс, можливо, національної мови. Перші 32 символи набору ASCII служать для спеціальних цілей. Для того, щоб вони і деякі інші службові символи ніколи не з'явилися у вихідному потоці використовується транспортне кодування.

Найпростіший метод полягає в записі кожного байта двома шістнадцятковими цифрами-символами. Так байт 252 буде записаний двома символами 'FC'; байт з кодом 26, потрапляючий на спеціальний символ CTRL-Z, буде записаний двома допустимими символами '1A'. Але ця схема дуже надмірна : в одному байті передається тільки 4 біти інформації.
Насправді практично в будь-якій системі комунікації без проблем можна передавати близько 68 символів (латинський алфавіт рядковий і прописний, цифри і розділові знаки). З цього виходить, що цілком реально створити систему з передачею 6 біт в одному байті (26<68), тобто кодувати 3 байти довільного змісту 4-мя байтами з виключно дозволених (так званих друкарських) символів. Подібна система була розроблена і стандартизована на рівні протоколів мережі Інтернет – це система Base64 (стандарт RFC1251).
Процес кодування перетворить 4 вхідні символи у вигляді 24-бітової групи, обробляючи їх зліва направо. Ці групи потім розглядаються як 4 сполучені 6-бітові групи, кожна з яких транслюється в одиночну цифру алфавіту base64. При кодуванні base64 вхідний потік байтів повинен бути впорядкований старшими бітами вперед.
Кожна 6-бітова група використовується як індекс для масиву 64-х друкарських символів. Символ, на який указує значення індексу, поміщається у вихідний рядок. Ці символи вибрані так, щоб бути універсально уявними і виключають символи, що мають спеціальне значення (".", CR, LF).

Алфавит Base64

Значение	Код	Значение	Код	Значение	Код	Значение	Код
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v	заполнитель	=
14	O	31	f	48	w		
15	P	32	g	49	x		
16	Q	33	h	50	y		

Вихідний потік (закодовані байти) повинен мати довжину рядків не більше 76 символів. Всі ознаки перекладу рядка і інші символи, відсутні в таблиці 1, повинні бути проігноровані декодером base64. Серед даних в Base64 символи, не перераховані в табл. 1, переклади рядка і т.п. повинні говорити про помилку передачі даних, і, відповідно, програма-декодер повинна оповістити користувача про неї.
Якщо в хвості потоку кодованих даних залишилося менше, ніж 24 біти, справа додаються нульові біти до утворення цілого числа 6-бітових груп. А до кінця 24-бітової групи може залишатися тільки від 0 до 3-х не дістаючих 6-бітових груп, замість кожної з яких ставиться символ-заповнювач "=". Оскільки весь вхідний потік є цілим числом 8-бітових груп (тобто, просто байтних значень), то можливі лише наступні випадки:

1. Вхідний потік закінчується рівно 24-бітовою групою (довжина файлу кратна 3). У такому разі вихідний потік закінчуватиметься чотирма символами Base64 без яких або додаткових символів.

2. "Хвіст" вхідного потоку має довжину 8 біт. Тоді в кінці вихідного коду будуть два символи Base64, з додаванням двох символів "=".
3. "Хвіст" вхідного потоку має довжину 16 біт. Тоді в кінці вихідного стоятимуть три символи Base64 і один символ "=".

Оскільки символ "=" є хвостовим заповнювачем, його поява в тілі листу може означати тільки те, що кінець даних досягнутий. Але спиратися на пошук символу "=" для виявлення кінця файлу невірно, оскільки, якщо число переданих бітів кратне 24, то у вихідному файлі не з'явиться жодного символу "="

Загальна схема симетричної криптосистеми

Загальна схема симетричної криптосистеми з урахуванням всіх розглянутих пунктів зображена на малюнку 1.

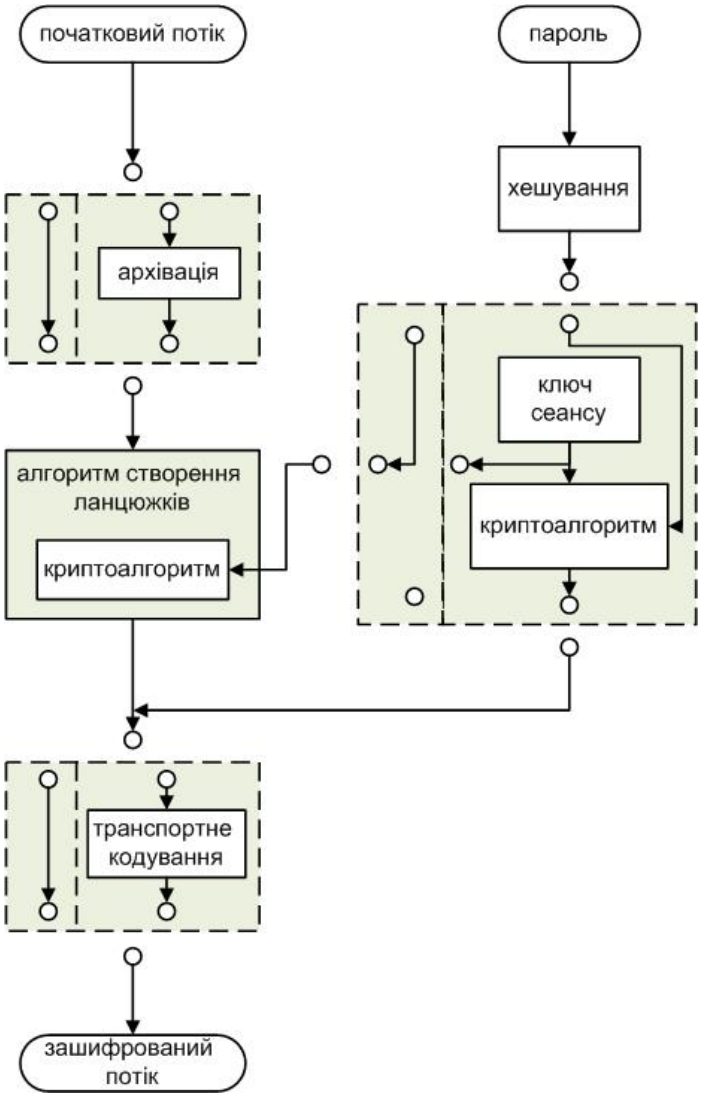


Рис.1.

Асиметричні криптоалгоритмы

2.4.1. Общие сведения об асимметричных криптоалгоритмах
Кожний користувач асиметричної криптосистеми заздалегідь створює по певному алгоритму пару ключів : закритий і відкритий – вони надалі використовуватимуться для відправки листів саме йому. Для відправки листу іншому абоненту мережі необхідно буде скористатися саме його відкритим ключем.
2.4.2. Алгоритм RSA
Алгоріт RSA є класикою асиметричної криптографії. В ньому як необоротне перетворення відправки використовується зведення цілих чисел у великі ступені по модулю.
2.4.3. Технологии цифровых подписей
Асиметрична криптографія, як виявилося, дозволяє дуже красиво вирішувати і задачу аутентифікації автора повідомлення – простою зміною порядку використовування відкритого і закритого ключів.
2.4.4. Механизм распространения открытых ключей
Асиметрична криптографія зробила ще і достатньо могутній прорив в технології первинного розповсюдження ключів. Якщо для симетричних криптосистем обов'язковим був попередній обмін по закритому каналу (звичайно особисто з рук в руки), то тепер з'явилися абсолютно нові способи для цього.
2.4.5. Обмен ключами по алгоритму Диффи-Хеллмана
Метод Діффі-Хеллмана використовує алгоритм, подібний алгоритму RSA, для первинного обміну ключами в симетричних криптосистемах по відкритому каналу, але тільки такому, в якому неможлива фальсифікація повідомлень.

Загальні відомості про асиметричні криптоалгоритмах

Симетричні криптосистеми, розглянуті нами в попередніх розділах, не дивлячись на безліч переваг, володіють одним серйозним недоліком, про який Ви, напевно, ще не замислювалися. Зв'язаний він з ситуацією, коли спілкування між собою проводять не три-чотири люди, а сотні і тисячі людей. В цьому випадку для кожної пари, що переписується між собою, необхідно створювати свій секретний симетричний ключ. Це у результаті приводить до існування в системі з N користувачів N2/2 ключів. А це вже дуже "пристойне" число. Крім того, при порушенні конфіденційності якої-небудь робочої станції зловмисник дістає доступ до всіх ключів цього користувача і може відправляти, нібито від його імені, повідомлення всім абонентам, з якими "жертва" вела листування.
Своєбразним рішенням цієї проблеми з'явилася поява асиметричної криптографії. Ця область криптографії дуже молода в порівнянні з іншими представниками. Перша схема, що мала прикладну значущість, була запропонована всього близько 20 років тому. Але за цей час асиметрична криптографія перетворилася на один з основних напрямів кріптологиі, і використовується в сучасному світі також часто, як і симетричні схеми.
Асиметрична криптографія спочатку задумана як засіб передачі повідомлень від одного об'єкту до іншого (а не для конфіденційного зберігання інформації, яке забезпечують тільки симетричні алгоритми). Тому подальше пояснення ми вестимемо в термінах "відправник" – особа, шифруюшєє, а потім отпраляющєє інформацію по незахищеному каналу і "одержувач" – особа, що приймає і поновлююче інформацію в її початковому вигляді. Основна ідея асиметричних криптоалгоритмов полягає в тому, що для шифрування повідомлення використовується один ключ, а при дешифрованії – інший.

Крім того, процедура шифрування вибрана так, що вона необратіма навіть по відомому ключу шифрування – це друга необхідна умова асиметричної криптографії. Тобто, знаючи ключ шифрування і зашифрований текст, неможливо відновити початкове повідомлення – прочитати його можна тільки за допомогою другого ключа – ключа дешифрованія. А раз так, то ключ шифрування для відправки листів якій-небудь особі можна взагалі не приховувати – знаючи його все одно неможливо прочитати зашифроване повідомлення. Тому, ключ шифрування називають в асиметричних системах "відкритим ключем", а ось ключ дешифрованія одержувачу повідомлень необхідно тримати в секреті – він називається "закритим ключем". Напрошується питання : "Чому, знаючи відкритий ключ, не можна обчислити закритий ключ ?" – це третя необхідна умова асиметричної криптографії – алгоритми шифрування і дешифрованія створюються так, щоб знаючи відкритий ключ, неможливо обчислити закритий ключ.
У цілому система листування при використовуванні асиметричного шифрування виглядає таким чином. Для кожного з N абонентів, що ведуть листування, вибрана своя пара ключів : "відкритий" Ej і "закритий" Dj, де j – номер абонента. Всі відкриті ключі відомі всім користувачам мережі, кожний закритий ключ, навпаки, бережеться тільки у того абонента, якому він належить. Якщо абонент, скажемо під номером 7, збирається передати інформацію абоненту під номером 9, він шифрує дані ключем шифрування E9 і відправляє її абоненту 9. Не дивлячись на те, що всі користувачі мережі знають ключ E9 і, можливо, мають доступ до каналу, по якому йде зашифроване послання, вони не можуть прочитати початковий текст, оскільки процедура шифрування необратіма по відкритому ключу. І лише абонент №9, одержавши послання, проводить над ним перетворення за допомогою відомого тільки йому ключа D9 і відновлює текст послання. Помітьте, що якщо повідомлення потрібно відправити в протилежному напрямі (від абонента 9 до абонента 7), то потрібно буде використовувати вже іншу пару ключів (для шифрування ключ E7, а для дешифрованія – ключ D7).
_Як ми бачимо, по-перше, в асиметричних системах кількість існуючих ключів пов'язана з кількістю абонентів лінійно (в системі з N користувачів використовуються 2*N ключів), а не квадратична, як в симетричних системах. По-друге, при порушенні конфіденційності до-ой робочої станції зловмисник взнає тільки ключ Dk : це дозволяє йому читати всі повідомлення, що приходять абоненту до, але не дозволяє вивадавать себе за нього при відправці листів. Окрім цього, асиметричні криптосистеми володіють ще декількома дуже цікавими можливостями, які ми розглянемо через декілька розділів.

Алгоритм RSA

Алгоритм RSA стоїть біля витоків асиметричної криптографії. Він був запропонований трьома исследователями-математиками Рональдом Рівестом (R.Rivest), Аді Шаміром (A.Shamir) і Леонардом Адльманом (L.Adleman) в 1977-78 роках.
Першим етапом будь-якого асиметричного алгоритму є створення пари ключів : відкритого і закритого і розповсюдження відкритого ключа "по всьому світу". Для алгоритму RSA етап створення ключів складається з наступних операцій :

1. Вибираються два прості (!) числа p і q
2. Обчислюється їх твір n(=p*q)
3. Вибирається довільне число e (e<n), таке, що НОД(e,(p-1)(q-1))=1, тобто e повинне бути взаємно простим з числом (p-1)(q-1).
4. Методом Евкліда розв'язується в цілих числах (!) рівняння e*d+(p-1)(q-1)*y=1. Тут невідомими є змінні d і y – метод Евкліда якраз і знаходить безліч пар (d,y), кожна з яких є рішенням рівняння в цілих числах.

5. Два числа (e,n) – публікуються як відкритий ключ.
6. Число d бережеться в строгому секреті – це і є закритий ключ, який дозволить читати всі послання, зашифровані за допомогою пари чисел (e,n).

Як же проводиться власне шифрування за допомогою цих чисел :

1. Відправник розбиває своє повідомлення на блоки, рівні $k=[\log 2_{(n)}]$ битий, де квадратні дужки позначають узяття цілої частини від дробового числа.
2. Подібний блок, як Ви знаєте, може бути інтерпретований як число з діапазону (0;2k-1). Для кожного такого числа (назвемо його mi) обчислюється вираз ci=((mi)e) mod n. Блоки ci і є зашифроване повідомлення Їх можна спокійно передавати по відкритому каналу, поскольку.операция зведення в ступінь по модулю простого числа, є необоротною математичною задачею. Зворотна їй задача носить назву "Логарифмування в кінцевому полі" і є на декілька порядків складнішою задачею. Тобто навіть якщо зловмисник знає числа e і n, то по ci прочитати початкові повідомлення mi він не може ніяк, окрім як повним перебором mi.

А ось на приймальній стороні процес дешифрованія все ж таки можливий, і допоможе нам в цьому число, в таємниці якого зберігається, d. Достатньо давно була доведена теорема Ейлера, окремий випадок якої утверждает, що якщо число n уявно у вигляді двох простих чисел p і q, то для будь-кого x має місце рівність (x(p-1)(q-1)) mod n = 1. Для дешифрованія RSA-сообщений скористаємося цією формулою. Піднесемо обидві її частини до ступеня (-y) : (x(-y)(p-1)(q-1))mod n = 1(-y)= 1. Тепер помножимо обидві її частини на x : (x(-y)(p-1)(q-1)+1)mod n = 1*x = x.
А зараз пригадаємо як ми створювали відкритий і закритий ключі. Ми підбирали за допомогою алгоритму Евкліда d таке, що e*d+(p-1)(q-1)*y=1, тобто e*d=(-y)(p-1)(q-1)+1. А отже в останньому виразі попереднього абзацу ми можемо замінити показник ступеня на число (e*d). Одержуємо (xe*d) mod n = x. Тобто для того, щоб прочитати повідомлення $ci=((mi)^e)$ mod n достатньо піднести його до ступеня d по модулю m : $((ci)d)_{\text{mod n = ((mi}}e*d)$mod n = mi.
Насправді операції зведення в ступінь великих чисел достатньо трудомісткі для сучасних процесорів, навіть якщо вони проводяться по оптимізованих за часом алгоритмах. Тому звичайно весь текст повідомлення кодується звичним блоковим шифром (набагато швидшим), але з використанням ключа сеансу, а ось сам ключ сеансу шифрується якраз асиметричним алгоритмом за допомогою відкритого ключа одержувача і поміщається в початок файлу.

Технології цифрових підписів

Як виявилося, теорія асиметричного шифрування дозволяє дуже красиво вирішувати ще одну проблему інформаційної безпеки – перевірку достовірності автора повідомлення. Для вирішення цієї проблеми за допомогою симетричної криптографії була розроблена дуже трудомістка і складна схема. В той же час за допомогою, наприклад, того ж алгоритму RSA створити алгоритм перевірки достовірності автора і незмінності повідомлення надзвичайно просто.
Припустимо, що нам потрібно передати який-небудь текст, не обов'язково секретний, але важливо то, щоб в нього при передачі по незахищеному каналу не були внесені зміни. До таких текстів звичайно відносяться різні розпорядження, довідки, і тому подібна

документація, що не представляє секрету. Обчислимо від нашого тексту яку-небудь хеш-функцію – це буде число, яке більш менш унікально характеризує даний текст.
У принципі, можна знайти інший текст, який дає те ж саме значення хеш-функции, але змінити в нашому тексті десять-двадцять байт так, щоб текст залишився повністю осмисленим, та ще і змінився у вигідну нам сторону (наприклад, зменшив суму до оплати в два рази) – надзвичайно складно. Саме для усунення цієї можливості хеш-функции створюють такими ж складними як і криптоалгоритмы – якщо текст з таким же значенням хеш-функции можна буде підібрати тільки методом повного перебору, а безліч значень складатиме як і для блокових шифрів 232–2128 можливих варіантів, то для пошуку подібного тексту зловмиснику "потрібно" ті ж самі мільйони років.
Таким чином, якщо ми зможемо передати одержувачу захищеним від зміни методом хеш-суму від тексту, що пересилається, то у нього завжди буде можливість самостійно обчислити хеш-функцію від тексту вже на приймальній стороні і звірити її з присланою нами. Якщо хоча б один біт в обчисленій їм самостійно контрольній сумі тексту не співпаде з відповідним бітом в одержаному від нас хеш-значении, значить, текст по ходу пересилки піддався несанкціонованій зміні.
Представимо тепер готову до передачі хеш-суму у вигляді декількох до-бітових блоків hi, де до – це розмір повідомлень по алгоритму RSA в попередньому параграфі. Обчислимо над кожним блоком значення si=((hi)d) mod n, де d – це той самий закритий ключ відправника. Тепер повідомлення, що складається з блоків si можна "спокійно" передавати по мережі. Ніякій небезпеці по відомим hi і si знайти Ваш секретний ключ немає – це настільки ж складна задача, як і задача "логарифмування в кінцевому полі". А ось будь-який одержувач повідомлення може легко прочитати початкове значення hi, виконавши операцію $((si)_e)$ mod n = $((hi)_d$*e)mod n = hi – Ваш відкритий ключ (e,n) є у всіх, а то, що зведення будь-якого числа в ступінь (e*d) по модулю n дає початкове число, ми довели в минулому параграфі. При цьому ніхто інший, окрім Вас, не знаючи Вашого закритого ключа d не може, змінивши текст, а отже, і хеш-суму, обчислити такі s'i, щоб при їх зведенні в ступінь e вийшла хеш-сума h'i, співпадаюча з хеш-суммой фальсифікованого тексту.
Таким чином, маніпуляції з хеш-суммой тексту представляють з себе "асиметричне шифрування навпаки" : при відправці використовується закритий ключ відправника, а для перевірки повідомлення – відкритий ключ відправника. Подібна технологія одержала назву "електронний підпис". Інформацією, яка унікально ідентифікує відправника (його віртуальним підписом), є закритий ключ d. Жодна людина, що не володіє цією інформацією, не може створити таку пару (текст,si), що описаний вище алгоритм перевірки дав би позитивний результат.
Подібний обмін місцями відкритого і закритого ключів для створення з процедури асиметричного шифрування алгоритму електронного підпису можливий тільки в тих системах, де виконується властивість коммутативності ключів. Для інших асиметричних систем алгоритм електронного підпису або значно відрізняється від базового, або взагалі не реалізовуваний.

Механізм розповсюдження відкритих ключів

Здавалося б, асиметричні криптосистеми позбавлені одного з найголовніших недоліків симетричних алгоритмів – необхідності попереднього обміну сторонами секретним ключем по захищеній схемі (наприклад, з рук в руки або за допомогою повіреного кур'єра). Неначебто достатньо "розтрубити" по всьому світлу про свій відкритий ключ, і ось готова надійна лінія передачі повідомлень.

Але виявляється не все так просто : припустимо я Ваш потенційний співбесідник. Для того, щоб відправити зашифроване повідомлення, я повинен взнати Ваш відкритий ключ. Якщо Ви не приносили мені його особисто на дискеті, значить я його просто узяв з інформаційної мережі. А зараз головне питання : де доказ, що даний набір байт є саме Вашим відкритим ключем? Адже зловмисник може згенерувати довільну пару (закритий ключ, відкритий ключ), потім активно поширювати або пасивно підміняти при запиті Ваш відкритий ключ створеним їм. В цьому випадку при відправці повідомлення 1) я зашифрую його тим ключем, який думаю, що є Вашим, 2) зловмисник, перехопивши повідомлення дешифрує його парним закритим ключем, прочитає і більш того : 3) може переслати далі, зашифрувавши дійсно вже Вашим відкритим ключем. Так само, але по інверсній схемі, він може підмінити і мій електронний підпис під моїм листом.
Таким чином, якщо між відправником і одержувачем немає конфіденційної схеми передачі асиметричних ключів, то виникає серйозна небезпека появи зловмисника-посередника. Але асиметрична криптографія знайшла витончений спосіб дуже значного зниження ризику подібної атаки. Якщо задуматися, то неправильно говорити, що між Вами і Вашим співбесідником немає гарантованої лінії зв'язку. Поза сумнівом у Вас знайдеться троє-четверо надійних знайомих в столиці або за рубежем, у них в свою чергу також знайдеться безліч знайомих в багатьох точках країни і миру. Врешті-решт, Ви користуєтеся програмним забезпеченням фірм, якщо не центри, то хоча б філіали яких знаходяться в тій країні або в тому місті, куди Ви хочете відправити лист. Проблема тільки в тому, що починаючи, з другої від Вас ланки ні Ви не знаєте людини, ні він Вас, і вірогідність того, що він, або більш того, крупна компанія, будуть що-небудь робити ради Вас, дуже мала.
Але у принципі, якщо безліч однодумців об'єднаються з метою створити надійну мережу розповсюдження ключів, то це буде їм цілком під силам. А сама асиметрична криптографія допоможе їм в цьому таким чином : насправді нікуди ходити з дискетою, одержавши прохання від свого знайомого передати відкритий ключ містера V.M.B. містеру R.H.J., не потрібно. Адже Ви спілкуєтеся з Вашим знайомим, значить, у Вас є його відкритий ключ, одержаний яким-небудь надійним способом. А отже, він може Вам прислати цей відкритий ключ містера V.M.B., підписавши повідомлення своїм електронним підписом. А від Вас до своєї черги вимагається всього лише відправити цей ключ далі по ланцюжку у напрямі містера R.H.J., підписавши вже своїм електронним підписом. Таким чином, минувши дещо переподпісиваній, відкритий ключ дійде від місця відправлення до місця вимоги по надійному шляху. У принципі від Вас навіть може не вимагатися ніяких дій – просто поставте на Вашій ЕОМ спеціальний сервер розповсюдження ключів, і він всі тільки що описані дії виконуватиме автоматично.
На сьогоднішній день не існує єдиної мережі розповсюдження відкритих ключів, і справа, як це часто буває, полягає у війні стандартів. Розвиваються декілька незалежних систем, але жодна з них не одержала переваги над іншими, яка називется "світовим стандартом, що є достатній".
Необхідно відзначити, що ланцюжок розповсюдження ключів в реальних випадках не дуже великий. Звичайно вона складається з двох-чотирьох ланок. Із залученням до процесу розповсюдження ключів крупних фірм-виробників програмних продуктів вона стає ще коротше. Дійсно, якщо на компакт-диску (не піратському !) з купленим програмним забезпеченням вже знаходиться відкритий ключ цієї фірми, а сама вона має крупний ринок збуту, то ланцюжок складатиметься або з однієї ланки (якщо ПО цієї ж фірми коштує і у Вашого потенційного співбесідника), або з двох (другим стане який-небудь інший гігантський концерн, чиє ПО встановлено у співбесідника – вже між собою-то всі крупні компанії обмінялися ключами електронних підписів достатньо давно). Відкритий ключ, підписаний якою-небудь третьою стороною, називається завіреним за допомогою

сертифікату. Сертифікатом називається інформаційний пакет, що містить який-небудь об'єкт (звично ключ) і електронний підпис, підтверджуючий цей об'єкт від імені чиєї-небудь особи.

Обмін ключами по алгоритму Діффі-Хеллмана

Даний параграф присвячений ще одному цікавому алгоритму, який достатньо важко класифікувати. Він допомагає обмінюватися секретним ключем для симетричних криптосистем, але використовує метод, дуже схожий на асиметричний алгоритм RSA. Алгоритм названий по прізвищах його творців Діффі (Diffie) і Хеллмана (Hellman). Визначимо круг його можливостей. Припустимо, що двом абонентам необхідно провести конфіденційне листування, а в їх розпорядженні немає спочатку обумовленого секретного ключа. Проте, між ними існує канал, захищений від модифікації, тобто дані, передавані по ньому, можуть прослуховувати, але не змінені (такі умови мають місце досить часто). В цьому випадку дві сторони можуть створити однаковий секретний ключ, жодного разу не передавши його по мережі, по наступному алгоритму.
Припустимо, що обом абонентам відомо деякі два числа v і n. Вони, втім, відомі і всій решті зацікавлених осіб. Наприклад, вони можуть бути просто фіксований "зашиті" в програмне забезпечення. Для того, щоб створити невідомий більш нікому секретний ключ, обидва абоненти генерують випадкові або псевдовипадкові прості числа : перший абонент – число x, другий абонент – число y. Потім перший абонент обчислює значення (vx) mod n і пересилає його другому, а другий обчислює (vy) mod n і передає першому. Зловмисник набуває обидва ці значення, але модифікувати їх (втрутитися в процес передачі) не може. На другому етапі перший абонент на основі того, що є у нього x і одержаного по мережі (vy) mod n обчислює значення (((vy) mod n)x) mod n, а другий абонент на основі того, що є у нього y і одержаного по мережі (vx) mod n обчислює значення $(((vx)^{\text{mod n}})y)$ mod n. Насправді операція зведення в ступінь переносима через операцію узяття модуля по простому числу (тобто комутативна в кінцевому полі), тобто у обох абонентів вийшло одне і те ж число : ((vx*y) mod n. Його вони і можуть використовувати як секретний ключ, оскільки тут зловмисник знову зустрінеться з проблемою RSA при спробі з'ясувати по перехопленим (vx) mod n і (vy) mod n самі числа x і y – це дуже і дуже ресурсоємна операція, якщо числа v,n,x,y вибрані достатньо великими.
Необхідно ще раз відзначити, що алгоритм Діффі-Хеллмана працює тільки на лініях зв'язку, надійно захищених від модифікації. Якби він був застосовний на будь-яких відкритих каналах, то давно зняв би проблему розповсюдження ключів і, можливо, замінив собою всю асиметричну криптографію. Проте, в тих випадках, коли в каналі можлива модифікація даних, з'являється очевидна можливість уклинення в процес генерації ключів "зловмисника-посередника" по тій же самій схемі, що і для асиметричної криптографії.

Загальна схема асиметричної криптосистеми

Загальна схема асиметричної криптосистеми зображена на малюнку 1. По структурі вона практично ідентична симетричній криптосистемі з ключем сеансу.

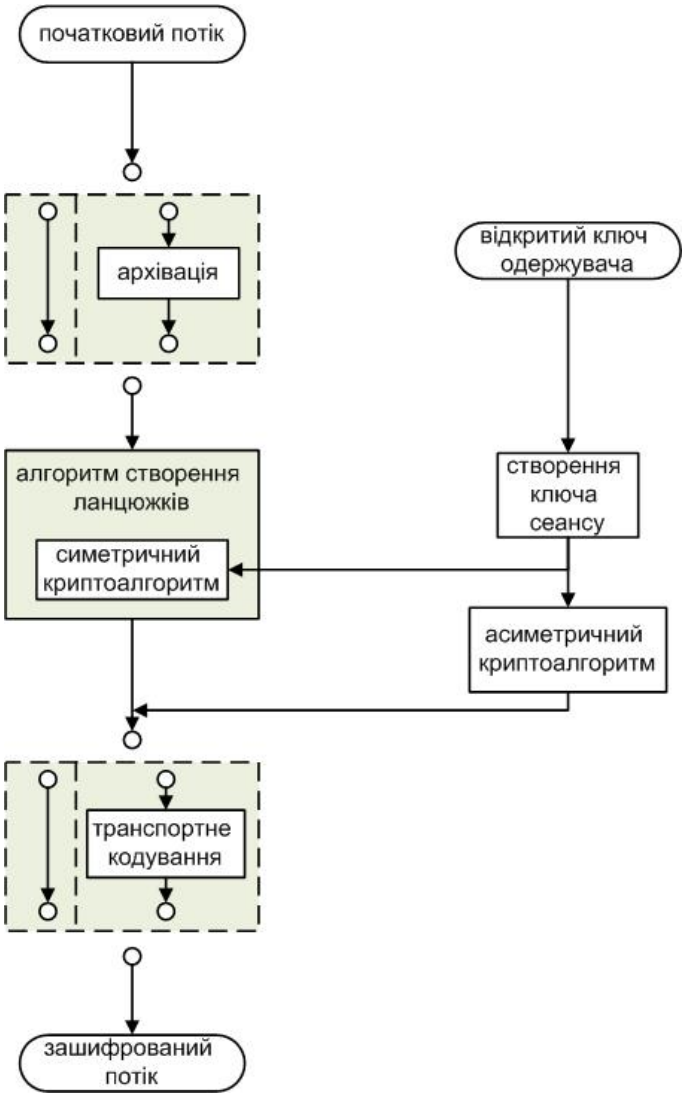


Рис.1.

3 Мережна безпека

3.1. Атакуемые сетевые компоненты

Класифікація мережних атак по меті, вибираній зловмисником для атаки. Розглянуті серверу, робочі станції, середовище передачі інформації і вузли комутації мереж.

3.2. Уровни сетевых атак согласно модели OSI

Еталонна модель взаємодії відкритих систем OSI дозволяє кваліфікувати мережні атаки по рівню протоколу, що атакується.

Мережні компоненти, що атакуються

3.1.1. Сервера

Серверу призначені для зберігання інформації або надання певних видів послуг. Внаслідок цього, основними класами атак проти серверів є "відмова в сервісі" і спроби розкриття

конфіденційної інформації. Специфічними атаками є атаки, що полягають у фальсифікації службових сервісів.

3.1.2. Рабочие станции

Основною задачею зловмисника відносно робочих станцій є отримання інформації, що бережеться локально на їх жорстких дисках, або отримання паролів, що вводяться оператором, шляхом копіювання буфера клавіатури.

3.1.3. Среда передачи информации

Різні середовища передачі даних (ефірна, кабельна) вимагають від зловмисника різних витрат для їх прослуховування.

3.1.4. Узлы коммутации сетей

Атаки на вузли комутації переслідують звичайно дві цілі : або порушення цілісності мережі ("відмова в сервісі"), або перенаправлення трафіку по невірному шляху, яким-небудь чином вигідному зловмиснику.

Серверу

Основними компонентами будь-якої інформаційної мережі є серверу і робочі станції. Серверу надають інформаційні або обчислювальні ресурси, на робочих станціях працює персонал. У принципі будь-яка ЕОМ в мережі може бути одночасне і сервером і робочою станцією – в цьому випадку до неї застосовні описи атак, присвячені і серверам і робочим станціям.

Основними задачами серверів є зберігання і надання доступу до інформації і деякі види сервісів. Отже, і всі можливі цілі зловмисників можна класифікувати як

- отримання доступу до інформації,
- отримання несанкціонованого доступу до послуг,
- спроба висновку з робочого режиму певного класу послуг,
- спроба зміни інформації або послуг, як допоміжний етап якої-небудь крупнішої атаки.

Спроби отримання доступу до інформації, що знаходиться на сервері, у принципі нічим не відрізняються від подібних спроб для робочих станцій, і ми расмотрім їх пізніше. Проблема отримання несанкціонованого доступу до послуг приймає надзвичайно різноманітні форми і грунтується в основному на помилках або недокументованих можливостях самого програмного забезпечення, що надає подібні послуги.

А ось проблема висновку з ладу (порушення нормального функціонування) сервісів досить актуальна в сучасному комп'ютерному світі. Клас подібних атак одержав назву атака "відмова в сервісі" (англ. deny service – DoS). Атака "відмова в сервісі" може бути реалізована на цілому діапазоні рівнів моделі OSI : фізичному, канальному, мережному, сеансовому. Детально схеми реалізації даної атаки ми розглянемо в параграфі, присвяченому моделі OSI.

Зміна інформації або послуг як частина більш великомасштабної атаки є також дуже важливою проблемою в захисті серверів. Якщо на сервері бережуться паролі користувачів або які-небудь дані, які можуть дозволити зловмиснику, змінивши їх, увійти до системи (наприклад, сертифікати ключів), то природно, сама атака на систему почнеться з атаки на подібний сервер. Як сервери послуг, що найчастіше піддається модифікації, слід назвати DNS-сервера.

DNS-служба (англ. Domain Name System – служба доменних імен) в мережах Intra- і Inter- Net відповідає за зіставлення "вимовних" і легко запам'ятовуються доменних імен (наприклад, www.intel.com або mail.metacom.ru) до їх IP-адресам (наприклад, 165.140.12.200 або 194.186.106.26). Пакети між станціями завжди передаються тільки на підставі IP-адресов (маршрутизатори орієнтуються тільки на їх значення при виборі напряму відправки пакету – доменне ім'я взагалі не включається в пакет, що відправляється), а служба DNS була створена в основному для зручності користувачів мережі. Як наслідок і в багатьох мережних програмах ім'я видаленого комп'ютера для більшої гнучкості або для зручності операторів заноситься не у вигляді 4-байтного IP-адреса, а у вигляді доменного імені. Так, дійсно, двох вказаних переваг буде досягнуто в цьому випадку, а ось безпека постраждає.
Річ у тому, що, якщо зловмиснику вдасться дістати права доступу до DNS-серверу, обслуговуючого дану ділянку мережі, то він цілком може змінити програму DNS-сервиса. Звичайно зміна робиться так, щоб по деяких видах запитів замість правильної IP-адреса клієнту видавалася IP-адрес якої-небудь допоміжної машини зловмисника, а вся решта запитів оброблялася коректно. Це дає можливість змінювати шлях проходження трафіку, який можливо містить конфіденційну інформацію, і робити так, що весь потік інформації, який в нормальному режимі пройшов би поза досяжністю від прослуховування, тепер поступав спочатку прямо в руки зловмисника (а потім його вже можна переправляти по справжній IP-адресу другого абонента).

Робочі станції

Основною метою атаки робочої станції є, звичайно, отримання даних, оброблюваних, або локально бережених на ній. А основним засобом подібних атак дотепер залишаються "троянські" програми. Ці програми по своїй структурі нічим не відрізняються від комп'ютерних вірусів, проте при попаданні на ЕОМ прагнуть поводитися якомога непомітніше. При цьому вони дозволяють будь-якій сторонній особі, що знає протокол роботи з даною троянською програмою, проводити видалений з ЕОМ будь-які дії. Тобто основною метою роботи подібних програм є руйнування системи мережного захисту станції зсередини – пробиття в ній величезного пролому.
Для боротьби з троянськими програмами використовується як звичне антивірусне ПО, так і декілька специфічних методів, орієнтованих винятково на них. Відносно першого методу як і з комп'ютерними вірусами необхідно пам'ятати, що антивірусне ПО знаходить величезна кількість вірусів, але тільки таких, які широко розійшлися по країні і мали численні прецендента зараження. В тих же випадках, коли вірус або троянська програма пишеться з метою отримання доступу саме до Вашої ЕОМ або корпоративної мережі, то вона практично з вірогідністю 90% не буде знайдена стандартним антивірусним ПО.
Ті троянські програми, які постійно забезпечують доступ до зараженої ЕОМ, а, отже, тримають на ній відкритий порт якого-небудь транспортного протоколу, можна знаходити за допомогою утиліт контролю за мережними портами. Наприклад, для операційних систем клона Microsoft Windows такою утилітою є програма NetStat. Запуск її з ключем "netstat -a" виведе на екран всі активні порти ЕОМ. Від оператора в цьому випадку вимагається знати порти стандартних сервісів, які постійно відкриті на ЕОМ, і тоді, будь-який новий запис на моніторі повинен привернути його увагу. На сьогоднішній день існує вже декілька програмних продуктів, що проводять подібний контроль автоматично.
_Відносно троянських програм, які не тримають постійно відкритих транспортних портів, а просто методично пересилають на сервер зловмисника яку-небудь інформацію (наприклад, файли паролів або повну копію тексту, що набирає з клавіатури), можливий тільки мережний

моніторинг. Це достатньо складна задача, що вимагає або участі кваліфікованого співробітника, або громіздкої системи ухвалення рішень.
Тому найпростіший шлях, що надійно захищає як від комп'ютерних вірусів, так і від троянських програм – це установка на кожній робочій станції програм контролю за змінами в системних файлах і службових областях даних (реєстрі, завантажувальних областях дисків і т.п.) – так званих адвізоров (англ. adviser – уведомітель).

Середовище передачі інформації

Природно, основним видом атак на середовище передачі інформації є її прослуховування. Відносно можливості прослуховування всі лінії зв'язку діляться на :

- широкомовні з необмеженим доступом
- широкомовні з обмеженим доступом
- канали "крапка-крапка"

До першої категорії відносяться схеми передачі інформації, можливість прочитування інформації з яких нічим не контролюється. Такими схемами, наприклад, є інфрачервоні і радіохвильові мережі. До другої і третьої категорій відносяться вже тільки дротяні лінії : читання інформації з них можливе або всіма станціями, підключеними до даного дроту (широкомовна категорія), або тільки тими станціями і вузлами комутації через які йде пакет від пункту відправки до пункту призначення (категорія "крапка-крапка").
До широкомовної категорії мереж відносяться мережа TokenRing, мережа EtherNet на коаксіальній жилі і на повторітелях (хабах – англ. hub). Цілеспрямовану (захищену від прослуховування іншими робочими станціями) передачу даних в мережах EtherNet проводять мережні комутатори типу свіч (англ. switch) і різного роду маршрутизатори (роутери – англ. router). Мережа, побудована по схемі із захистом трафіку від прослуховування суміжними робочими станціями, майже завжди коштуватиме дорожче, ніж широкомовна топологія, але за безпеку потрібно платити.
Відносно прослуховування мережного трафіку пристроями, що підключаються ззовні, існує наступний список кабельних з'єднань за збільшенням складності їх прослуховування :

- невита пара – сигнал може прослуховуватися на відстані в декілька сантиметрів без безпосереднього контакту,
- вита пара – сигнал дещо слабкіший, але прослуховування без безпосереднього контакту також можливе,
- коаксіальний дріт – центральна жила надійно екранована оплеткой : необхідний спеціальний контакт, що розсовує або ріжучий частину оплетки, і проникаючий до центральної жили,
- оптичне волокно – для прослуховування інформації необхідне уклинення в кабель і дороге устаткування, сам процес під'єднування до кабелю супроводжується перериванням зв'язку і може бути знайдений, якщо по кабелю постійно передається який-небудь контрольний блок даних.

Висновок систем передачі інформації з ладу (атака "відмова в сервісі") на рівні середовища передачі інформації можливий, але звичайно він розцінюється вже як зовнішня механічна або

електронна (а не програмне) дія. Можливі фізичне руйнування кабелів, постановка шумів в кабелі і в інфра- і радіо- трактах.

Вузли комутації мереж

Вузли комутації мереж представляють для зловмисників 1) як інструмент маршрутизації мережного трафіку, і 2) як необхідний компонент працездатності мережі.
Відносно першої мети отримання доступу до таблиці маршрутизації дозволяє змінити шлях потоку можливо конфіденційній інформації в сторону, що цікавить зловмисника. Подальші його дії можуть бути подібні атаці на DNS-сервер. Досягти цього можна або безпосереднім адмініструванням, якщо зловмисник яким-небудь одержав права адміністратора (найчастіше взнав пароль адміністротора або скористався незміненим паролем за умовчанням). В цьому плані можливість видаленого управління пристроями комутації не завжди благо : дістати фізичний доступ до пристрою, керованого тільки через фізичний порт, набагато складніше.
Або ж можливий другий шлях атаки з метою зміни таблиці маршрутизації – він заснований на динамічній маршрутизації пакетів, включеній на багатьох вузлах комутації. В такому режимі пристрій визначає найвигідніший шлях відправки конкретного пакету, грунтуючись на історії приходу певних службових пакетів мережі – повідомлень маршрутизації (протоколи ARP, RIP і інші). В цьому випадку при фальсифікації по певних законах декількох подібних службових пакетів можна добитися того, що пристрій почне відправляти пакети по шляху, що цікавить зловмисника, думаючи, що це і є найшвидший шлях до пункту призначення.
При атаці класу "відмова в сервісі" зловмисник звичайно примушує вузол комутації або передавати повідомлення по невірному "тупиковому" шляху (як цього можна добитися ми розглянули вище), або взагалі перестати передавати повідомлення. Для досягнення другої мети звичайно використовують помилки в програмному забезпеченні, запущеному на самому маршрутизаторі, з метою його "зависання". Так, наприклад, зовсім недавно було знайдено, що цілий модельний ряд маршрутизаторів однієї відомої фірми під час вступу на його IP-адрес досить невеликого потоку неправильних пакетів протоколу TCP або перестає передавати вся решта пакетів до тих пір, поки атака не припинитися, або взагалі зациклюється.

Рівні мережних атак згідно моделі OSI

Еталонна модель взаємодії відкритих систем OSI (англ. Open Systems Interconnection) була розроблена інститутом стандартизації ISO з метою розмежувати функції різних протоколів в процесі передачі інформації від одного абонента іншому. Подібних класів функцій було виділено 7 – вони одержали назву рівнів. Кожний рівень виконує свої певні задачі в процесі передачі блоку інформації, причому відповідний рівень на приймальній стороні проводить перетворення, точно зворотні тим, які проводив той же рівень на передаючій стороні. В цілому проходження блоку даних від відправника до одержувача показано на рис.1. Кожний рівень додає до пакету невеликий об'єм своєї службової інформації – префікс (на малюнку вони зображені як P1...P7). Деякі рівні в конкретній реалізації цілком може бути відсутнім.

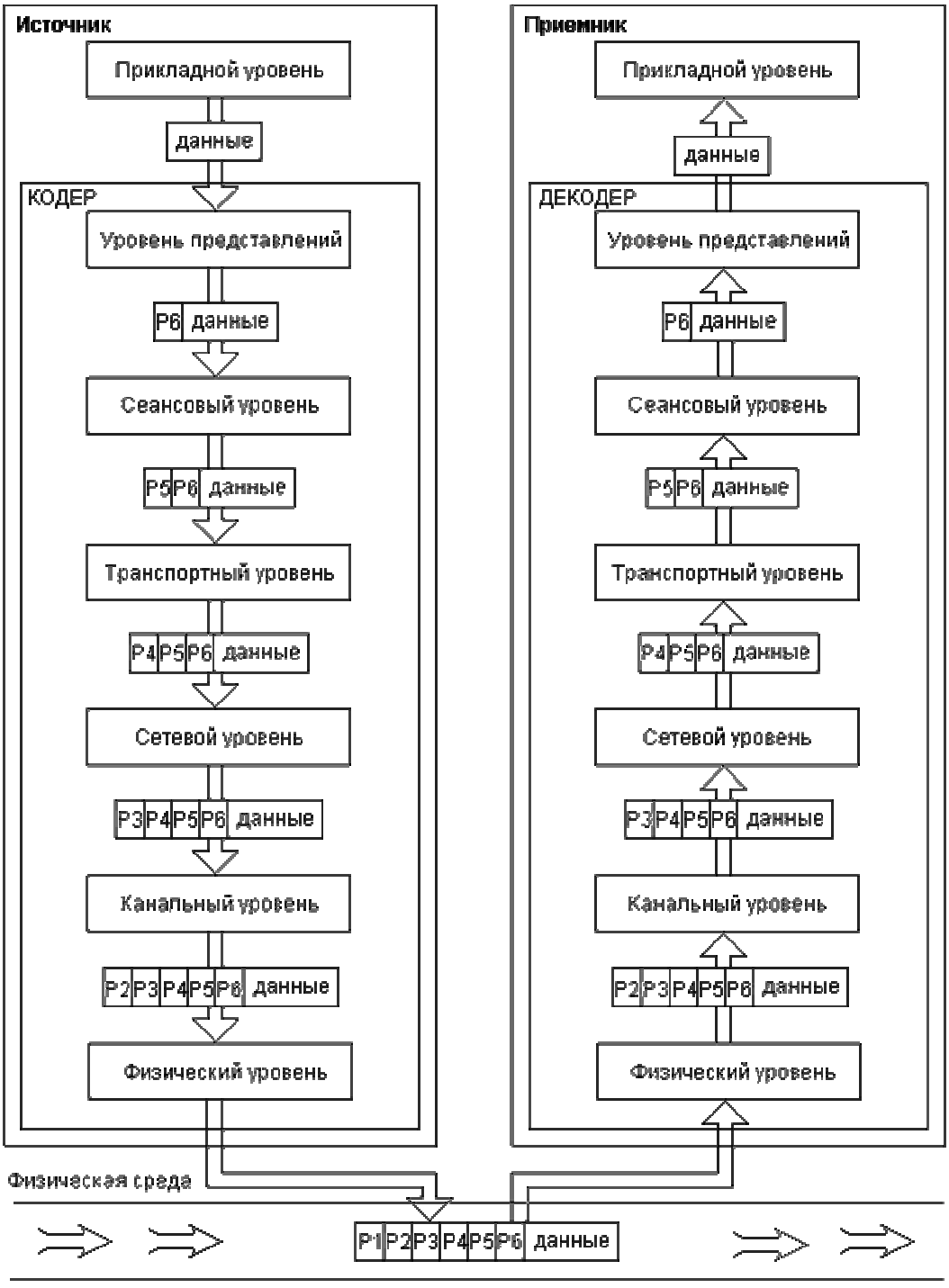


Рис.1.

Дана модель дозволяє провести класифікацію мережних атак згідно рівню їх дії.

Фізичний рівень відповідає за перетворення електронних сигналів в сигнали середовища передачі інформації (імпульси напруги, радіохвилі, інфрачервоні сигнали). На цьому рівні основним класом атак є "відмова в сервісі". Постановка шумів по всій смузі пропускання каналу може привести до "надійного" розриву зв'язку.

Канальний рівень управляє синхронізацією двох і більшої кількості мережних адаптерів, підключених до єдиного середовища передачі даних. Прикладом його є протокол EtherNet.

Дії на цьому рівні також полягають в основному в атаці "відмова в сервісі". Проте, на відміну від попереднього рівня, тут проводиться збій синхропосилок або самої передачі даних періодичною передачею "без дозволу і не свого часу".
Мережний рівень відповідає за систему унікальних імен і доставку пакетів по цьому імені, тобто за маршрутизацію пакетів. Прикладом такого протоколу є протокол Інтернету IP. Всі атаки, засновані на явно неправильній маршрутизації пакетів, ми вже розглянули.
Транспортний рівень відповідає за доставку великих повідомлень по лініях з комутацією пакетів. Оскільки в подібних лініях розмір пакету є звичайно невеликим числом (від 500 байт до 5 кілобайт), то для передачі великих об'ємів інформації їх необхідно розбивати на передаючій стороні і збирати на приймальні. Транспортними протоколами в мережі Інтернет є протоколи UDP і TCP. Реалізація транспортного протоколу – досить складна задача, а якщо ще врахувати, що зловмисник придумує самі різні схеми складання неправильних пакетів, то проблема атак транспортного рівня цілком об'ясніма.
Вся річ у тому, що пакети на приймальну сторону можуть приходити і іноді приходять не в тому порядку, в якому вони були відправлені. Причина звичайно полягає у втраті деяких пакетів через помилки або переповненість каналів, рідше – у використовуванні для передачі потоку двох альтернативних шляхів в мережі. А, отже, операційна система повинна берегти деякий буфер пакетів, чекаючи приходу затримались в дорозі. А якщо зловмисник з наміром формує пакети так, щоб послідовність була великою і явно неповною, то тут можна чекати як постійної зайнятості буфера, так і небезпечніших помилок через його переповнювання.
Сеансовий рівень відповідає за процедуру встановлення початку сеансу і підтвердження (квитування) приходу кожного пакету від відправника одержувачу. В мережі Інтернет протоколом сеансого рівня є протокол TCP (він займає і 4, і 5 рівні моделі OSI). Відносно сеансового рівня дуже широко поширена специфічна атака класу "відмова в сервісі", заснована на властивостях процедури встановлення з'єднання в протоколі TCP. Вона одержала назву SYN-Flood (зд. flood – англ. "великий потік").
При спробі клієнта підключитися до серверу, що працює по протоколу TCP (а його використовують більше 80% інформаційних служб, зокрема HTTP, FTP, SMTP, POP3), він посилає серверу пакет без інформації, але з бітом SYN, встановленим в 1 в службовій області пакету – запитом на з'єднання. Після отримання такого пакету сервер зобов'язаний вислати клієнту підтвердження прийому запиту, після чого з третього пакету починається власне діалог між клієнтом і сервером. Одночасно сервер може підтримувати залежно від типу сервісу від 20 до декількох тисяч клієнтів.
При атаці типа SYN-Flood зловмисник починає на своїй ЕОМ створювати пакети, що є запитами на з'єднання (тобто SYN-пакеты) від імені довільних IP-адресов (можливо навіть неіснуючих) на ім'я серверу, що атакується, по порту сервісу, який він хоче припинити. Всі пакети доставлятимуться одержувачу, оскільки при доставці аналізується тільки адреса призначення. Сервер, починаючи з'єднання по кожному з цих запитів, резервує під нього місце в своєму буфері, відправляє пакет-підтвердження і починає чекати третього пакету клієнта протягом деякого проміжку часу (1-5 секунд). Пакет-підтвердження піде за адресою, вказаною як помилковий відправник в довільну точку Інтернету і або не знайде адресата взагалі, або надмірно "здивує" операційну систему на цій IP-адресе (оскільки вона ніяких запитів на даний сервер не посилала) і буде просто проігнорований. А ось сервер при достатньо невеликому потоці таких запитів постійно триматиме свій буфер заповненим непотрібними очікуванням з'єднань і навіть SYN-запросы від справжніх легальних користувачів не поміщатимуться в буфер : сеансовий рівень просто не знає і не може взнати, які із запитів фальшиві, а які теперішні часи і могли б мати більший пріоритет.
Атака SYN-Flood набула досить широке поширення, оскільки для неї не вимагається ніяких додаткових підготовчих дій. Її можна проводити з будь-якої точки Інтернету на адресу будь-

якого серверу, а для відстежування зловмисника потрібно сумісні дії всіх провайдерів, що становлять ланцюжок від зловмисника до серверу, що атакується (до честі сказати, практично всі фірми-провайдери, якщо вони володіють відповідним програмним забезпеченням і кваліфікованим персоналом, беруть активну участь у відстежуванні атакуючої ЕОМ по першому ж проханню, у тому числі і від зарубіжних колег).

4 ПО і інформаційна безпека

4.1. Обзор современного ПО

Сучасне програмне забезпечення, що навіть випускається гігантами індустрії ПО, грішить тими, що з'являються і зникаючими від версії до версії помилками і непередбаченими можливостями, які зловмисники використовують в своїх цілях.

4.2. Ошибки, приводящие к возможности атак на информацию

Двома основними категоріями помилок в розробці програмного забезпечення є 1) непрогнозована взаємодія суміжних програм, процесів, процедур і т.п. і 2) невиконання набору угод, які розробник ПО вважав "саме собою розуміючими".

4.3. Основные положения по разработке ПО

Приведені узагальнені за два десятки років тисячами програмістів рекомендації по написанню додатків, в яких вірогідність появи помилок, як загальних що так і відносяться до інформаційної безпеки, прагне до нуля.

Огляд сучасного ПО

4.1.1. Операционные системы

Операційні системи є основною частиною програмного комплексу ЕОМ і при цьому виконують величезний набір проміжних операцій між прикладними програмами і апаратним забезпеченням. Природно, що помилки в них цікавлять зловмисників понад усе.

4.1.2. Прикладные программы

Основною проблемою прикладного ПО є розробка його фірмами, які просто не в змозі виділити асигнування на ретельну перевірку свого продукту. Середній клас прикладного програмного забезпечення є найвірогіднішим місцерозташуванням помилок в системі безпеки, оскільки він з одного боку вже достатньо складний, щоб в ньому з'явилися ці помилки, і з другого боку проводиться недостатньо розвинутими фірмами, щоб проходити повноцінне тестування з питань безпеки.

Операційні системи

Операційна система є найважливішим програмним компонентом будь-якої обчислювальної машини, тому від рівня реалізації політики безпеки в кожній конкретній ОС багато в чому залежить і загальна безпека інформаційної системи. В даному параграфі буде приведений короткий огляд основних сучасних операційних систем. В першу чергу нас цікавитимуть безпечне розділення оперативної пам'яті і файлів між процесами і користувачами і стійкість ОС до мережних атак.

Операційна система MS-DOS є ОС реального режиму мікропроцесора Intel, а тому тут не може йти мові про розділення оперативної пам'яті між процесами. Всі резидентні програми і основна програма використовують загальний простір ОЗУ. Захист файлів відсутній, про мережну безпеку важко сказати що-небудь визначене, оскільки на тому етапі розвитку ПО драйвери для мережної взаємодії розроблялися не фірмою MicroSoft, а стороннімі розробниками.

Сімейство операційних систем Windows 95, 98, Millenium – це клони, спочатку орієнтовані на роботу в домашніх ЕОМ. Ці операційні системи використовують рівні привілеїв захищеного режиму, але не роблять ніяких додаткових перевірок і не підтримують системи дескрипторів безпеки. В результаті цього будь-який додаток може дістати доступ до всього об'єму доступної оперативної пам'яті як з правами читання, так і з правами запису. Заходи мережної безпеки присутні, проте, їх реалізація не на висоті. Більш того, у версії Windows 95 була допущена грунтовна помилка, що дозволяє видалений буквально за декілька пакетів приводити до "зависання" ЕОМ, що також значно підірвало репутацію ОС, в подальших версіях було зроблено багато кроків по поліпшенню мережної безпеки цього клона.
Покоління операційних систем Windows NT, 2000 вже значно надійніша розробка компанії MicroSoft. Вони явлються дійсно розрахованими на багато користувачів системами, що надійно захищають файли різних користувачів на жорсткому диску (правда, шифрування даних все ж таки не проводиться і файли можна прочитати без проблем, завантажившися з диска іншої операційної системи – наприклад, MS-DOS). Дані ОС активно використовують можливості захищеного режиму процесорів Intel, і можуть надійно захистити дані і код процесу від інших програм, якщо тільки він сам не схоче надавати до них додаткового доступу ззовні процесу.
За довгий час розробки була врахована безліч різних мережних атак і помилок в системі безпеки. Виправлення до них виходили у вигляді блоків оновлень (англ. service раськ). На сьогоднішній день для Windows NT 4.0 самим останнім є оновлення "Service Раск 6", природно всі виправлення, включені в нього були враховані і при розробці Windows 2000. Таким чином дві ці операційні системи мають приблизно рівну (і дуже високу) систему безпеки.
Інша гілка клонів росте від операційної системи UNIX. Ця ОС спочатку розроблялася як мережна і розрахована на багато користувачів, а тому зразу ж містила в собі засоби інформаційної безпеки. Практично всі широко поширені клони UNIX пройшли довгий шлях розробки і у міру модифікації врахували всі відкриті за цей час способи атак. Достатньо себе зарекомендували : LINUX (S.U.S.E.), OpenBSD, FreeBSD, Sun Solaris. Природно все сказане відноситься до останніх версій цих операційних систем. Основні помилки в цих системах відносяться вже не до ядра, яке працює бездоганно, а до системних і прикладних утиліт. Наявність помилок в них часто приводить до втрати всього запасу міцності системи.

Прикладні програми

Помилки в прикладному програмному забезпеченні були і залишаються основним шляхом проникнення зловмисника як на серверу, так і на робочі станції. Об'єктивна причина цього – розробка подібного ПО різними групами розробників, які просто не в змозі уділити належної уваги мережної і локальної безпеки свого продукту. І якщо фірми-розробники операційних систем витрачають величезні суми на ретельні випробування поведінки їх програм в нестандартних ситуаціях, а також активно враховують багаторічний досвід своїх же помилок, то для невеликих фірм це просто не під силу, та і украй невигідно економічно.
Помилки активно шукаються групами "хакерів" практично у всьому більш менш поширеному ПО, проте, найбільшу популярність придбавають, звичайно, дослідження програм, встановлених майже у кожного користувача. Так, наприклад, в одній з недавніх версій MicroSoft Internet Explorer'a була знайдена помилка, пов'язана з переповнюванням буфера, яка призводила до того, що частина URL-адреса потрапляла на "виконання" і трактувалася як послідовність команд процесора. При цьому довжини цієї ділянки вистачало, наприклад, для того, щоб завантажити на ЕОМ з мережі троянську програму і передати їй

управління. В подальшій версії помилка була виправлена. Програма ICQ – найпопулярніший електронний пейджер в мережі Інтернет – в черговій своїй версії була забезпечена своїми творцями можливістю підтримувати мініатюрний WWW-сервер. Проте, помилка в його реалізації дозволяла при додаванні зліва крапок в імені першого каталога діставати доступ до всіх файлів жорсткого диска – відкривався повний (!) мережний доступ по читанню.
Багато атак використовують не тільки безпосередні помилки в реалізації ПО, але і непродумані розробниками аспекти використовування стандартних можливостей програми. Так, подаруй, найяскравішим прикладом цього є MACRO-вирусы в документах системи MicroSoft Office. Можливість виконання макросів була вбудована в цю систему з самих благих спонук, але той факт, що макроси можуть запускатися на певні події (наприклад, відкриття документа) і діставати доступ на модифікацію до інших документів, зразу ж був використаний творцями вірусів зовсім не в благих цілях.
До подібних же прикладів слід віднести можливість запуску здійснимих DLL-файлов з HLP-файлов. Здавалося б, відкривається нешкідливий текстовий файл довідки, а виявляється він може чесно ініціювати виклик з DLL-библиотеки, що додається, та ще і без жодного повідомлення про це користувача.
Мораллю цього параграфа є правило "сім разів відміряй – один відріж" на етапі розробки власного програмного забезпечення.

Помилки, що приводять до можливості атак на інформацію

Двома основними класами помилок при розробці програмного забезпечення, що приводять до потенційної можливості проведення атак на інформацію, є інтерференція даних і порушення неявних обмежень.
Інтерференція (тобто непередбачена взаємодія) даних між собою і даних з кодом є менш распространеним, але небезпечнішим синдромом, ніж описувана нижче проблема обмежень за умовчанням. Практично єдиним способом викликати нашарування даних один на одного або даних на код програми є спроба дійти при операції запису до межі області пам'яті, відведеної під даний блок інформації, і подолати її – тобто умисне переповнювання буфера. Природно, що це можливо тільки в тих ситуаціях, коли програмний код не проводить перевірки довжини записуваного значення.
Для цілих і дробових чисел, значень часу і тому подібних типів даних запис проводиться завжди у фіксованому об'ємі (2 байти, 4 байти, 10 байт). А ось для рядків і масивів даних перевірки довжини перед операціями запису необхідні. Для того, щоб примусити ЕОМ виконати код або записати дані туди, куди у нього немає прав запису, зловмисник спеціально примушує систему обробляти рядки дуже великої довжини, або поміщати в масив кількість елементів більше, ніж його об'єм. У разі успіху можливе або попадання частини рядка в сегмент коду або стека з подальшим виконанням, або модифікація яких-небудь службових даних, що дозволить потім зловмиснику увійти до системи в обхід системи захисту. Природно, що вміст кінця рядка (що виявляється після переповнювання буфера в неналежній області пам'яті) підбирається спеціальним чином. Самі дані або рядки можуг бути абсолютно безглуздими.
Проблема обмежень, які розробник програми вважає саме собо розуміючими, але які насправді можуть не виконуватися, зустрічається набагато частіше, але рідше приводить до яких-небудь серйозних наслідків. Найчастіше результатом обробки подібних даних стає переривання роботи програми з повідомленням про помилку або просто "зависання". Тобто даний клас атак використовується з метою проведення атаки "відмова в сервісі".

Спектр можливих обмежень, не продуманих на етапі розробці ПО, надзвичайно широкий. Це можуть бути і негативний час або сума платежу, і текстове значення на місці очікуваного числа, і рядки, створені цілком з управляючих символів, і, звичайно ж, порожні рядки. Все це приводить до одного з головних правил розробки ПО : ретельно і повністю перевіряйте ті вхідні дані програми, які поступають в неї від людини, або передаються по каналу зв'язку, незахищеному від модифікації.

Основні положення по розробці ПО

Загальні рекомендації по написанню стійко працюючих алгоритмів (необхідне, але не достатня умова їх інформаційної безпеки):

- не використовуйте екзотичні і недокументовані можливості мови програмування : Ви не упевнені в тому, як вони реалізуються насправді
- оформляйте початковий текст ясно і чітко, використовуйте необхідні коментарі
- використовуйте дужки для явної вказівки порядку операцій : компілятор може оптимізувати виконання виразів і почати, скажімо, складання F(1)+F(2)+F(3) з другого знаку "+", тим самим викликавши спочатку функцію F від 2, потім від 3, а тільки потім від 1 – якщо у функції змінюються які-небудь глобальні змінні це може привести до непредськазумим наслідків
- при всій слушній нагоді використовуйте передачу параметрів функції як аргументи, а не в глобальних змінних
- використовуйте структурне програмування : розбивайте складні блоки коду на процедури з ясною структурою і легко контрольованим набором параметрів
- ніколи не програмуйте недокументовані возможнності : технологія "reverse engineering" – дизасемблювання і зворотна компіляція" – на сьогоднішній день досягла величезних результатів, особливо відносно високорівневих мов програмування
- закривайте файли зразу ж після закінчення роботи з ними, а якщо Ви записуєте важливу інформацію протягом довгого часу – періодично викликайте функції скидання файлового буфера на дисковий накопичувач
- перевіряйте вільне місце на диску перед записом у файл : деякі операційні видають помилки при записі на переповнений диск нестандартним чином, результат цього може бути плачевним
- блокуйте файли і набори даних, якщо Ви звертаєтеся до них по запису з декількох паралельно працюючих процесів або програм
- прагніть якомога сильніше скоротити час запису у файли, що спільно використовуються, а, отже, і час їх блокування
- не будьте наперед упевненими, що програма запущена з тієї директорії, де розташований її здійснимий файл, – одну з перших команд після запуску програми явно змініте каталог на бажаний
- при роботі із зовнішніми і мережними пристроями і дисками будуйте цикли очікування так, щоб з них був можливий вихід після закінчення певного періоду очікування відповіді – тайм-ауту

- дуже ретельно розробляйте схему синхронізації паралелльно працюючих з одними і тими ж даними процесів
- ретельно перевіряйте алгоритми на синдром "мертвої петлі" – це ситуація, коли процес А, почавши змінювати об'єкт 1 і заблокувавши його у зв'язку з цим, чекає зняття блокування з об'єкту 2, тоді як процес В, в той же самий час що почав змінювати об'єкт 2 і заблокувавши його, чекає зняття блокування з об'єкту 1 – подібна проблема при такій схемі синхронізації теоретично нерозв'язна, єдиний вихід з неї – розглядати об'єкти 1 і 2 як єдине ціле з можливістю тільки сумісного блокування
- акуратно виділяйте і очищайте об'єкти в динамічній пам'яті
- при необхідності використовуйте криптографію
- ніколи не передавайте пароль відкритим текстом
- використовуйте криптостойкие алгоритми шифрування і хешування
- вичищайте блоки оперативної пам'яті після того, як інформація (паролі, ключі, конфіденційні дані), що знаходилася в них, стала непотрібною
- завжди перевіряйте довжини рядків і масивів перед початком роботи з ними
- встрайвайте у Ваші системи вимога реєстрації кожного оператора з унікальним паролем і записом якомога більшої кількості інформації про сеанс в балку-файл, неприступну для зміни операторам
- ретельно тестуйте Ваші додатки, зокрема на великих і неправильних вхідних даних

5 Комплексна система безпеки

5.1. Классификация информационных объектов

Оброблювані дані можуть класифікуватися згідно різним категоріям інформаційної безпеки : вимогам до їх доступності (безвідмовності служб), цілісності, конфіденційності.

5.2. Политика ролей

Ролями називаються характерні набори функцій і ступеня ответственності, властиві тими або іншими групами осіб. Чітке визначення ролей, класифікація їх рівнів доступу і відповідальності, складання списку відповідності персоналу тим або іншим ролям робить політику безпеки відносно робітництва і службовців підприємства чіткої, ясної і легкої для виконання і перевірки.

5.3. Создание политики информационной безопасности

Методика створення політики безпеки підприємства складається з обліку основних (самих небезпечних) ризиків інформаційних атак, сучасної ситуації, чинників непереборної сили і генеральної вартості проекту.

5.4. Методы обеспечения безотказности

Безвідмовність сервісів і служб зберігання даних досягається за допомогою систем самотестірованія і внесення надмірності на різних рівнях : апаратному, програмному, інформаційному.

Класифікація інформаційних об'єктів

5.1.1. Классификация по требуемой степени безотказности

Від різних типів даних потрібен різний ступінь безвідмовності доступу. Витрачати великі

гроші на системи безвідмовності для не дуже важливої інформації невигідно і навіть збитково, але не можна і неправильно оцінювати дійсно постійно необхідну інформацію – її відсутність в невідповідний момент також може принести значні збитки.

5.1.2. Классификация по уровню конфиденциальности

Класифікація по ступеню конфіденційності – одна з основних і найстаріших класифікацій даних. Вона застосовувалася ще задовго до появи обчислювальної техніки і з тих пір змінилася трохи.

5.1.3. Требования по работе с конфиденциальной информацией

Різні класи конфіденційної інформації необхідно забезпечувати різними по рівню безпеки системами технічних і адміністративних заходів.

Класифікація по необхідному ступеню безвідмовності

Безвідмовність, або надійність доступу, до інформації є однією з категорій інформаційної безпеки. Пропонується наступна схема класифікації інформації на 4 рівні безвідмовності.

Параметр	**клас 0**	**клас 1**	**клас 2**	**клас 3**
Максимально можливий безперервний час відмови	1 тиждень	1 доби	1 година	1 година
У який час час відмови не може перевищувати вказаний вище ?	у робоче	у робоче	у робоче	24 години в доба
Середня вірогідність доступності даних в довільний момент часу	80%	95%	99.5%	99.9%
Середній максимальний час відмови	1 день в тиждень	2 години в тиждень	20 хвилин в тиждень	12 хвилин в місяць

Класифікація по рівню конфіденційності

Рівень конфіденційності інформації є однією з найважливіших категорій, що приймаються в розгляд при створенні певної політики безпеки установи. Пропонується наступна схема класифікації інформації на 4 класи по рівню її конфіденційності.

Клас	**Тип інформації**	**Опис**	**Приклади**
0	відкрита інформація	загальнодоступна інформація	інформаційні брошури, відомості публікававшиеся в ЗМІ
1	внутрішня інформація	інформація, неприступна у відкритому вигляді, але не несуча ніякої небезпеки при її розкритті	фінансові звіти і тестова інформація за давно минулі періоди, звіти про звичні засідання і зустрічі, внутрішній телефонний довідник фірми

2	конфіденційна інформація	розкриття інформації веде до значних втрат на ринку	реальні фінансові дані, плани, проекти, повний набір відомостей про клієнтів, інформація про колишні і нинішні проекти з порушеннями етичних норм
3	секретна інформація	розкриття інформації приведе до фінансової загибелі компанії	(залежить від ситуації)

Вимоги по роботі з конфіденційною інформацією

При роботі з інформацією 1-го класу конфіденційності рекомендується виконання наступних вимог :

- інформування співробітників про закритість даної інформації,
- загальне ознайомлення співробітників з основними можливими методами атак на інформацію
- обмеження фізичного доступу
- повний набір документації за правилами виконання операцій з даною інформацією

При роботі з інформацією 2-го класу конфіденційності до перерахованих вище вимог додаються наступні :

- розрахунок ризиків атак на інформацію
- підтримка списку осіб, що мають доступ до даної інформації
- по можливості видача подібної інформації по розписку (зокрема електронну)
- автоматична система перевірки цлостності системи і її засобів безпеки
- надійні схеми фізичної ранспортіровки
- обов'язкове шифрування при передачі по лініях зв'язку
- схема безперебійного живлення ЕОМ

При роботі з інформацією 3-го класу конфіденційності до всіх перерахованих вище вимог додаються наступні :

- детальний план порятунку або надійного знищення інформації в аварійних ситуаціях (пожежа, повінь, вибух)
- захист ЕОМ або носіїв інформації від пошкодження водою і високою температурою
- криптографічна перевірка цілісності інформації

Політика ролей

Функції кожної людини, так чи інакше пов'язаної з конфіденційною інформацією на підприємстві, можна класифікувати і в деякому наближенні формалізувати. Подібний загальний опис функцій оператора носить назву ролі. Залежно від розмірів підприємства деякі з перерахованих нижче ролей може бути відсутнім взагалі, а деякі можуть поєднуватися однією і тією ж фізичною особою.
Фахівець з інформаційної безпеки виконує основну роль в розробці і підтримці політики безпеки підприємства. Він проводить розрахунок і перерахунок ризиків, отвественен за пошук найсвіжішої інформації про знайдених уязвімостях в програмному забезпеченні, що використовується у фірмі, і в цілому в стандартних алгоритмах.
Власник інформації – особа, що безпосередньо працює з даною інформацією, (не потрібно плутати з оператором). Часто тільки він в змозі реально оцінити клас оброблюваної інформації, а іноді і розповісти про нестандартні методи атак на неї (узкоспецифічних для цього виду даних). Власник інформації не повинен брати участь в аудиті системи безпеки.
Постачальник апаратного і програмного забезпечення. Звична стороннєє особа, яка несе відповідальність перед фірмою за підтримку належного рівня інформаційної безпеки в продуктів, що поставляються їм.
Розробник системи и/или програмного забезпечення виконує основну роль в рівні безпеки системи, що розробляється. На етапах планування і розробки повинен активно взаємодіяти з фахівцями з інформаційної безпеки.
Лінійний менеджер або менеджер відділу є проміжною ланкою між операторами і фахівцями з інформаційної безпеки. Його задача – своєчасно і якісно інструктувати підлеглий йому персонал про всі вимоги служби безпеки і стежити за її їх виконанням на робочих місцях.
Лінійні менеджери повинні бути обізнані про всю політику безпеки підприємства, але доводити до зведення підлеглих тільки ті її аспекти, які безпосередньо їх торкаються.
Оператори – особи, отвественниє тільки за свої вчинки. Вони не ухвалюють ніяких рішень і ні за ким не спостерігають. Вони повинні бути обізнані про клас конфіденційності інформації, з якою вони працюють, і про те, якого збитку буде завданий фірми при її розкритті.
Аудитори – зовнішні фахівці або фірми, наймані підприємством для періодичної (досить рідкісної) перевірки організації і функціонування всієї системи безпеки.

Створення політики інформаційної безпеки

Політика безпеки – це комплекс превентивних заходів по захисту конфіденційних даних і інформаційних процесів на підприємстві. Політика безпеки включає вимоги на адресу персоналу, менеджерів і технічних служб. Основні напрвленія розробки політики безпеки :

- визначення які дані і наскільки серйозно необхідно захищати,
- визначення хто і який збиток може нанести фірмі в інформаційному аспекті,
- обчислення ризиків і визначення схеми зменшення їх до пріємлімой величини.

Існують дві системи оцінки поточної ситуації в області інформаційної безпеки на підприємстві. Вони одержали образні назви "дослідження і "дослідження зверху вниз". Перший метод достатньо простий, вимагає набагато менших капітальних вкладень, але і володіє меншими можливостями. Він заснований на відомій схемі : "Ви – зловмисник. Ваші

дії ?". Тобто служба інформаційної безпеки, грунтуючись на даних про всі відомі види атак, намагається застосувати їх на практиці з метою перевірки, а чи можливо така атака з боку реального зловмисника.
Метод зверху "вниз" є, навпаки, детальним аналізом всієї існуючої схеми зберігання і обробки інформації. Першим етапом цього методу є, як і завжди, визначення, які інформаційні об'єкти і потоки необхідно захищати. Далі слідує вивчення поточного стану системи інформаційної безпеки з метою визначення, що з класичних методик захисту іфнормациі вже реалізоване, в якому об'ємі і на якому рівні. На третьому етап проводиться класифікація всіх інформаційних об'єктів на класи відповідно до її конфіденційності, вимог до доступності і цілісності (незмінності).
Далі слідує з'ясування наскільки серйозний збиток може принести фірмі розкриття або інша атака на кожний конкретний інформаційний об'єкт. Цей етап носить назву "обчислення ризиків". В першому наближенні ризиком називається твір "можливого збитку від атаки" на "вірогідність такої атаки". Існує безліч схем обчислення ризиків, зупинимося на одній з найпростіших.
Збиток від атаки може бути представлений ненегативним числом в приблизній відповідності з наступною таблицею :

Величина збитку	**Опис**
0	Розкриття іфнормациі принесе нікчемний моральний і фінансовий збиток фірмі
1	Збиток від атки є, але він незначний, основні фінансові операції і положення фірми на ринку не торкнуться
2	Фінансові операції не ведуться протягом деякого часу, за цей час фірма терпить збитки, але її положення на ринку і кількість клієнтів змінюються мінімально
3	Значні втрати на ринку і в прибутку. Від фірми йде відчутна частина клієнтів
4	Втрати дуже значні, фірма на період до року втрачає положення на ринку. Для відновлення положення потрібні крупні фінансові позики.
5	Фірма припиняє існування

Вірогідність атаки представляється ненегативним числом в приблизній відповідності з наступною таблицею :

Вірогідність	**Середня частота появи**
0	Даний вид атаки відсутній
1	рідше, ніж раз на рік
2	близько 1 разу на рік
3	близько 1 разу на місяць
4	близько 1 разу на тиждень
5	практично щоденно

Необхідно відзначити, що класифікацію збитку, що наноситься атакою, повинен оцінювати власник інформації, або працюючий з нею персонал. А ось оцінку вірогідності появи атаки краще довіряти технічним співробітникам фірми.
Наступним етапом складається таблиця ризиків підприємства. Вона має наступний вигляд :

Опис атаки	**Збиток**	**Вірогідність**	**Ризик (=Ущерб*Вероятность)**
Спам (переповнювання поштового ящика)	1	4	4
Копіювання жорсткого диска з центрального офісу	3	1	3
...	...	...	2
		Разом :	9

На етапі аналізу таблиці ризиків задаються деяким максимально допустимим ризиком, наприклад значенням 7. Спочатку перевіряється кожний рядок таблиці на не перевищення ризику цього значення. Якщо таке перевищення має місце, значить, даний рядок – це одна з першочергових цілей розробки політики безпеки. Потім проводиться порівняння подвоєного значення (в нашому випадку 7*2=14) з інтегральним ризиком (осередок "Разом"). Якщо інтегральний ризик перевищує допустиме значення, значить, в системі набирається безліч дрібних огрешностей в системі безпеки, які в сумі не дадуть підприємству ефективно працювати. В цьому випадку з рядків вибираються ті, які дає найзначніший внесок в значення інтегрального ризику і проводиться спроба їх зменшити або усунути повністю.
На найвідповідальнішому етапі проводиться власне розробка політики безпеки підприємства, яка забезпечить належні рівні як окремих ризиків, так і інтегрального ризику. При її розробці необхідно, проте, враховувати об'єктивні проблеми, які можуть встати на шляху реалізації політики безпеки. Такими проблемами можуть стати закони країни і міжнародного співтовариства, внутрішні вимоги корпорації, етичні норми суспільства.
Після опису всіх технічних і адміністративних заходів, планованих до реалізації, проводиться розрахунок економічної вартості даної програми. У тому випадку, коли фінансові вкладення в програму безпеки є непріємлімими або просто економічно невигідними в порівнянні з потенційним збитком від атак, проводиться повернення на рівень, де ми задавалися максимально допустимим ризиком 7 і збільшення його на один або два пункти.
Завершується розробка політики безпеки її твердженням у керівництва фірми і детальним документуванням. За цим повинна слідувати активна реалізація всіх вказаних в плані компонентів. Перерахунок таблиці ризиків і, як наслідок, модифікація політики безпеки фірми найчастіше проводиться раз в два роки.

Методи забезпечення безвідмовності

Методи підтримки безвідмовності є суміжною областю в схемах комплексної інформаційної безпеки підприємства. Основним методом в цій сфері є внесення надмірності. Вона може реалізовуватися в системі на трьох рівнях : рівні даних (або інформації), рівні сервісів (або додатків) і рівні апаратного забезпечення.
Внесення надмірності на рівні даних практикується достатньо давно : це резервне копіювання і перешкодостійке кодування. Резервне копіювання виконується звичайно при зберіганні інформації на сучасних пристроях, що запам'ятовують (оскільки для них в аварійній ситуації характерний вихід з ладу великих блоків даних цілком – трудновосстановімоє за допомогою перешкодостійкого кодування пошкодження). А ось

використовування кодів з виявленням і деяким потенціалом для виправлення помилок одержало широке вживання в засобах телекомунікації.
Внесення надмірності на рівні додатків використовується набагато рідше. Проте,, особливо мережні, служби багато кого спочатку підтримують можливість роботи з резервним або взагалі з необмеженою наперед невідомою кількістю альтернативних служб. Введення такій можливості рекомендується при розробці програмного забезпечення, проте, сам процес автоматичного перемикання на альтернативну службу винен подтвержаться криптографічним обміном первинною (настановної) інформацією. Це необхідне робити для того, щоб зловмисник не міг, вивівши з ладу реальний сервіс, нав'язати Вашій програмі свій сервіс з фальсифікованою інформацією.
Внесення надмірності на апаратному рівні реалізується звичайно відносно периферійних пристроїв (накопичувачі на гнучких і жорстких дисках, мережні і відео- адаптери, монітори, пристрої введення інформації від користувача). Це пов'язано з тим, що дублювання роботи основних компонентів ЕОМ (процесора, ОЗУ) набагато простіше виконати, встановивши просто повноцінну дублюючу ЕОМ з тими ж функціями. Для автоматичного визначення працездатності ЕОМ в програмне забезпечення вбудовуються або 1) перевірка контрольних сум інформації, або 2) тестові приклади з явно відомим результатом, що запускаються час від часу, або 3) монтування трьох і більш дублюючих пристроїв і звірка їх вихідних результати за мажоритарним правилом (яких результатів більше – ті і є правильні, а машини, що видали не такі результати, виведені з ладу).

Список літератури

1. Дж. Л. Месси. Введення в сучасну кріптологию. // ТІІЕР, т.76, №5, Травень 88 – М, Мир, 1988, с.24-42.
2. У. Діффі. Перші десять років криптографії з відкритим ключем. // ТІІЕР, т.76, №5, Травень 88 – М, Мир, 1988, с.54-74.
3. А. В. Спесивцев і ін. Захист інформації в персональних комп'ютерах. – М., Радіо і зв'язок. 1992, с.140-149.
4. В. Жельников. Криптографія від папірусу до комп'ютера. – М., ABF, 1996.
5. (!) A.Menezes, P.van Oorshot, S.Vanstone. Handbook Applied Cryptography – CRC Press Inc., 1997.
6. Hal Tipton and Micki Krause. Handbook Information Security Management – CRC Press LLC, 1998.