МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Ю.А.Тарнавський

ТЕХНОЛОГІЇ ЗАХИСТУ ІНФОРМАЦІЇ

*Затверджено Вченою радою КПІ ім. Ігоря Сікорського як підручник для студентів,
які навчаються за спеціальністю 122 «Комп'ютерні науки»,
спеціалізаціями «Інформаційні технології моніторингу довкілля»,
«Геометричне моделювання в інформаційних системах»*

Київ
КПІ ім. Ігоря Сікорського
2018

Рецензенти: *Рач В.А.*, д-р техн. наук, проф.
Бахонський О.В., канд. фіз.-мат. наук

Відповідальний редактор *Коваль О.В.* канд. техн. наук, доц.

Гриф надано Вченою радою КПІ ім. Ігоря Сікорського (протокол №7 від 25.06.2018 р.)

Електронне мережне навчальне видання

Тарнавський Юрій Адамович, канд. фіз.-мат. наук, доц.

ТЕХНОЛОГІЇ ЗАХИСТУ ІНФОРМАЦІЇ

Технології захисту інформації [Електронний ресурс] : підручник для студ. спеціальності 122 «Комп'ютерні науки», спеціалізацій «Інформаційні технології моніторингу довкілля», «Геометричне моделювання в інформаційних системах» / Ю. А. Тарнавський; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 2,04 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 162 с.

Підручник розроблений на підставі програми навчальної дисципліни «Технології захисту інформації» та містить основний теоретичний матеріал, необхідний для використання криптографічних технологій для захисту інформації.

Призначений для студентів, які навчаються за освітньою програмою підготовки бакалаврів за спеціальністю 122 «Комп'ютерні науки та інформаційні технології», спеціалізаціями «Інформаційні технології моніторингу довкілля», «Геометричне моделювання в інформаційних системах».

ЗМІСТ

Вступ

Предметом вивчення в даному посібнику є криптогра́фія (від грецького kryptós — прихований і gráphein — писати) — наука про методи захисту конфіденційності, цілісності і автентичності інформації.

Навчальна дисципліна «Технології захисту інформації» є теоретичною та практичною основою сукупності знань та вмінь, що формують профіль фахівця в області інформаційних комп'ютерних систем та технологій та однією з головних у формуванні знань у студентів з питань забезпечення захисту інформації.

Метою викладання дисципліни є надання студентам системних знань з принципів побудови систем криптографічного захисту інформації, освоєння ними необхідних знань та отримання навиків з організації та забезпечення захисту інформації в інформаційно-телекомунікаційних системах.

Завданнями вивчення навчальної дисципліни є володіння основами теорії криптографії, теоретичними знаннями про основні засоби криптографічного захисту інформації, практичними навичками проектування алгоритмів криптографічних перетворень, основними способами шифрування даних.

У результаті вивчення навчальної дисципліни студент повинен:

Знати: основні криптографічні алгоритми симетричного шифрування; основні криптографічні алгоритми асиметричного шифрування та цифрового підпису; стандартні криптографічні примітиви та порядок їх застосування при захисті інформації з обмеженим доступом; типові вимоги до систем та засобів управління ключовими даними; базові стандарти в галузі криптографічного захисту інформації.

Вміти: працювати з технічною літературою і документацією; проектувати алгоритми криптографічних перетворень; розробляти криптографічні системи та криптографічні примітиви; здійснювати загальну оцінку якості криптографічного захисту інформації в

інформаційних, телекомунікаційних та інформаційно-телекомунікаційних системах.

Посібник призначений для студентів вищих навчальних закладів, що навчаються за напрямком «Комп’ютерні науки» та осіб, що самостійно вивчають дисципліну, а також для всіх, хто цікавиться питаннями криптографічного захисту інформації.

Розділ 1. Захист інформації, його складові і рівні формування режиму інформаційної безпеки

1.1. Основні поняття і визначення

Світ є сукупністю взаємопов'язаних систем. Система характеризується такими ознаками:

a) Організація – внутрішня упорядкованість і узгодженість елементів.

b) Структура – наявність стійких зв'язків.

c) Цілісність – неможливість представлення системи як суми властивостей елементів.

З погляду окремої системи результатом її взаємодії з іншими системами є зовнішній **вплив**. Цей вплив може виявитись настільки сильним, що приведе до втрати системою цілісності, тобто до її руйнування. Такий вплив називається **небезпечним**. Вплив, який не приводить до руйнування системи називається **безпечним**. **Безпека системи** визначається мірою впливу, який можна вважати безпечним. **Захист системи** передбачає здійснення комплексу заходів, спрямованих на забезпечення достатнього **рівня безпеки системи** (Рис.1.1).

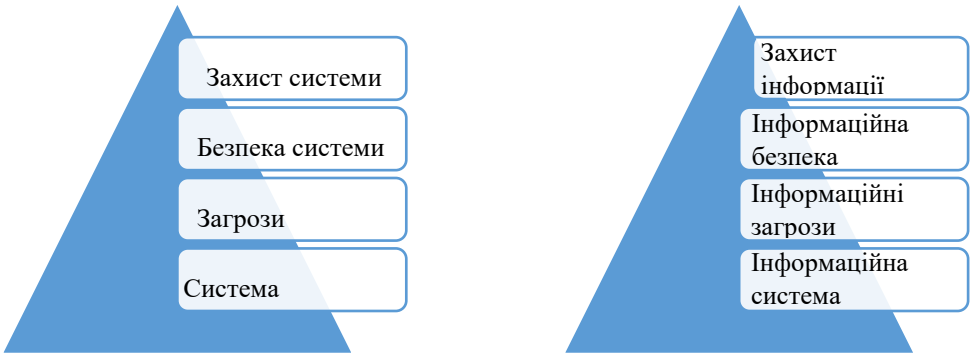


Рис.1.1. Структура понять захисту системи і інформації

Розглянемо замість абстрактної системи **інформаційну систему**, предметом якої є інформація.

Метою функціонування інформаційної системи є забезпечення обробки інформації у відповідності з визначеними правилами (які називають **політикою безпеки інформації** або **політикою інформаційної безпеки**).

Інформаційна безпека означає стійкість функціонування інформаційної системи по відношенню до випадкових або навмисних впливів на неї, що можуть нанести збиток власникам або користувачам системи.

Захист інформації передбачає комплекс заходів, спрямованих на забезпечення інформаційної безпеки.

1.2. Правові аспекти захисту інформації

В сучасному інформаційному суспільстві інформація виступає не тільки як предмет, і як продукт праці. Інформація як продукт має певну цінність. Її втрата наносить збитки власнику інформації. Запобігання і відшкодування таких збитків в суспільстві здійснюється у відповідності до встановлених юридичних норм. Тому базові поняття дисципліни формулюються в технічних термінах, що мають точні юридичні визначення.

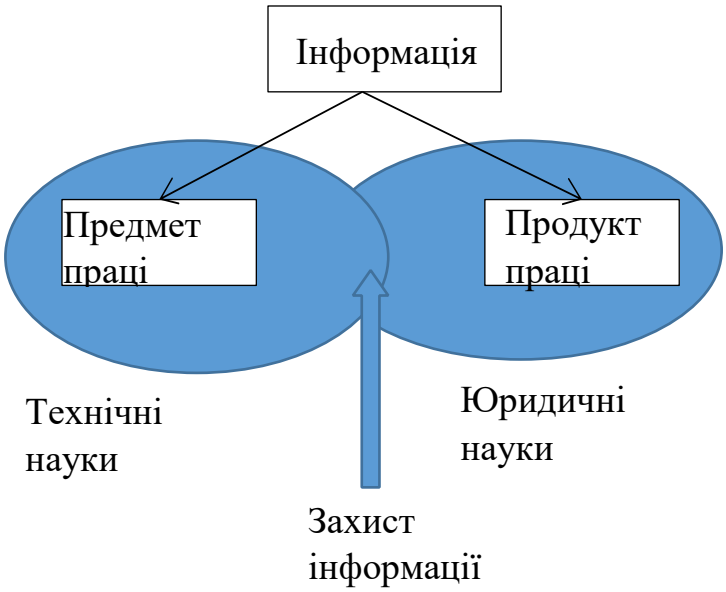


Рис.1.2. Місцезнаходження поняття «захист інформації»

Основи регулювання правових відносин щодо захисту інформації в автоматизованих системах регулює Закон України “Про захист інформації в

автоматизованих системах". Дія Закону поширюється на будь-яку інформацію, що обробляється в автоматизованих системах.

Закон визначає:

«захист інформації – сукупність організаційно-технічних заходів і правових норм для запобігання заподіянню шкоди інтересам власника інформації чи АС та осіб, які користуються інформацією».

Тут під **автоматизованою системою (АС)** розуміється «система, що здійснює автоматизовану обробку даних і до складу якої входять технічні засоби їх обробки (засоби обчислювальної техніки і зв'язку), а також методи і процедури, програмне забезпечення».

Таким чином, об'єктами захисту є:

- інформація, що обробляється в АС,
- права власників цієї інформації та власників АС,
- права користувача.

Інформація, яка є власністю держави, або інформація, захист якої гарантується державою, повинна оброблятись в АС, що має відповідний сертифікат (атестат) захищеності, в порядку, який визначається уповноваженим Кабінетом Міністрів України органом. У процесі сертифікації (атестації) цих АС здійснюються також перевірка, сертифікація (атестація) розроблених засобів захисту інформації. Інформація, яка є власністю інших суб'єктів, може оброблятися у зазначених АС за розсудом власника інформації.

Фінансування робіт, пов'язаних із захистом інформації, яка обробляється в АС, здійснюється власником АС.

Особи, винні в порушенні порядку і правил захисту оброблюваної в АС інформації, несуть дисциплінарну, адміністративну, кримінальну чи матеріальну відповідальність згідно з чинним законодавством України.

1.3. Властивості інформації з точки зору її захисту

Захист інформації полягає не лише в захисті засобів обробки інформації, а в організації засобів захисту для підтримки певних властивостей інформації. Виявилося, що такими **основними (фундаментальнимиі) властивостями** є

конфіденційність, цілісність та доступність (Рис.1.3), адже захист інформації в більшості випадків пов'язаний з комплексним рішенням трьох завдань:

1) забезпеченням конфіденційності інформації.
2) забезпеченням цілісності інформації;
3) забезпеченням доступності інформації.

Рис.1.3. Види властивостей інформації

Визначення понять конфіденційність, цілісність та доступність дається в Положенні про технічний захист інформації в Україні:

- «**конфіденційність**» - властивість інформації бути захищеною від несанкціонованого ознайомлення»;
- «**цілісність**» - властивість інформації бути захищеною від несанкціонованого спотворення, руйнування або знищення»;
- «**доступність**» - властивість інформації бути захищеною від несанкціонованого блокування».

Порушення кожної з трьох складових призводить до порушення інформаційної безпеки в цілому. Так, порушення доступності призводить до відмови в доступі до інформації, порушення цілісності призводить до фальсифікації інформації і, нарешті, порушення конфіденційності призводить до розкриття інформації.

На додаток до перерахованих вище основних характеристик безпеки можуть розглядатися також і інші характеристики безпеки. Зокрема, до таких характеристик відносяться:

- **Спостереженість** (accountability) –- забезпечення ідентифікації суб'єкта доступу та реєстрації його дій.
- **Неспростовність** (non-repudiation) - неможливість відмови від авторства.
- **Автентичність** (authenticity) - гарантує, що суб'єкт або ресурс ідентичні заявленим.
- **Достовірність** (reliability) - властивість відповідності передбаченому поводженню чи результату.
- **Адекватність** – відповідність створюваного з допомогою отриманої інформації образу реальному об'єкту, процесу, явища тощо

1.4. Рівні формування режиму інформаційної безпеки

Правильний з методологічної точки зору підхід до проблем інформаційної безпеки починається з виявлення суб'єктів інформаційних відносин та інтересів цих суб'єктів, пов'язаних з використанням інформаційних систем. Це обумовлено тим, що для різних категорій суб'єктів характер вирішуваних завдань може істотно розрізнятися. Наприклад, завдання, які вирішуються адміністратором локальної мережі щодо забезпечення інформаційної безпеки, значною мірою відрізняються від завдань, розв'язуваних користувачем на домашньому комп'ютері, не пов'язаному мережею.

Розрізняють три рівні формування режиму інформаційної безпеки (Рис.1.4):

1. законодавчо-правовий;
2. адміністративний (організаційний);
3. програмно-технічний.

Рис.1.4. Рівні формування режиму інформаційної безпеки

Законодавчо-правовий рівень включає комплекс законодавчих та інших правових актів, які установлюють правовий статус суб'єктів інформаційних відносин, суб'єктів і об'єктів захисту, методи, форми і способи захисту, їх правовий статус. До цього рівня відносяться стандарти і специфікації в галузі інформаційної безпеки.

До цього рівня можна віднести і морально-етичні норми поведінки, які склалися традиційно або складаються в міру поширення обчислювальних засобів в суспільстві. Морально-етичні норми можуть бути регламентованими в законодавчому порядку, тобто у вигляді зведення правил і приписів. Найбільш характерним прикладом таких норм є Кодекс професійної поведінки членів Асоціації користувачів ЕОМ США. Тим не менше, ці норми не є обов'язковими як законодавчі заходи.

Адміністративний рівень включає комплекс скоординованих організаційних і технічних заходів, що реалізують практичні механізми захисту в процесі створення і експлуатації систем захисту інформації.

Організаційний рівень повинен охоплювати всі структурні елементи систем обробки даних на всіх етапах їх життєвого циклу: будівництво приміщень, проектування системи, монтаж і налагодження обладнання, випробування і перевірки, експлуатації.

Програмно-технічний рівень включає три підрівня:

- фізичний;
- технічний (апаратний);
- програмний.

Фізичний підрівень вирішує завдання з обмеженням фізичного доступу до інформації та інформаційних систем. До нього відносяться технічні засоби, що реалізуються у вигляді автономних пристроїв і систем, не пов'язаних з обробкою, зберіганням і передачею інформації: система охоронної сигналізації, система спостереження, засоби фізичного перешкоджання доступу (замки, огородження, грати і т. д.).

Засоби захисту апаратного та програмного підрівня безпосередньо пов'язані з системою обробки інформації. Ці засоби або вбудовані в апаратуру, або поєднані з нею по стандартним інтерфейсам. До апаратних засобів відносяться схеми контролю інформації по парності, схеми доступу по ключу і т. д.

До програмних засобів захисту, що створює програмний підрівень, відносяться спеціальне програмне забезпечення, що використовується для захисту інформації, наприклад, антивірусний пакет і т. д. Програми захисту можуть бути як окремі, так і вбудовані. Так, шифрування даних можна виконати вбудованою в операційну систему файлової шифрувальної системи EFS (Windows 2000, XP) або спеціальною програмою шифрування.

Формування режиму інформаційної безпеки є складною системною задачею, рішення якої в різних країнах відрізняється за змістом і залежить від таких факторів, як науковий потенціал країни, ступінь впровадження засобів інформатизації в житті суспільства та економіку, розвиток виробничої бази, загальної культури суспільства і, нарешті, традицій і норм поведінки.

Розділ 2. Традиційні криптографічні системи

2.1. Криптографія і її основні поняття

В перекладі з грецької мови слово *криптографія* означає тайнопис. Основне призначення криптографії – утаємничити необхідну інформацію.

Криптографія надає засоби для захисту інформації і тому є складовою діяльності з забезпечення безпеки інформації.

Існують різні засоби втаємничення інформації:

- Приховування каналу передачі повідомлення.
- Маскування змісту повідомлення з використанням стеганографічних методів.
- Ускладнення можливості перехоплення самого повідомлення противником.
- Інші.

На відміну від перерахованих методів криптографія не «приховує» повідомлення, а перетворює їх у формы, недоступну для розуміння противником. Таке перетворення забезпечується використанням криптографічних систем.

2.2. Модель криптографічної системи

Криптографічна система (криптосистема) – система секретного зв'язку, в якій зміст інформації, що передається, утаємничується за допомогою криптографічних перетворень; при цьому сам факт передачі інформації не приховується.

Криптографічні перетворення визначаються певним параметром, який називається ключ. Зазвичай ключ є буквеною або числовою послідовністю. Кожне криптографічне перетворення однозначно визначається ключем і описується певним криптографічним алгоритмом.

Криптографічними перетвореннями є:

Шифрування– процес перетворення вихідного тексту (P) в зашифрований текст (C) за допомогою шифруючої функції (E) з секретним ключем шифрування (Ke) у відповідності з обраним алгоритмом шифрування: $C=E_{Ke}(P)$.

Розшифрування – обернений шифруванню процес перетворення зашифрованого тексту (C) в вихідний текст (P) за допомогою функції розшифрування (D) з секретним ключем розшифрування (Kd) у відповідності з обраним алгоритмом шифрування: $P=D_{Kd}(C)$.

Сімейство обернених перетворень защифрування і розшифрування називають **шифром**.

Алгоритми шифрування і розшифрування можуть відрізнятись, відповідно можуть розрізнятись і ключі шифрування і розшифрування.

В загальному випадку криптосистема має наступну структуру (Рис.2.1):

Рис.2.1. Структура криптосистеми

Робота криптосистеми може бути описана наступним чином:

1. З джерела ключів вибирається ключі (шифрування K_e і розшифрування K_d) і відправляються по надійним каналам передаючій і приймаючій стороні.
2. До вихідного (або відкритого) повідомлення p, що призначене для передачі, застосовується алгоритм шифрування E_{Ke}, внаслідок чого отримується зашифроване повідомлення $C=E_{Ke}(p)$.

3. Зашифроване повідомлення пересилається по каналу для обміну повідомленнями, який не вважається надійним (тобто зашифроване повідомлення може бути перехоплене порушником), приймаючій стороні.
4. На приймаючій стороні до зашифрованого повідомлення C застосовується обернене перетворення для отримання вихідного повідомлення $p=D_{Kd}(C)$.

В класичній криптографії для шифрування і розшифрування використовувався один і той самий ключ: $K_e=K_d=K$. Такі криптосистеми отримали назву криптосистем з секретним ключом (secret key cryptosystems); сьогодні їх також називають симетричним криптосистемами (symmetric cryptosystems).

Зауважимо, що алгоритми шифрування і розшифрування E і D відкриті, і секретність вихідного тексту p в даному шифротекста C залежить від таємності ключа K.

Для сучасної криптографії революційним стало усвідомлення того факту, що ключі шифрування і розшифрування можуть не співпадати. Такі криптосистеми отримали назву асиметричних криптосистем (asymmetric cryptosystems).

Оскільки зашифроване повідомлення передається через канал, доступний противнику, можливе його перехоплення і «прочитання» особою, яка не має ключа. В цьому випадку говорять про **дешифрування** повідомлення.

Таким чином, терміши «розшифрування» і «дешифрування» слід розрізняти: при розшифровуванні ключ вважається відомим, тоді як при дешифруванні ключ невідомий.

Розшифрування здійснюється так само просто, як і шифрування. Дешифрування є значно складнішою задачею. Рівень складності цієї задачі і визначає здатність протистояти спробам противника заволодіти інформацією,

яка захищається. В зв'язку з цим говорять про криптографічну стійкість шифру, розрізняючи більш і менш стійкі.

Наука, що вивчає методи дешифрування, називається **криптоаналізом**.

Галузь знань, що об'єднує криптографію і крипто аналіз, називають **криптологією**.

2.3. Принцип Керкхоффса

Основне правило криптографії - використовувати відкриті й опубліковані алгоритми та протоколи.

Вперше цей головний принцип був сформульований у 1883 році Агустом Керкхгоффсом (A.Kerckhoffs): в криптографічній системі єдиним секретом має залишатися ключ, сам же алгоритм не повинен бути засекречений.

Сучасні криптологи повністю прийняли цей принцип, називаючи все, що йому не відповідає, "безпекою через неясність". Будь-яка система, що тримає в цілях безпеки свої алгоритми в секреті, просто ігнорується співтовариством і обзивається "ханаанським бальзамом".

Висновок з принципу Кірхгофа полягає в тому, що чим менше секретів містить система, тим вище її безпека. Якщо втрата будь-якого із секретів призводить до руйнування системи, то система з меншим числом секретів безумовно буде надійнішою. Чим більше секретів містить система, тим вона більш крихка. Менше секретів - вище міцність.

Обґрунтування принципу Кірхгофа полягає у такому: якщо для забезпечення безпеки системи криптоалгоритм повинен залишатися в таємниці, то система буде більш крихкою просто тому, що в ній виявиться більше секретів, які для забезпечення її безпеки потрібно зберігати.

2.4. Етапи розвитку криптографічних систем

В ході розвитку криптосистем виділяють такі етапи (Таблиця 2.1).

Таблиця 2.1. **Етапи розвитку криптосистем**

Етап	**Час**	**Особливості**
I	V-IV тт. до н.е.	Виникнення перших шифрувальних пристроїв (Сцитала, табличка Енея і т.п.)
II	I ст. до н.е.	Початок застосування шифрованого зв'язку в системі органів влади (шифр Цезаря і т.п.)
III	XV-XVI ст. (Відродження)	Розвиток наукових методів криптографії і крипто аналізу (метод частотного аналізу, винайдення поліалфавітних шифрів, поворотної решітки)
IV	XVII-XVIII ст. (ера «чорних кабінетів»)	Використання номенклаторів як основного засобу шифрування
V	XIX ст.	Винайдення гаміювання
VI	XX ст.	Винайдення колісних шифраторів (Енігма)
VII	кінець XX ст.	Народження «нової криптографії»

Перші системи шифрування з'явились одночасно з письменністю у V-IV тт. до н.е. Відомо, що вже в той час в Спарті використовувався один з перших шифрувальних пристроїв – **Сцитала**. Це був жезл циліндричної форми, на який намотувалась стрічка пергаменту. Вздовж осі циліндру записувався текст, призначений для передачі. Після запису стрічка знімалась з жезлу і передавалась адресату, який мав таку саму Сциталу.

Цікаво, що був відомий і метод дешифрації такого шифру, винайдення якого приписується Арістотелю. Пропонувалось заточити на конус довгий брус і, обернувши його стрічкою, починати зсувати її по конусу від малого діаметру до найбільшого. Там, де діаметр конусу співпадав з діаметром Сцитали, букви тексту утворювали слова. Далі залишалось тільки виготовити циліндр потрібного діаметру.

Іншим шифрувальним засобом часів Спарти була **табличка Енея**. На невеличкій горизонтальній табличці розташовувався алфавіт, а по її боковим сторонам розташовувались виїмки для намотування нитки. При шифруванні нитка закріплювалась з однієї із сторін таблички і намотувалась на неї. На нитці робились відмітки (наприклад, вузлики) в місцях, що знаходились напроти букв даного тексту. Після шифрування нитка змотувалась і доставлялась адресату. Цей шифр був досить надійним: історії не відомі факти його дешифрування.

Подібні шифрові пристосування з невеличкими змінами проіснували до епохи розквіту Риму. Для управління імперією шифрований зв'язок в системі органів влади був життєво необхідним. Особливу роль відіграв шифр, запропонований Юлієм Цезарем: кожна буква повідомлення замінялась буквою алфавіту, зсунутою на три позиції (замість A – D, B – E, … , Z –A).

З часів Цезаря до XIV-XV століть криптографія розвивалась, але практика шифрування зберігалась у глибокій таємниці. Так, в часи хрестових походів шифрувальники Папи Римського після року роботи підлягали фізичному знищенню. Тому про цей період майже нічого невідомо.

В епоху Відродження почали розвиватись наукові методи криптографії і крипто аналізу. Зокрема для дешифрування шифрів став використовуватись **метод частотного аналізу** зустрічає мості букв, для шифрування почали використовуватись **шифри багатозначної заміни** (омофони), отримали поширення **багатоалфавітні шифри** (шифр Тритемія, шифр Віженера), в якості засобу шифрування почала використовуватись **поворотна решітка**.

Поворотна решітка представляла собою лист з твердого матеріалу з прорізями. Накладаючи таку решітку на лист паперу, можна записувати у вирізи таємне повідомлення. Після цього, знявши решітку, треба було заповнити вільні місця, що залишились, маскуючим текстом.

Подібним стеганографічним методом маскування повідомлень користувались багато історичних осіб, зокрема, кардинал Рішіл'є у Франції. Він використовував в якості решітки прямокутник розміру 7x10 (Рис.2.2) з

прорізами в позиціях (1,8), (2,9), (3,6), (4,5), (4,6), (5,1), (5,6), (5,7), (5,9), (6,2), (6,10), (7,9), (7,10). Для довгих повідомлень прямокутник застосовувався кілька разів.

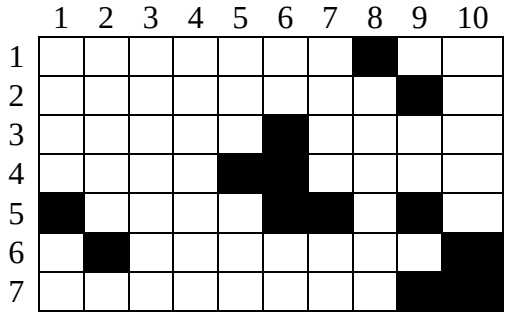


Рис.2.2. Решітка Рішіл'є

В історії криптографії XVII-XVIII ст. називають ерою «чорних кабінетів». В цей період в багатьох державах Європи з'явились дешифрувальні підрозділи, які і називались «чорними кабінетами». Перший з них був створений у Франції за ініціативою кардинала Рішіл'є.

Характерним для цього періоду є широке розповсюдження шифрів, які називались **номенклаторами**. Номерклатори поєднували в собі просту заміну і код. В якості прикладу номенклатура наведемо «малий шифр», що використовувався в наполеонівській армії (Таблиця 2.2).

Таблиця 2.2. **«Малий шифр», що використовувався в наполеонівській армії**

A-15 AR-25 AL-39	J-87 JAI-123	R-169 RA-146 RE-126 RI-148
B-37 BU-3 BO-35 BI-29	K-?	S-167 SA-171 SE-177 SI-134
C-6 CA-32 CE-20	L-96 LU-103 LE-117	SO-168 SU-174
D-23 DE-52	LA-106	T-176 TI-145 TO-157
E-53 ES-82 ET-50 EN-68	M-114 MA-107	U-138
F-55 FA-69 FE-58 FO-71	N-115 NE-94 NI-116	V-164 VE-132 VI-161 VO-175
G-81 GA-51	O-90 OT-153	W, X, Y-?
H-85 HI-77	P-137 PO-152	Z-166
I-119 IS-122	Q-173 QUE-136	

Приклад застосування цього шифру наведений в Таблиці 2.3.

Таблиця 2.3. **Приклад шифрування «малим шифром» слова NAPOLEON**

N	A	P	O	L	E	O	N
115	15	137	90	96	53	90	115
або							
N	A	PO	LE	O	N		
115	15	152	117	90	115		

В простих нуменклаторах код складався з кількох десятків слів або фраз з двобуквеними кодовими позначеннями. З часом списки слів в номерклаторах збільшились до двох-трьох тисяч.

В цілому XVII-XVIII ст. не дали для криптографії нових ідей. Ера «чорних кабінетів» закінчилась в 40их роках XIX ст.

Нові ідеї з'явились в середині XIX ст. Цей період пов'язаний з виникненням стійкого способу ускладнення числових кодів – **гаміювання**. Він полягав у перешифровуванні закодованого повідомлення за допомогою деякого ключового числа, яке називалось **гамою**. Шифрування за допомогою гами полягало у сумуванні всіх кодованих груп повідомлення з одним і тим самим ключовим числом.

Приклад 1. Результат накладення гами 6413 на кодований текст (одиниці перенесення, що з'являються при додаванні між кодовими групами, опускаються):

$$\begin{array}{rr} & 3425\ \ 7102\ \ 8139 \\ + & \underline{6413\ \ 6413\ \ 6413} \\ & 9838\ \ 3515\ \ 4552 \end{array}$$

Приклад 2. Результат оберненої операції «зняття гами» 6413:

$$\begin{array}{rr} & 9838\ \ 3515\ \ 4552 \\ - & \underline{6413\ \ 6413\ \ 6413} \\ & 3425\ \ 7102\ \ 8139 \end{array}$$

Перша світова війна стала поворотним пунктом в розвитку криптографії в зв'язку зі зростанням обсягів шифропереписки. Криптографія стала широким полем діяльності. Проте нових наукових ідей в цей час не з'явилося.

Майже половина XX ст. була пов'язана з використанням **колісних шифраторів**. Їх різні конструкції були запатентовані майже одночасно в різних країнах (Голандії, Німеччині, Швеції). Найбільш відомою шифромашиною цих часів була «Енігма», яка у довоєнний період і під час другої світової війни використовувалась в германській армії. Принцип її роботи такий. На екрані оператору показувалася буква, якою шифрувалася відповідна літера на клавіатурі. Те, якою буде зашифрована літера, залежало від початкової конфігурації коліс. Суть в тому, що існувало понад сто трильйонів можливих комбінацій коліс, і з часом набору тексту колеса зсувалися самі, так що шифр змінювався протягом усього повідомлення. Все Енігми були ідентичними, так що при однаковому початковому положенні коліс на двох різних машинах і текст виходив однаковий. У німецького командування були Енігми і список положень коліс на кожен день, так що вони могли з легкістю розшифровувати повідомлення один одного, але вороги не повідомляючи положень послання прочитати не могли.

У другій половині XX ст. в зв'язку з розвитком елементної бази обчислювальної техніки з'явились **електронні шифратори**. Сьогодні для шифрування даних найбільш широко застосовують три види шифраторів: апаратні, програмно-апаратні та програмні. Їх основна відмінність полягає не тільки в способі реалізації шифрування і ступеня надійності захисту даних, але і в ціні. Найдешевші пристрої шифрування - програмні, потім йдуть програмно-апаратні засоби і, нарешті, найдорожчі - апаратні.

В 70-их роках народилась «нова криптографія» - **криптографія з відкритим ключем**. Криптографія почала широко використовуватись не тільки у військовій, дипломатичній, державній сферах, а також в комерційній, банківській та інших сферах.

2.5. Види історичних шифрів

Історичними (докомпютерними) називають шифри, що використовувались до 1960 року, тобто в доком'ютерну епоху. Ці шифри є типовими прикладами симетричних алгоритмів шифрування і залишаються основою багатьох сучасних криптосистем.

В класичній криптографії доведено, що в основі криптографічних алгоритмів знаходяться тільки два основних типи перетворень – заміни і перестановки; всі інші є лише їх комбінацією.

В **перестановочних** шрифтах символи відкритого тексту змінюють свої розташування. Наприклад, відкритий текст виписується у вигляді матриці з пронумерованими стовпцями, після чого порядок стовпців змінюється:

Простий шифр маршрутної перестановки

Саме цей шифр реалізовувала Сцитала.

У даному вигляді шифру текст пишеться на горизонтально разграфленому аркуші паперу фіксованої ширини, а шифротекст зчитується по вертикалі. Розшифрування полягає в записі шіфротекста вертикально на аркуші розграфлений паперу фіксованої ширини і потім зчитуванні відкритого тексту горизонтально.

Для приклада (Рис.2.3) в якості відкритого тексту взято КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ.

К	И	Ї	В	С
	Й	И	К	Ь
П	О	Л	І	Т
Ч	І	Н	Х	Е
Н	И	Й		І
Т	И	Т	С	Н
У	Т			

Рис.2.3. Приклад використання маршрутної перестановки

Тоді зашифрований текст буде таким: к пчнтуийоіиит тйнлиївкіх с ніетьс.

Стовпчикова транспозиція

Стовпчикова транспозиція – перестановочний шифр, в якому використовується блоковий алгоритм шифрування:

1. Відкрите повідомлення вписується в прямокутник [nxm] заздалегідь обумовленим способом.
2. Стовпчики прямокутника нумеруються в звичайному порядку.
3. Стовпчики переміщуються в порядок, що задається вказаною ключовою послідовністю (або в порядку букв ключа - літерного ключового слова).

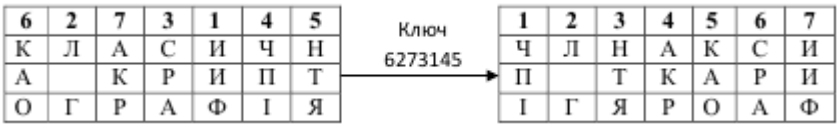


Рис.2.4. Приклад використання стовпчикової перестановки

Простий перестановочний шифр

Фіксується матриця перестановки n-елементів. Для n=5 вона може бути такою:

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 1 & 3 & 4 \end{pmatrix}$$

Вхідне повідомлення розбивається на блоки по n символів, в кожному з яких символи переставляються у відповідності з матрицею перестановки.Саме перестановка є секретним ключем.

Зашифруємо в якості прикладу відкритий текст ПЕРЕСТАНОВКА.

Розіб'ємо текст на частини по 5 букв: ПЕРЕС ТАНОВ КА. Потім переставимо букви в них відповідно до нашої матриці: еспре автно ка. Прибравши проміжки між групами, отримаємо шіфротекст: еспреавтнока.

Шифр зсуву

Шифр зсуву – один з перших відомих шифрів, що використовувався ще в давнину. Використовується поточний алгоритм шифрування, який полягає у такому:

1. Перенумеровуються всі літери вихідного алфавіту, починаючи з 0, наприклад, букві «a» присвоюється номер 0, «b» - 1, і т.д. до літери «z» під номером 25.
2. Переписується вихідний текст з заміною кожної букви відповідним номером.
3. До кожного числа в утвореній послідовності додається значення3 ключа К по модулю 26; К- ціле число між 0 і 26.

Криптостійкість шифру зсуву вкрай низька. Наївний шлях атаки на шифр зсуву полягає в простому переборі можливих значень ключа до тих пір, поки не вийде осмислений текст. Оскільки існує рівно 25 варіантів (ключ О не змінює тексту), то для їх перебору потрібно не дуже багато часу, особливо у випадку короткої шифрограми.

Приклад 1. Шифр зсуву з кроком 1 використовував ще імператор Август (1 в. н. е.). У своєму листуванні він замінював першу букву латинського алфавіту на другу, другу - на третю і т. д., нарешті, останню – на першу:

a b c d e f g h i j k l m n o p q r s t u v w x y z

B C D E F G H I J K L M N O P Q R S T U V W X Y Z A

Улюблений вислів імператора Августа виглядало так: GFTUJOB MFOUF ("Festina lente" - лат. "Поспішай повільно").

Приклад 2. Шифр зсуву з ключем 3 використовував ще Юлій Цезар; тому його також називають **шифром Цезаря**. У 1 в. н.е. Юлій Цезар під час війни з галлами, листуючись зі своїми друзями в Римі, заміняв у повідомленні першу літеру латинського алфавіту (A) на четверту (D), другу (B) - на п'яту (E), нарешті, останньому - на третю:

a b c d e f g h i j k l m n o p q r s t u v w x y z

D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Донесення Ю. Цезаря Сенату про здобутуї їм перемогу над Понтійським царем виглядало так: YHQL YLGL YLFL ("Veni, vidi, vici" - лат. "Прийшов, побачив, переміг").

Приклад 3. Прикладом шифру простої заміни може служити програма ROT13, яку зазвичай можна знайти в операційній системі UNIX. З її допомогою буква "A" відкритого тексту англійською мовою замінюється на літеру "N", "B" - на "O" і так далі. Таким чином, ROT13 циклічно зсуває кожну букву англійського алфавіту на 13 позицій вправо. Щоб отримати вихідний відкритий текст треба застосувати функцію шифрування ROT 13 двічі: P = ROT13 (ROT13 (P)).

В шифрах **заміни** один символ відкритого тексту замінюється символом зашифрованого тексту.

Шифр заміни

Основний недолік шифру зсуву полягає в тому, що існує надто мало можливих ключів – всього 25. З метою усунення цього недоліку був винайдений шифр заміни.

Шифр заміни використовує блочний алгоритм шифрування, який полягає у такому:

1. Описуєтьяс ключ такого шифру, для чого спочатку виписується алфавіт, а беспосередньо під ним - той же алфавіт, але з переставленими літерами тогож або іншого алфавіту.
2. Шифрування полягає в заміні кожної букви у відкритому тексті на відповідну їй нижню літеру.
3. Щоб розшифрувати шифротекст, потрібно кожну його букву знайти в нижньому рядку таблиці і замінити її відповідною верхньою.

Кількість всіх можливих ключів шифру заміни збігається з числом всіх перестановок 26 елементів, тобто рівним 26! ~10^9. Тому, перебираючи всі можливі ключі за допомогою комп'ютера, ми витратимо стільки часу, що задача про дешифрування конкретного повідомлення перестане бути

актуальною. Тим не менш, шифр заміни можна зламати, спираючись на частотному аналізі символів мови.

Шифр заміни є блоковим, блок в якому складається з однієї букви. Блок шифротексту отримується з блоку відкритого тексту в результаті застосування ключа (таблиці відповідності).

Приклад 4. Криптограма слова «hello» буде виглядати як ESVVJ, якщо користуватися наведеною

відповідністю:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
G	O	Y	D	S	I	P	E	L	U	A	V	C	R	J	W	X	Z	N	H	B	Q	F	T	M	K

Поліалфавітний шифри заміни

Шифр заміни відноситься до так званих **моноалфавітних** шифрів заміни, в яких використовується тільки один впорядкований набір символів, підміняючий собою стандартний алфавіт.

Основний недолік шифру заміни (і зсуву) полягає в тому, що кожна буква відкритого тексту при шифруванні замінюється раз і назавжди фіксованим символом. Тому при дешифруванні ефективно працює частотний аналіз символів мови. З початку XIX століття розробники шифрів намагалися подолати такий зв'язок між відкритим текстом і його шифрованим варіантом.

Один із шляхів вирішення вказаної проблеми полягає в тому, щоб брати кілька наборів символів замість стандартного алфавіту і шифрувати букви відкритого тексту, вибираючи відповідні знаки з різних наборів у визначеній послідовності. Шифри такого типу носять назву **поліалфавітних** шифрів заміни.

Алгоритм поліалфавітного шифру заміни може полягати у такому:

1. Описується відповідність між алфавітом відкритого тексту і кількома рівнями алфавітів шифротексту[1]:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
T	M	K	G	O	Y	D	S	I	P	E	L	U	A	V	C	R	J	W	X	Z	N	H	B	Q	F
D	C	B	A	H	G	F	E	M	L	K	J	I	Z	Y	X	W	V	U	T	S	R	Q	P	O	N

2. При шифруванні букви відкритого тексту, що знаходяться на непарних позиціях, замінюються відповідними буквами другого рядка, а ті, що знаходяться на парних позиціях – буквами третього рядка.
3. При розшифруванні букви криптотексту, що знаходяться на непарних позиціях, шукаються в другому рядку, а ті, що знаходяться на парних позиціях – в третьому рядку.

Приклад 5. Вхідне слово hello в шифротексті буде виглядати як SHLJV, що суттєво ускладнює застосування для атаки частотного аналізу мови.

На практиці можна використовувативати не два, а аж до п'яти різних алфавітів шифротексту, багаторазово збільшуючи простір ключів. Дійсно, легдо підрахувати, що якщо ми беремо символи з п'яти заміщуючих наборів, то число можливих ключів $(26!)^5 \sim 2^{441}$. Проте в цьому випадку ключ - послідовність з 26*5 = 130 літер. Для середнього користувача початку XIX століття такий ключ був занадто великим, щоб запам'ятати його.

Незважаючи на зазначений недолік, найвідоміші шифри XIX століття грунтувалися саме на описаному принципі. До них відносяться модифікований шифр Цезаря та шифр Віженера.

Модифікований шифр Цезаря

Модифікований шифр Цезаря - один з варіантів поліалфавітного шифру зсуву.

Використовується потоковий алгоритм шифрування, який полягає у такому:

---

[1] Тут перший рядок - англійський алфавіт, а другиі і третій - перший і другий алфавіти шифротекста.

1. Як і у випадку шифру зсуву, перенумеруємо 26 літери вхідного алфавіту, починаючи з 0:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

2. Створюємо секретний ключ – будь-яке слово, яке наивають **гаслом.**
3. Гасло підписується ють під повідомленням з повторенням.
4. Щоб отримати шифрований текст, складають номер чергової букви з номером відповідної букви ключа. Якщо отримана сума більше 26, то з неї віднімають 26. В результаті отримують послідовновательность чисел від 1 до 31. Знову замінюючи числа цієї последовательности відповідними буквами, отримують шифрований текст.

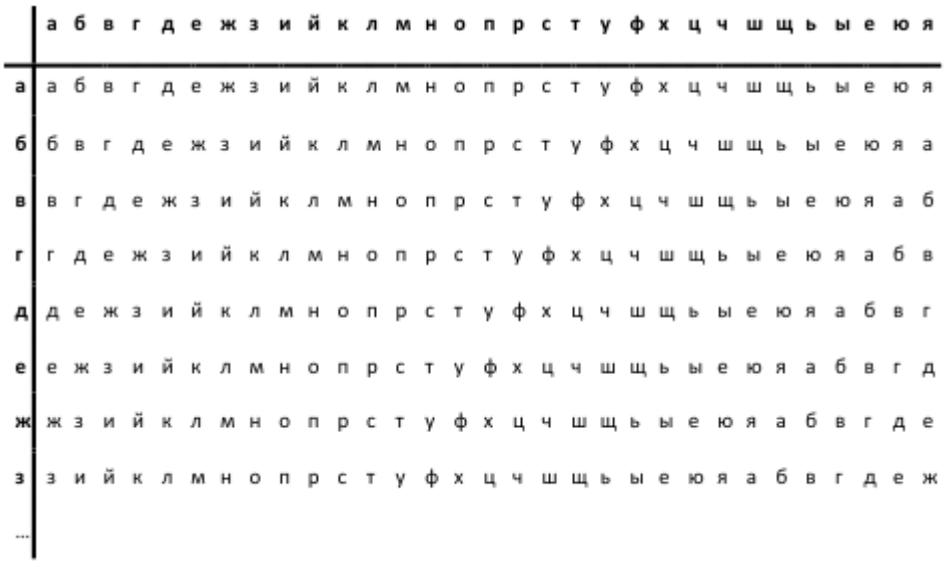
Приклад 6. Якщо гаслом є слово sesame, то шифрування виглядає так:

t	h	i	s	i	s	a	t	e	s	t	m	e	s	s	a	g	e
19	7	8	18	8	18	0	19	4	18	19	12	4	18	18	0	6	4
s	e	s	a	m	e	s	e	s	a	m	e	s	e	s	a	m	e
18	4	18	0	12	18	18	4	18	0	12	18	18	4	18	0	12	18
L	L	A	S	U	W	S	X	W	S	F	Q	W	W	K	A	S	I
11	11	0	18	20	22	18	23	22	18	5	16	22	22	10	0	18	8

Шифр Віженера

Шифр Віженера є поліалфавітним блоковим шифром. Шифрування і розщифрування грунтується на використанні **таблиці Віженера**. В загальному випадку першим її рядком є алфавіт відкритого тексту, а першим стовпчиком – алфавіт ключа. Тіло таблиці складається з циклічно зсунутих алфавітів, причому його перший рядок може бути довільним змішаним алфавітом.

В найпростішому випадку використання незмішаного алфавіту для самоключа таблиця Віженера матиме такий вигляд (складена з 31 літери російського алфавіту – без літер **ё** і **ъ**) :



	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ы	е	ю	я
а	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ы	е	ю	я
б	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ы	е	ю	я	а
в	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ы	е	ю	я	а	б
г	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ы	е	ю	я	а	б	в
д	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ы	е	ю	я	а	б	в	г
е	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ы	е	ю	я	а	б	в	г	д
ж	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ы	е	ю	я	а	б	в	г	д	е
з	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ы	е	ю	я	а	б	в	г	д	е	ж
…																															

Щоб зашифрувати повідомлення:

1. Вибирається слово - гасло і підписується з повторенням над буквами відкритого повідомлення.
2. Щоб отримати шифрований текст, знаходять черговий знак лозунгу, починаючи з першого у вертикальному алфавіті, а відповідний йому знак повідомлення – в горизонтальному.

Приклад 7. Вибираємо гасло - математика. Знаходимо стовпець, що відповідає букві "м" гасла, а потім рядок, відповідну букві "к". На перетені виділених стовпця і рядка знаходимо букву "ц".

м	а	т	е	м	а	т	и	к	а	м	а	т	е	м	а	т	и	к	а	м	а	т	е	м	а
к	р	и	п	т	о	г	р	а	ф	и	я	с	е	р	ь	е	з	н	а	я	н	а	у	к	а

Продовжуючи далі, знаходимо весь зашифрований текст:

Ц Р Ь Ф Я О Х Ш К Ф Ф Я Д К Э Ь Ч П Ч А Л Н Т Ш Ц А.

У 1888 р. француцз маркіз де Віарі довів, що щифрування по Віженеру відтворює алгебраїчна формула

$$C=(p+G) \bmod 26 ,$$

де C – будь-яка буква шифротексту, p– будь-яка буква відкритого тексту, G – будь-яка буква гами, а заміна букв числами здійснюється за таблицею:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Процес розшифрування відтворює формула

$$p = (C – G +26) \bmod 26 ,$$

Використання рівняння шифрування дозволило відмовитись від громіздкої таблиці Віженера. Згодом лозунгова гама стала довільною числовою послідовністю, а шифр з вказаним рівнянням шифрування став називатись **шрифтом гаміювання**.

Як бачимо, у простому варіанті з коротким ключовим словом і з таблицею, що складається зі звичайних алфавітів, шифр Віженера є еквівалентним модифікованому шифру Цезаря. В літературі часто саме цей найпростіший варіант і називають шифром Віженера.

Запитання для самоконтролю

1. Що є предметом вивчення криптографії?
2. Які властивості інформації захищаються криптографічними засобами?
3. Що називається ключем шифрування?
4. Яку архітектуру має криптографічна система?
5. Що таке шифр?
6. У чому полягає принцип Керкхоффса?
7. Коли в криптографії почали використовуватись наукові підходи?
8. Яке призначення мали нуменклатори?
9. Як класифікуються алгоритми шифрування?
10. Які шифри називаються історичними?
11. У чому сутність шифру маршрутної перестановки?

12. Як здійснюється шифрування за допомогою стовпчикової перестановки?
13. Як влаштований шифр матричної перестановки?
14. Як здійснюється шифрування за допомогою шифру заміни?
15. У чому сутність поліалфавітної заміни?
16. Як здійснюється шифрування за допомогою модифікованого шифру Цезаря?
17. Як використовується шифр Віженера?

Результати навчання

Знання тих властивостей інформації, захист яких забезпечується криптографічними засобами, , розуміння загальних підходів до побудови симетричних криптосистем, аналіз переваг і недоліків основних класичних шифрів і їх класифікації, знання базових перетворень в шифрах класичної криптографії,необхідних для розуміння основних підходів до побудови перестановочних шифрів, принципів побудови потокових шифрів.

Розділ 3. Криптографічна стійкість шифрів

3.1. Поняття криптографічної стійкості

Криптографічна стійкість (або крипостійкість) - здатність криптографічного алгоритму протистояти можливим атакам на нього.

Стійким вважається алгоритм, який для успішної атаки вимагає від противника:

- недосяжних обчислювальних ресурсів,
- недосяжного обсягу перехоплених відкритих і зашифрованих повідомлень чи
- такого часу розкриття, що по його закінченню захищена інформація буде вже не актуальна, і т. д.

Стійкість не можна підтвердити, її можна тільки спростувати зломом.

Для оцінки стійкості шифру використовують оцінку обчислювальної складності алгоритму успішної атаки на криптосистему.

Рівень криптостійкості (англ. security level) - показник криптостійкості алгоритму, пов'язаний з обчислювальною складністю виконання успішної атаки на криптосистему.

Зазвичай рівень криптостійкості вимірюється в бітах. N-бітний рівень криптостійкості криптосистеми означає, що для її злому потрібно виконати n обчислювальних операцій. Наприклад, якщо симетрична криптосистема зламуються не швидше, ніж за повний перебір значень n-бітного ключа, то кажуть, що рівень криптостійкості дорівнює n.

Складність алгоритмів зазвичай оцінюють за часом виконання, але не менш важливі й інші показники - вимоги до обсягу пам'яті, вільного місця на диску.

В теорії алгоритмів **обчислювальна складність алгоритму** - це функція, яка визначає залежність обсягу роботи, виконуваної деяким алгоритмом, від розміру вхідних даних.

У загальному випадку складність алгоритму можна оцінити по порядку величини. Алгоритм має складність O (f (n)), якщо при збільшенні розмірності

вхідних даних n, час виконання алгоритму зростає з тією ж швидкістю, що і функція f (n).

Використання великої літери O (або так звана O-нотація) прийшло з математики, де її застосовують для порівняння асимптотичної поведінки функцій. Формально O (f (n)) означає, що час роботи алгоритму (або обсяг займаної пам'яті) росте в залежності від обсягу вхідних даних не швидше, ніж деяка константа, помножена на f (n).

Типові приклади використання O-нотації:

O (n) - лінійна складність. Таку складність має, наприклад, алгоритм пошуку найбільшого елемента в невідсортованому масиві. Нам доведеться пройтися по всіх n елементах масиву, щоб зрозуміти, який з них максимальний.

O (log n) - логарифмічна складність. Найпростіший приклад - бінарний пошук. Якщо масив відсортований, ми можемо перевірити, чи є в ньому якесь конкретне значення, шляхом поділу навпіл. Перевіримо середній елемент, якщо він більше шуканого, то відкинемо другу половину масиву - там його точно немає. Якщо ж менше, то навпаки - відкинемо початкову половину. І так будемо продовжувати ділити навпіл, в результаті перевіримо log n елементів.

O (n^2) - квадратична складність. Таку складність має, наприклад, алгоритм сортування вставками. У канонічної реалізації він вдає із себе два вкладених циклу: один, щоб проходити по всьому масиву, а другий, щоб знаходити місце чергового елементу уже відсортованої частини. Таким чином, кількість операцій буде залежати від розміру масиву як n * n.

Для наочності наводимо час виконання алгоритму з певною складністю в залежності від розміру вхідних даних при швидкості виконання 10^6 операцій в секунду (Таблиця 3.1).

Таблиця 3.1. **Час виконання алгоритму з певною складністю в залежності від розміру вхідних даних при швидкості виконання 10^6 операцій в секунду**

розмір / складність	10	20	30	40	50	60
n	0,00001 сек.	0,00002 сек.	0,00003 сек.	0,00004 сек.	0,00005 сек.	0,00005 сек.
n^2	0,0001 сек.	0,0004 сек.	0,0009 сек.	0,0016 сек.	0,0025 сек.	0,0036 сек.
n^3	0,001 сек.	0,008 сек.	0,027 сек.	0,064 сек.	0,125 сек.	0,216 сек.
n^5	0,1 сек.	3,2 сек.	24,3 сек.	1,7 хв.	5,2 хв.	13 хв.
2^n	0,0001 сек.	1 сек.	17,9 хв.	12,7 днів	35,7 століть	366 століть
3^n	0,059 сек.	58 хв.	6,5 років	3855 століть	$2x10^8$ століть	$1,3x10^{13}$ століть

Множина задач, для вирішення яких існують алгоритми, схожі за обчислювальною складністю, утворює **клас складності**.

Розрізняють такі основні класи складності:

- **Клас P** - вміщує всі ті проблеми, вирішення яких вважається «швидким», тобто поліноміально залежать від розміру входу (наприклад, сортування, пошук, з'ясування зв'язності графів і т.п.).
- **Клас NP** - містить задачі, які не в змозі вирішити за поліноміальну кількість часу (наприклад, факторизація, дискретне логарифмування та інші).

3.2. Межі застосування «грубої сили» до атак на шифри

Найбільш загальний вид атаки на шифр – метод повного перебору (або метод «грубої сили»).

Повний перебір (англ. brute force) - загальний метод вирішення завдань шляхом перебору всіх можливих потенційних рішень.

У криптографії на обчислювальній складності повного перебору ґрунтується оцінка криптостойкости шифрів. Зокрема, шифр вважається крипостійким, якщо не існує методу «злому» істотно більш швидкого, ніж повний перебір всіх ключів. Криптографічні атаки, засновані на методі повного перебору, є найбільш універсальними, але і найдовшими.

Стійкість до brute-force атаки визначає використовуваний в криптосистемі ключ шифрування. Так, зі збільшенням довжини ключа складність злому цим методом зростає експоненціально. У найпростішому випадку шифр довжиною в n бітів зламуються, в найгіршому випадку, за час $O(2^n)$.

Взагалі будь-яка задача з класу NP може бути вирішена повним перебором, але це вимагатиме експоненціального часу роботи.

Існують способи підвищення стійкості шифру до «brute force», наприклад заплутування (обфускація) шифрованих даних, що робить нетривіальним відмінність зашифрованих даних від незашифрованих.

Виходячи з фізичних міркувань можна показати, що існує можливість вибрати таку довжину ключа, що таку атаку методом «грубої сили» неможливо буде виконати в принципі, безвідносно до потужностей наявної обчислювальної техніки.

Доведення 1 (термодинамічний підхід).

За законами термодинаміки поглинання енергії при здійсненні незворотного перетворення має порядок ~kT, де $k=1.4*10^{-23}$ Дж/К, Т – температура зовнішнього середовища.

Потужність сонячного випромінення $Pc\sim3.86 *10^{26}$ Вт. Час існування Всесвіту t_c ~14 млд. років$\sim14*10^9$ років ~ $4*10^{19}$ с. Температура Сонця $T_c\sim10^6$ К.

Тоді витрати на одну обчислювальну операцію $\sim kT=1.4*10^{-23}*10^6\sim1.4*10^{-17}$ (Дж).

Вся енергія виділена Сонцем за час його існування Всесвіту $E= Pc* t_c \sim 3.86 *10^{26}*4*10^{19} \sim1.6* 10^{46}$ (Дж). Тому можлива кількість виконаних операцій $N=E/ kT\sim1.6* 10^{46} / 1.4*10^{-17} \sim 10^{63}$.

А це означає, при довжині ключа шифрування $L\geq63$ перебір всіх варіантів не можна здійснити за час існування Всесвіту.

Доведення 2 (гео-протонний підхід).

Маса Землі $M \sim 6*10^{24}$ кг, маса протону $m \sim 1.6*10^{-27}$ кг; тобто Земля містить $N=M/m \sim 6*10^{24} / 1.6*10^{27} \sim 3.8*10^{51}$ протонів.

Співставим кожному протону комп'ютер і вважатимемо, що на виконання однієї операції в ньому витрачається час t, за який світловий промінь (швидкість $c \sim 3*10^{10}$ м/с) проходить відстань, рівну діаметру протона ($d \sim 10^{-15}$ м): $t= d/c \sim 10^{-15}/3*10^{10} \sim 1/3*10^{-25}$ (с).

Це означає, що швидкість роботи одного такого уявного комп'ютера $v= t^{-1} \sim 3*10^{25}$ операцій/с.

Сумарна швидкодія всіх «протонних» комп'ютерів $V=N*v \sim 3.8*10^{51} * 3*10^{25} \sim 10^{76}$ операцій/с. За весь час існування Всесвіту $t_c \sim 4*10^{19}$ с можна виконати не більше ніж $M=V* t_c \sim 10^{76} * 4*10^{19} \sim 10^{95}$ операцій.

Знову ж таки, це означає, при довжині ключа шифрування $L \geq 95$ перебір всіх варіантів не можна здійснити за час існування Всесвіту.

Резюмуючи, робимо наступний висновок: «хорошим» ключем шифрування (тобто таким, який не можна підібрати методом «грубої сили») є ключ довжиною не менше $L \geq 100$.

3.3. Абсолютна криптостійкість шифрів

Абсолютна крипостійкість означає, що:

- результатом розшифрування може бути будь-який текст,
- не існує ніякого способу перевірки, що розшифрування виконане правильно.

Доведення існування абсолютно стійких алгоритмів шифрування було виконано Клодом Шенноном та опубліковано в роботі «Теорія зв'язку в секретних системах» (1946 рік). Там же визначені вимоги до такого роду систем:

1. Ключ генерується для кожного повідомлення (кожен ключ використовується один раз).

2. Ключ статистично надійний (тобто ймовірності появи кожного з можливих символів однакові, символи в ключовий послідовності незалежні і випадкові).
3. Довжина ключа дорівнює або більше довжини повідомлення.

Стійкість цих систем не залежить від того, якими обчислювальними можливостями володіє криптоаналітик.

Майже всі використовувані на практиці шифри характеризуються як умовно надійні, оскільки вони можуть бути розкриті в принципі при наявності необмежених обчислювальних можливостей. Абсолютно надійні шифри не можна зруйнувати навіть при наявності необмежених обчислювальних можливостей.

Єдиним відомим шифром, який задовольняє вимогам абсолютної криптостійкості, є шифр Вернама.

Шифр Вернама (інша назва: англ. One-time pad - **схема одноразових блокнотів**) названий на честь телеграфіста AT & T Гільберта Вернама, який в 1917 році побудував телеграфний апарат, що виконував цю операцію автоматично - треба було тільки подати на нього стрічку з ключем.

Для шифруівння відкритий текст «сумується» з ключем (який називається **одноразовим блокнотом** або **шифроблокнотом**) аналогічно розширеному шифру Цезаря. Але при цьому ключ повинен задовольняти трьом критично важливими умовам:

1. Бути істинно випадковим.
2. Співпадати за розміром з заданим відкритим текстом.
3. Застосовуватись тільки один раз.

Будемо вважати, що число символів розширеного алфавіту, тобто сукупності літер, а також знаків пунктуації та пропусків між словами, дорівнює N. Занумерувавши всі символи розширеного алфавіту числами від 0 до N, текст, що передається, можна розглядати як послідовність $\{a_n\}$ чисел множини A = {0,1,2, ..., N}.

Маючи випадкову послідовність $\{k_n\}$ з чисел множини A тієї ж довжини що і ключ, складаємо по модулю N число a_n переданого тексту з відповідним числом k_n ключа $a_n + k_n \equiv c_n \pmod N$, $0 \le c_n \le N$, одержимо послідовність $\{c_n\}$ знаків шифрованого тексту.

Щоб отримати переданий текст, можна скористатися тим ж ключем: $a_n \equiv c_n - k_n \pmod N$, $0 \le a_n \le N$.

У двох абонентів, що знаходяться в секретному листуванні, є два однакових блокнота, складених з відривних сторінок, на кожній з яких надрукована таблиця з випадковими числами або буквами, тобто випадкова послідовність чисел з множини A. Таблиця має тільки дві копії: одна використовується відправником, інша - одержувачем.

Приблизна сторінка шифроблокноту показана на Рис.3.1.

```
ZDXWWW EJKAWO FECIFE WSNZIP PXPKIY URMZHI JZTLBC YLGDYJ
HTSVTV RRYYEG EXNCGA GGQVRF FHZCIB EWLGGR BZXQDQ DGGIAK
YHJYEQ TDLCQT HZBSIZ IRZDYS RBYJFZ AIRCWI UCVXTW YKPQMK
CKHVEX VXYVCS WOGAAZ OUVVON GCNEVR LMBLYB SBDCDC PCGVJX
QXAUIP PXZQIJ JIUWYH COVWMJ UZOJHL DWHPER UBSRUJ HGAAPR
CRWVHI FRNTQW AJVWRT ACAKRD OZKIIB VIQGBK IJCWHF GTTSSE
EXFIPJ KICASQ IOUQTP ZSGXGH YTYCTI BAZSTN JKMFXI RERYWE
```

Рис.3.1. Приклад сторінки шифроблокноту

Відправник свій текст шифрує зазначеним вище способом за допомогою першої сторінки блокнота. Зашифрувавши повідомлення, він знищує використувану сторінку і відправляє її іншому абоненту. Одержувач шифрованого тексту розшифровує його і також знищує використаний лист блокнота.

Неважко бачити, що одноразовий шифр не можна розкрити в принципі, так як символ у тексті може бути замінений будь-яким іншим символом і цей вибір абсолютно випадковий.

3.4. Основи квантової криптографії

Квантова криптографія – наука, що вивчає методи захисту систем зв'язку і базується на принциповій непорушності закономірностей квантової фізики,

об’єкти якої забезпечують процеси безпечного передавання інформації між легітимними користувачами.

Основними. засадами є:

- постулат вимірювання (результат принципу невизначеності Гейзенберґа) та
- теорема про заборону клонування.

Принцип невизначеності – це фундаментальна нерівність (відношення невизначеностей), згідно з якою усі динамічні параметри поділяють на дві групи: перша – часові та просторові координати; друга – імпульс та енергія. При чому неможливо одночасно виміряти параметри з різних груп (напр., місцерозташування та енергію об’єкта).

Теорема про заборону клонування стверджує неможливість створення точних (на 100 %) копій квантових об’єктів.

Ці властивості не притаманні класичним системам зв’язку і забезпечують абсолютну й безумовну стійкість квантової криптографії. В останні роки значний інтерес викликають системи захисту інформації на базі методів квантової криптографії. Виокремлюють такі напрями, як квантовий розподіл ключів (КРК), квантовий прямий безпечний зв’язок (КПБЗ), квантове розділення секрету (КРС), квантовий цифровий підпис (КЦП) та кванттова стеганографія.

Ідею використання квантових об’єктів для захисту інформації вперше запропонував С. Вайснер (1970 рік). В 1984 році Ч. Беннет (компанія IBM) та Ж. Брассар (Монреальський університет) розвинули її і запропонували перший протокол квантової криптографії BB84, що став альтернативним вирішенням проблеми розподілу ключів шифрування.

У ньому для передачі даних використовуються фотони, поляризовані в чотирьох різних напрямках, в двох базисах - під кутом 0 і 90 градусів (позначається знаком +) або 45 і 135 градусів (позначається знаком х).

Відправник повідомлення Аліса поляризує кожен фотон в випадково обраному базисі, а потім відправляє його одержувачу Бобу. Боб вимірює кожен

фотон, теж в випадково обраному базисі. Після цього Аліса по відкритому каналу повідомляє Бобу послідовність своїх базисів, і Боб відкидає неправильні (не співпали) базиси і повідомляє Алісі, які дані «не пройшли». При цьому самі значення, отримані в результаті вимірів, вони по відкритому каналу не обговорюють. Якщо шпигун Єва захоче перехопити секретний ключ, вона повинна буде вимірювати поляризацію фотонів. Оскільки вона не знає базису, то повинна буде визначати його випадковим чином. Якщо базис буде визначено неправильно, то Єва не отримає вірних даних, а крім того, змінить поляризацію фотона. Помилки, що з'являться, відразу виявлять і Аліса, і Боб.

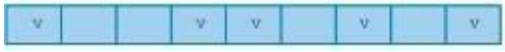
Описану схему можна проілюструвати наступним чином:

Аліса надсилає фотони, що мають одну з чотирьох можливих поляризацій, яку вона обирає випадково.

Для кожного фотона Боб обирає випадково тип вимірювання: прямолінійна поляризація (+), або діагональна (x).

Боб записує та таємно зберігає отримані результати вимірювань.

Боб відкрито оголошує типи вимірювань, які він обрав для кожного фотона, а Аліса повідомляє йому, які вимірювання були правильними.



Аліса і Боб зберігають всі результати, що були отримані у тих випадках, коли Боб застосував правильний тип вимірювання. Ці результати переводять у біти (0 і 1), послідовність яких і є результатом квантового розподілу ключа.

Протокол BB84 вважають основоположником протоколів квантової криптографії. Нині існують вже кілька десятків декілька подібних систем, які вийшли на світовий ринок і позиціонуються як надійні системи розподілу ключів та криптографічного захисту інформації.

Першим у світі комерційним рішенням була система QPN Security Gateway (QPN-8505), запропонована американською компанією MagiQ. Вона пропонує захист VPN за допомогою КРК (бл. 100 ключів 256 біт за секунду на відстань до 140 км) та інтегроване шифрування. За допомогою цієї системи можна організувати захищену квантову мережу.

Іншим ефективним рішенням є система КРК Clavis швейцарської компанії ID Quantique. З її допомогою можна здійснювати захищений розподіл ключів шифрування між двома абонентами на відстань до 100 км. Крім того, компанія пропонує квант. систему Cerberis. Ця система може розподіляти ключі на відстань до 50 км.

На початку 21 ст. британською компанією Toshiba Research Europe Ltd представлено систему КРК Quantum Key Server, яка відрізняється простотою своєї архітектурири та забезпечує генерацію близько 100 ключів (довжиною 256 біт) за секунду та їхнє одностороннє передавання від передавача до приймача.

Інша британська компанія QinetiQ запропонувала першу у світі комп'ютерну мережу, що використовує квантову криптографію, – Quantum Net (Qnet). Максимальна довжина ліній зв'язку цієї мережі становить 120 км.

Запитання для самоконтролю

1. Що розуміється під криптографічною стійкістю шифру?
2. Як визначається обчислювальна складність алгоритму?
3. Якою має бути довжина ключа, щоб атаку методом «грубої сили» неможливо було виконати в принципі? З яких міркувань це випливає?
4. Які шифри називаються абсолютно криптостійкими?
5. За якими ознаками можна визначити абсолютно криптостійкі шифри?
6. Як працює схема одноразових блокнотів?
7. Що таке квантова криптографія?
8. Яке призначення протоколу BB84? Опишіть принцип його роботи.

Результати навчання

Знання поняття і ознак абсолютної криптостійкості, уміння оцінювати стійкість шифрів, знання принципів функціонування і використання квантової криптографії.

Розділ 4. Блокові шифри як основа сучасних криптосистем

4.1. Блокові алгоритми і режими шифрування

Як зазначалось раніше, головний недолік абсолютно криптостійкого шифру одноразових блокнотів – велика довжина ключа, яка має бути не меншою від довжини повідомлення. Через це використовувати його на практиці складно.

Для побудови стійкого шифру, зручного для практичного використання, можна запропонувати такі підходи:

1. **Блоковість**. Вибираємо довжину ключа меншою за довжину повідомлення, розбиваємо повідомлення на окремі блоки і шифруємо кожний з них (крім, можливо, останнього) шляхом сумування з ключем по модулю два.
2. **Режими шифрування**. Для збільшення стійкості блокового шифрування можна забезпечити використання при шифруванні кожного наступного блоку результатів шифрування попереднього блоку (наприклад, в якості нового ключа шифрування для блоку). Тоді зловмисник не зможе розшифрувати блок криптотексту, доки не розшифрує всі попередні блоки.
3. **Багатораундовість**. Додатково збільшити стійкість блокового шифрування можна з рахунок багаторазового виконання шифрування кожного блоку за умови, що функція шифрування є нелінійним перетворенням.

В блокових алгоритмах вхідна послідовність розбивається на **блоки** – ділянки певної довжини (найчастіше, по 64 біти). Якщо довжина відкритого тексту виявляється некратною довжині блоку, застосовується операція **доповнення** (padding) останнього блоку до необхідної довжини, яка полягає у дописуванні необхідної кількості нулів або випадкового набору символів (Рис.4.1).

Рис.4.1. Представлення даних в блоковому алгоритмі

Криптографічне перетворення в блокових алгоритмах шифрування здійснюється над кожним блоком окремо (Рис.4.2). Його сутність полягає у застосуванні до блока багаторазово математичного перетворення. Внаслідок цього результуюче перетворення виявляється криптографічно більш сильним, ніж перетворення над окремо взятим блоком. Метою таких перетворень є створення залежності кожного біту блоку шифротексту від кожного біту ключа і кожного біту відкритого тексту:

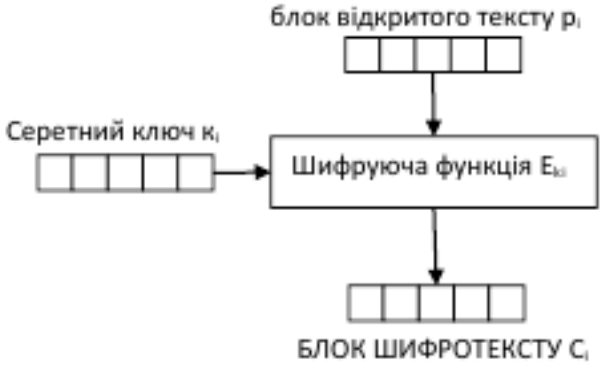


Рис.4.2. Схема шифрування блоку даних в блоковому алгоритмі

Зашифрований блок може використовуватися для шифрування наступного, в результаті чого кожний блок отримає контекст, що властивий всьому повідомленню (Рис.4.3). Такі механізми шифрування використовуються для уникнення від деяких атак, заснованих на стиранні або вставки блоків, і визначаються відповідним *режимом шифрування*.

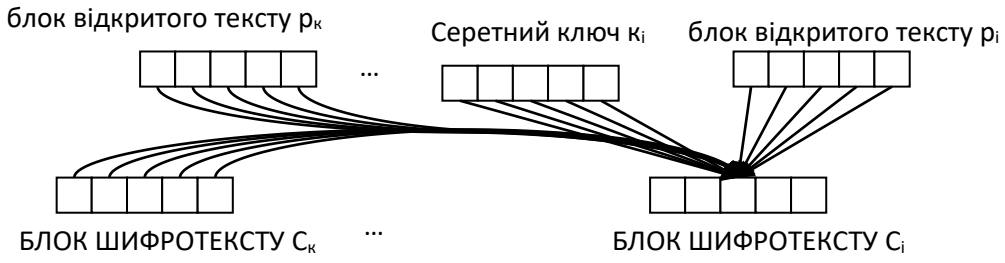


Рис.4.3. Механізм режимів шифрування

Режим шифрування – метод застосування блокового шифру, в якому для забезпечення вищого рівня криптостійкісті шифрування блоків вхідного повідомлення здійснюється з використання блоків криптотексту.

Розрізняють п’ять основних режимів шифрування:

1. **ECB** (Electronic Codebook Mode) – режим електронної кодової книги.
2. **CBC** (Cipher Block Chaining Mode) – режим зціплення блоків по криптотексу.
3. **CFB** (Cipher Feedback Mode) – режим з оберненим зв'язком по криптотексу.
4. **OFB** (Output-Feedback Mode) – режим з оберненим зв'язком по виходу.
5. **CTR** (Coutner) – режим з лічильником.

Блокові алгоритми шифрування сьогодні є основним засобом криптографічного захисту інформації. Основні переваги блокових алгоритмів шифрування:

- Висока швидкість шифрування/розшифрування.
- Висока гарантована стійкість, яка до того ж може бути доведена математично.
- Можливість ефективної програмної реалізації.

4.2. Режим електронної кодової книги (ECB)

В цьому режимі блоки відкритого тексту шифрують ся незалежно від інших за допомогою одного й того ж ключа.

Шифрування описується рівнянням: $C_i=E_{ki}(p_i)$ для $i=1\div N$, де C_i та p_i – блоки відповідно зашифрованого і відкритого тексту, E_{ki} – функція шифрування.

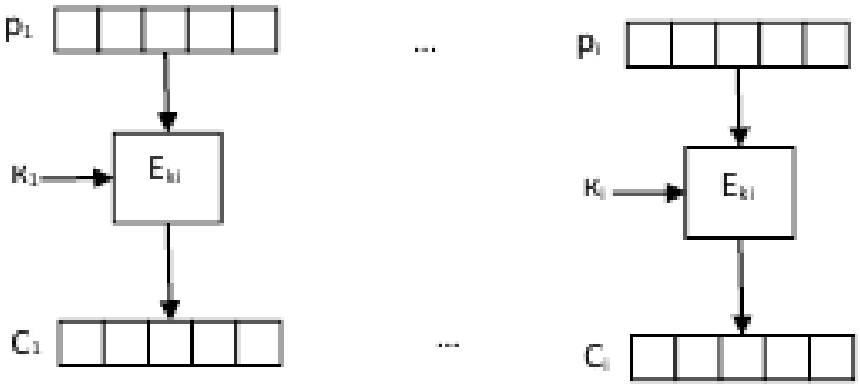


Рис.4.3. Схема шифрування в режимі ECB

Розшифрування здійснюється за допомогою функції розшифрування $p_i=D_{ki}(C_i)$:

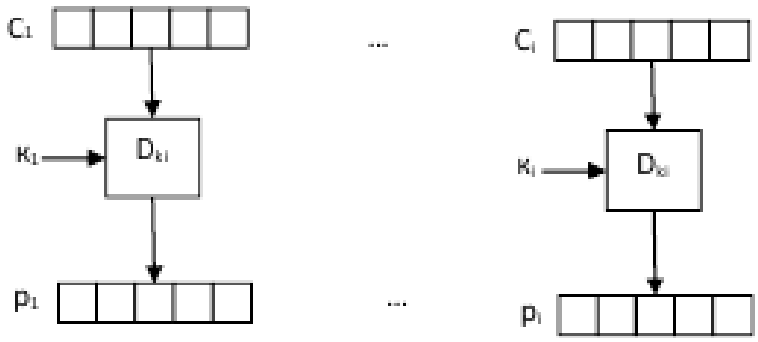


Рис.4.3. Схема розшифрування в режимі ECB

Такий режим є криптографічно слабким. Основні недоліки:

- Режим не є критичним по відношенню до вставки і втрати блоків в процесі передачі.
- Однакові блоки будуть перетворюватись в такі ж самі, що може дати ключ для аналізу вмісту повідомлення.

Перевагою режиму є те, що шифрування (розшифрування) блоків може виконуватись паралельно.

4.3. Режим зціплення блоків по криптотексу (CBC)

В цьому режимі шифрування кожний блок відкритого тексту,крім першого, складається по модулю 2 (операція XOR) з попереднім зашифрованим блоком. Шифрування першого блоку здійснюється за допомогою початкового **вектора ініціалізації** (**синхропосилки**), яка передається по відкритому каналу передачі даних.

Шифрування описується рівнянням: $C_i= E_{ki}(p_i \text{ XOR } C_{i-1})$ для $i=2\div N$, C_0 – синхропосилка (Рис.4.4).

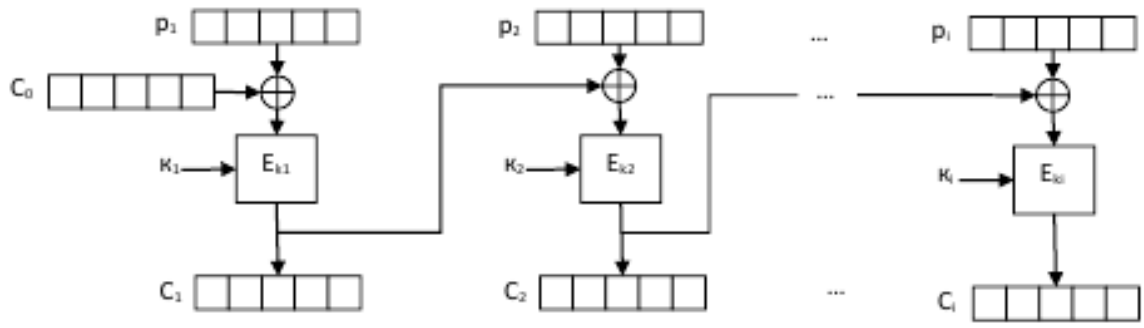


Рис.4.4. Схема шифрування в режимі CBC

Розшифрування описується рівнянням: $p_i = C_{i-1} XOR D_{ki}(C_i)$ для $i=2 \div N$, C_0 – синхропосилка (Рис.4.5).

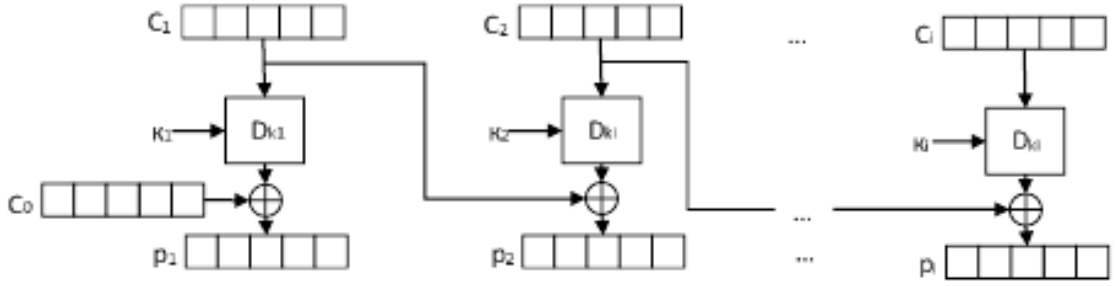


Рис.4.5. Схема розшифрування в режимі CBC

Важливо відмітити, що останній блок шифротексту залежить від ключів, синхропосилки і всіх бітів відкритого тексту. Тому його можна розглядати в якості ідентифікатора повідомлення. Цей ідентифікатор широко використовується для автентифікації повідомлення та відправника і називається **кодом автентифікації повідомлення** або **MAC** (Message Authentication Code).

Коди автентифікації повідомлень не забезпечують секретність, але гарантують автентифікацію і цілісність. Вони дають впевненість, що повідомлення прийшло саме від тієї людини, який позначений як автор, і що повідомлення по дорозі не змінилося. Хто завгодно може прочитати повідомлення. Але той, хто знає ключ MAC, може впевнитися, що воно не було змінено.

Для використання MAC Аліса спочатку домовляється з Бобом про ключ. Потім, коли вона хоче послати Бобу повідомлення, вона обчислює MAC повідомлення і додає його до повідомлення. Коли Боб отримує повідомлення,

він обчислює його MAC і порівнює його з тим значенням MAC, яке прислала Аліса. Якщо вони збігаються, то він може бути впевнений у двох речах: повідомлення дійсно прийшло від Аліси і це повідомлення незмінене. Банки використовують таку просту систему автентифікації вже кілька десятиліть.

MAC постійно використовуються в Інтернет, наприклад, у протоколі IPsec, щоб гарантувати, що IPпакети не були змінені в проміжку між відправленням і прибуттям на місце призначення. Їх використовують у всіх можливих протоколах міжбанківських переказів для встановлення автентичності повідомлень.

Переваги режиму CBC:

- Режим CBC є критичним по відношенню до вставки і втрати блоків в процесі передачі.
- Забезпечується автентифікація повідомлення та відправника на основі MAC.

Недоліком режиму є те, що шифрування (розшифрування) не піддається розпаралеленню.

4.4. Режим з оберненим зв’язком по криптотексу (CFB)

В цьому режимі алгоритму блокового шифрування черговий блок відкритого тексту шумується по модулю 2 з блоком попереднього шифротексту, але тільки після того, як він буде додатково оброблений за допомогою функції шифрування.

Важливо відмітити, що в цьому режимі як для шифрування, так і розшифрування використовується тільки функція шифрування, а обернена до неї функція розшифрування взагалі не потрібна.

Шифрування описується рівняннями: $C_i = E_{ki}(C_{i-1})$ XOR p_i для $i=2 \div N$, C_0 – синхропосилка (Рис.4.6).

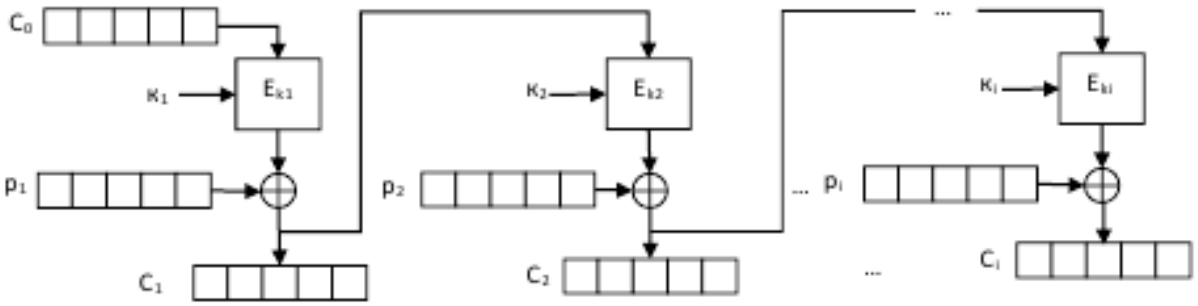


Рис.4.6. Схема шифрування в режимі CFB

Розшифрування описується рівнянням: $p_i = E_{ki}(C_{i-1})$ XOR C_i для i=2÷N, C_0 – синхропосилка (Рис.4.7).

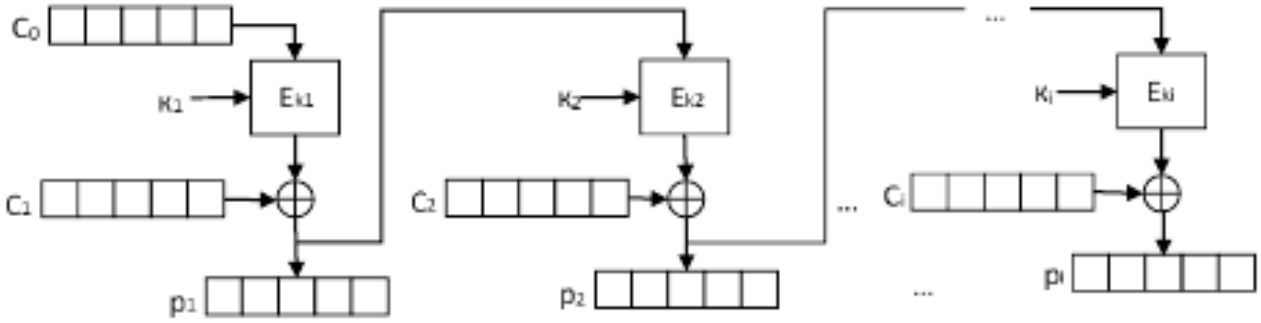


Рис.4.7. Схема розшифрування в режимі CFB

Переваги режиму CFB:

- Критичність по відношенню до вставки і втрати блоків в процесі передачі.
- Використовується тільки функція шифрування.
- Симетрія операцій шифрування/розшифрування

Недоліком режиму є те, що шифрування (розшифрування) не піддається розпаралеленню.

4.5. Режим з оберненим зв’язком по виходу (OFB)

В цьому режимі функція шифрування викликається до сумування по модулю 2 з блоком відкритого тексту, але на її вхід подається не шифротекст, а ті ж вхідні дані. Можна сказати, що при використанні даного режиму блочний шифр перетворюється в синхронний шифропотік, тобто генеруються ключові блоки, які є результатом сумування по модулю 2 з блоками відкритого тексту.

Операції шифрування і розшифрування виявляються симетричними і описуються рівняннями $C_i = p_i$ XOR O_i для i=1÷N (шифрування, Рис.4.8) і $p_i = C_i$ XOR O_i (розшифрування, Рис.4.9) для i=1÷N, де $O_1 = E_{ki}(C_0)$, $O_i = E_{ki}(O_{i-1})$ для i=1÷N.

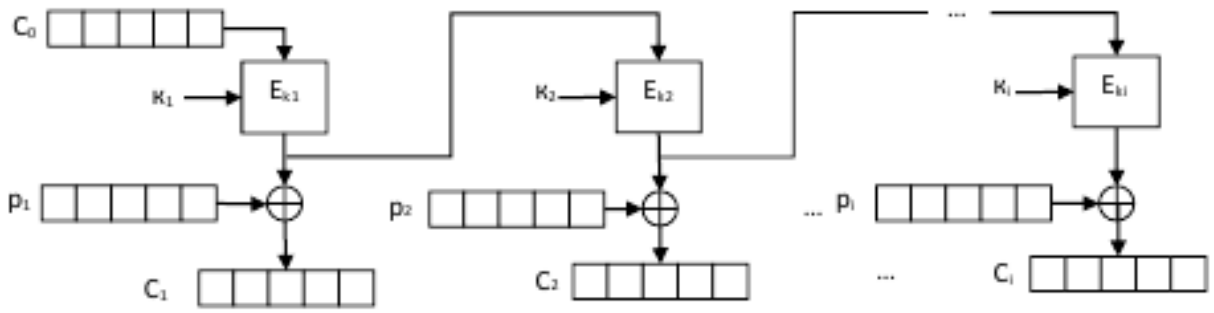


Рис.4.8. Схема шифрування в режимі OFB

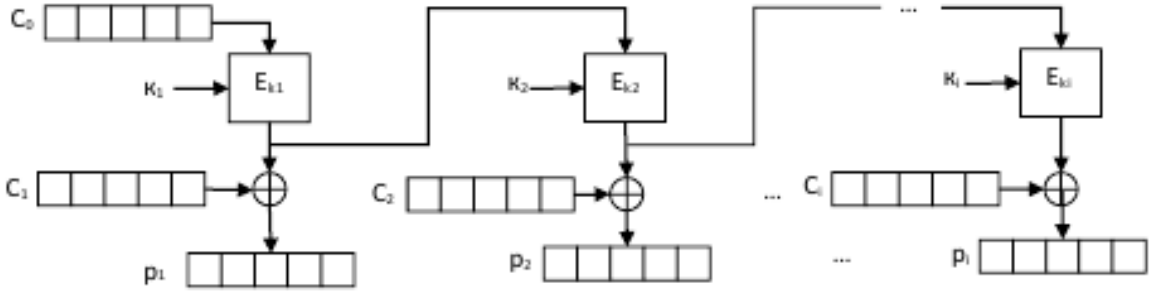


Рис.4.9. Схема розшифрування в режимі OFB

Переваги режиму CFB:

- Критичність по відношенню до вставки і втрати блоків в процесі передачі.
- Використовується тільки одна функція шифрування, а обернена до неї функція розшифрування взагалі не потрібна.
- Операції шифрування і розшифрування виявляються симетричними.

Паралельне виконання операцій шифрування (розшифрування) блоків в такому режимі неможливе.

4.6. Режим з лічильником (CTR)

В даному режимі на кожній i-ітерації на вхід шифрування подається деяке випадкове значення T_i. Всі T_i мають бути різними, тому генератор таких значень називається лічильником.

Операції шифрування і розшифрування теж виявляються симетричними і описуються такими рівняннями $C_i = p_i$ XOR O_i для i=1÷N (шифрування, Рис.4.10) і $p_i = C_i$ XOR O_i (розшифрування, Рис.4.11) для i=1÷N, де $O_i = E_{ki}(T_i)$ для i=1÷N.

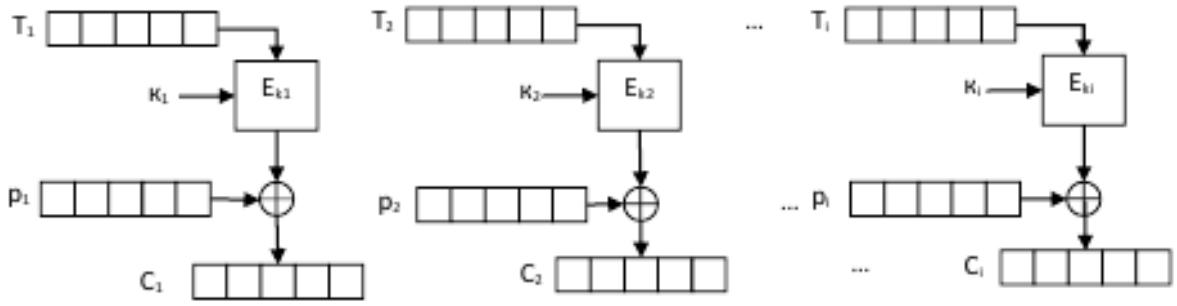


Рис.4.10. Схема шифрування в режимі CTR

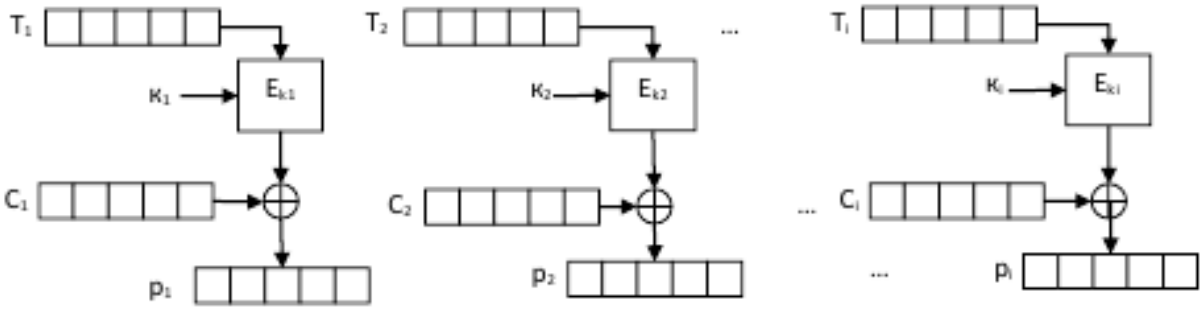


Рис.4.11. Схема розшифрування в режимі CTR

Переваги режиму CTR:

- На відміну від режиму OFB, тепер допускається паралельне виконання операцій шифрування (розшифрування) блоків.
- Для шифрування/розшифрування використовується одна функція. • Операції шифрування/розшифрування є симетричними.
- Можливість розпаралелення шифрування/розшифрування блоків.

Недоліки:

- Некритичність по відношенню до ставки і втрати блоків.
- Використання псевдо генератора випадкових чисел.

Узагальнені відомості про стандартні режими шифрування представленні в Таблиці 4.1.

Таблиця 4.1. **Узагальнені відомості про стандартні режими шифрування**

Режим	Перетворення	C_0	Переваги	Недоліки
ECB	$C_i=E_{ki}(p_i)$ $p_i=D_{ki}(C_i)$	-	Можливість розпаралелення шифрування	Некритичність до вставки і втрати блоків
CBC	$C_i= E_{ki}(p_i \text{ XOR } C_{i-1})$ $p_i= C_{i-1}\text{XOR } D_{ki}(C_i)$	+	Критичність до вставки і втрати блоків Автентифікація з використанням MAC	Неможливість розпаралелення шифрування
CFB	$C_i= E_{ki}(C_{i-1}) \text{ XOR } p_i$ $p_i = E_{ki}(C_{i-1}) \text{ XOR } C_i$	+	Критичність до вставки і втрати блоків Використовується одна функція шифрування Симетричність функцій шифрування/розшифрування	Неможливість розпаралелення шифрування
OFB	$C_i= p_i \text{ XOR } O_i$ $p_i = C_i \text{ XOR } O_i$	+	Критичність до вставки і втрати блоків Використовується одна функція шифрування Симетричність функцій шифрування/розшифрування	Неможливість розпаралелення шифрування
CTR	$C_i= p_i \text{ XOR } O_i$ $p_i = C_i \text{ XOR } O_i$	-	Можливість розпаралелення шифрування Використовується одна функція шифрування Симетричність функцій шифрування/розшифрування	Некритичність до вставки і втрати блоків

4.7. SP-мережа

На початку 70-их років XIX ст. Х.Фейстелем (IBM) було запропоновано два підходи до побудови блокових шифрів: SP-мережі і мережі Фейстеля.

Ідея, що лежить в основі SP-мереж, полягає в побудові крипостійкої системи за допомогою багаторазового застосування відносно простих криптографічних перетворень, в якості яких К.Шеннон запропонував використовувати перетворення **підстановки** (substitution) і **перестановки** (permutation).

У найпростішому варіанті SP-мережа (*Substitution-Permutation network*, підстановочноперестановочна мережа) являє собою «сендвіч» з шарів двох типів, які використовуються багаторазово по черзі (Рис.4.12). Перший тип шару - P-шар, що складається з P-блоку великої розрядності, за ним йде другий тип шару - S-шар, що представляє собою велику кількість S-блоків малої розрядності, потім знову P-шар і т.д.

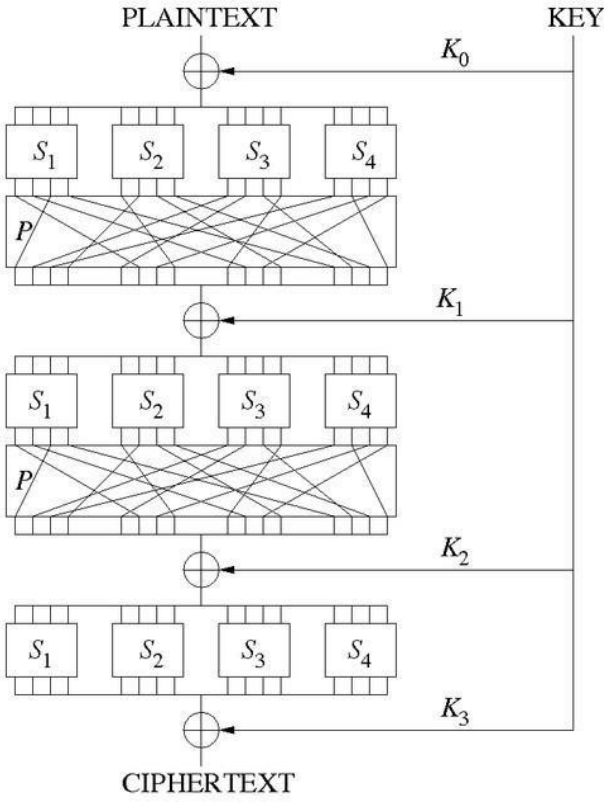


Рис.4.12. Схема SP-мережі

S-блок заміщає невеликий блок вхідних біт на інший блок вихідних біт. Ця заміна повинна бути взаємно однозначною, щоб гарантувати оборотність. Призначення S-блоку полягає в нелінійному перетворенні, що перешкоджає проведенню лінійного криптоаналізу.

P-блок (англ. Permutation box or P-box) - отримує на вхід виводи S-блоків, змінює місцями все біти і подає результат S-блокам наступного раунду. Важливою якістю P-блоку є можливість розподілити виводи одного S-блоку між входами якомога більшої кількості S-блоків.

Багаторазове використання цих перетворень дозволяє забезпечити дві властивості, які повинні бути притаманні стійким шифрами: дифузія (diffusion) і конфузія (confusion). Лінійна стадія перестановки розподіляє надмірність по всій структурі даних, породжуючи дифузію. Нелінійна стадія підстановки перемішує біти ключа з бітами відкритого тексту, створюючи конфузію.

Дифузія передбачає поширення впливу одного знаку відкритого тексту на значну кількість знаків шифротексту. Наявність у шифра цієї властивості дозволяє:

- приховати статистичну залежність між знаками відкритого тексту, інакше кажучи, перерозподілити надмірність вихідного мови за допомогою розповсюдження її на весь текст;
- не дозволяє відновлювати невідомий ключ по частинах.

Мета **конфузії** - зробити якомога більш складною залежність між ключем і шифротекстом.

Криптоаналітик на основі статистичного аналізу перемішаного тексту не повинен отримати будь-якої значної інформації про використаний ключ.

Застосування дифузії і конфузії порізно не забезпечує необхідну стійкість, надійна криптосистема виходить тільки в результаті їх спільного використання.

Лавинний ефект (avalanche) - це число символів, яке змінилося в зашифрованому тексті при зміні одного біта відкритого тексту або ключа. Чим більше лавинний ефект, тим вище надійність шифру.

Першим криптографічним алгоритмом на основі SP-мережі був «Люцифер» (1971). В даний час з алгоритмів на основі SP-мереж широко використовується AES.

Альтернативою SP-мереж є мережі Фейстеля.

4.8. Мережі Фейстеля

Мережею Фейстеля 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 розшифрування використовується один і той самий алгоритм - відмінність полягає тільки в порядку використання матеріалу ключа.

В схемі Фейстеля кожний блок розбивається на ліву (l_0) і праву (r_0) частини, над якими здійснюються S раундів шифрування. Після завершення S раундів шифрування ліва (L_s) і права (R_s) частини міняються місцями:

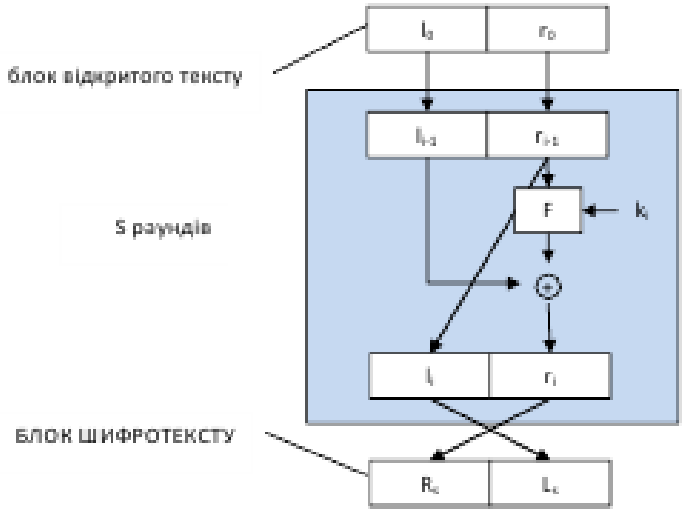


Рис.4.13. Схема мережі Фейстеля

Кожний раунд шифрування здійснюється за правилом:

$$l_i = r_{i-1},\ r_i = l_{i-1} + F(k_i, r_{i-1}).$$

Процес розшифрування шифру Фейстеля принципово не відрізняється від процесу шифрування. Застосовується той самий алгоритм, але на вхід подається шифрований текст, а підключі використовуються в зворотній послідовності: для першого раунду береться підключ S—го раунду шифрування, для другого – (S-1) -ий, І так далі до тих пір, поки не буде введений ключ для останнього раунду. Ця властивість даної схеми шифрування вилась дуже зручною, тому що для розшифрування не потрібно вводити інший алгоритм, відмінний від алгоритму шифрування.

Мережа Фейстеля надійно зарекомендувала себе як крипостійка схема проведення криптоперетворень, і її можна знайти практично в будь-якому сучасному блоковому шифрі.

Цікава особливість шифру Фейстеля полягає в тому, що функція раунду є оберненою незалежно від властивості функції F.

Для створення криптостійкого шрифту залишилося визначити:

- Спосіб генерування підключів k_i.

- Кількість раундів S.
- Визначити функцію F.

Відповіді на ці питання були дані в ході роботи над шифром DES.

Запитання для самоконтролю

1. Які переваги мають блокові шифри?
2. Яке призначення режимів шифрування?
3. Скільки існує стандартних режимів шифрування?
4. Які переваги і недоліки має кожний з стандартних режимів шифрування?
5. Який режим шифрування забезпечує автентифікацію відправника?
6. Що таке синхропосилка і яке її призначення?
7. Які режими шифрування вимагають використання синхропосилок?
8. Які режими шифрування можуть використовуватись при розпаралеленні процесів шифрування.розшифрування?

Результати навчання

Знання про можливості забезпечення високого рівня криптостійкості блокових шифрів, значення режимів шифрування. Розуміння принципів реалізації стандартних режимів. Уміння використовувати переваги та недоліки режимів шифрування. Розуміти основні підходи до побудови багатораундових функцій шифрування.

Розділ 5. Криптосистема DES

5.1. Загальна характеристика

Алгоритм **DES (Data Encryption Standard)** був розроблений у 1977 році і рекомендований

Національним бюро стандартів США в якості основного засобу криптографічного захисту інформації як в державних, так і в комерційних структурах [2] . Він став першим доступним всім бажаючим офіційним алгоритмом і першим світовим стандартом, що проіснував більше 20. Тому його слід відмітити як найважливішу віху на шляху криптографії від чисто військового використання до широкомасштабного застосування.

DES – алгоритм блокового шифрування з довжиною блоку 64 біти і симетричним ключем довжиною 56 біт. На практиці ключ має довжину 64 біти,з яких кожний восьмий використовується для контролю парності байту.

Головні риси шифру DES визначаються тим, що він ґрунтується на схемі Фейстеля з такими параметрами:

- довжина блоку – 64 біти,
- кількість раундів – 16,
- розмір ключа – 56 бітів,
- розмір кожного з підключів k_1, k_2, …, k_{16} – 48 бітів.

[2] Вітчизняним аналогом DES є алгоритм блокового шифрування, специфікований в ГОСТ 28147-89.

5.2. Алгоритм шифрування

Структура алгоритму DES показана на рис.5.1.

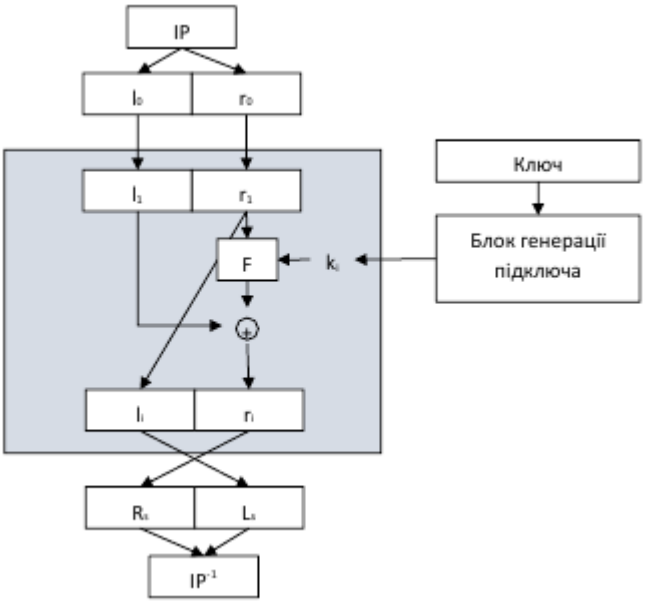


Рис.5.1. Структура алгоритму DES

Алгоритм DES описується за допомогою трьох етапів:

1. До вхідного блоку довжиною 64 біти застосовується фіксована початкова перестановка IP, яка описується виразом (l_0,r_0) ← IP (Вхідний блок).

Тут l_0 та r_0 називаються лівою і правою половинами блоку, кожна з яких має довжину 32 біти.

Перестановка IP застосовується для того, щоб здійснити початкове розсіювання статистичної структури повідомлення. Вона є фіксованою і відкритою (Рис.5.2).

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

Рис.5.2. Матриця початкової перестановки IP

2. Виконуються 16 раундів з таких операцій:
    - Права половина блоку перетворюється функцією F з використанням поточної ключової послідовності довжиною 48 біт, яка знімається з виходу блоку вироблення ключової послідовності.
    - Результат перетворення правої половини складається по модулю 2 з лівою частиною, а сума записується у вихідний регістр, при цьому вихідна права половина за допомогою операції зсуву записується на місце вихідної лівої половини.

Їх можна описати рівняннями:

$$l_i = r_i \ , \ r_i = l_i \ \ F(r_{i-1}, k_i) \ ,$$

де k_i – ключ раунду, який складається з 48-бітового підрядка 56-бітного вихідного ключа, F – функція шифрування, яка представляє собою прерстановочний шифр. Ці операції перестановки забезпечують значний рівень «дифузії повідомлення».

3. До результату 16-го раунду (L16, R16) застосовується завершуюча перестановка, яка є оберненою до початкової перестановки(Рис.5.3)

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

Рис.5.3. Матриця завершуючої перестановки IP

5.3. Структура функції F

Структура функції Fзображена на Рис.5.4.

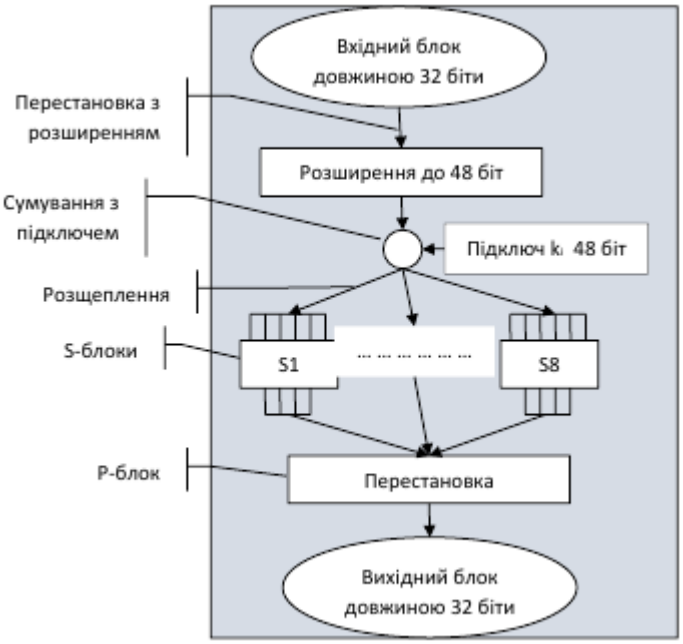


Рис.5.4. Структура функції F

В кожному раунді перетворення F складається з п’яти кроків:

1. **Перестановка з розширенням**. Права половина з 32 бітів розширюється до 48 бітів і перемішується. Ця операція передбачає дописування у вихідну послідовність окремих бітів у відповідності з підстановкою розширення (Рис.5.5).

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

Рис.5.5. Матриця підстановки розширення

Підстановка розширення забезпечує, що один вхідний біт впливає на дві заміни через S-блоки, що створює «лавиноподібний» ефект – мала відмінність між двома наборами вхідних даних перетворюється у велику на виході.

2. **Сумування з підключем.** До отриманого після перестановки з розширенням рядка з 48 бітів і підключа довжиною також 48 бітів

застосовується операція виключаю чого АБО, тобто кожна пара відповідних бітів складається по модулю 2.

48-бітний підключ отримується з 56-бітного ключа у такий спосіб. Біти ключа записуються у два 28-бітні циклічних регістрів зсуву, які переміщують вміст в кожному такті на кількість бітів, що залежить від номера раунду (Рис.5.6).

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

Рис.5.6. Значення циклічного зсуву в 16-ти раундах шифрування

Результуюча послідовність отримується шляхом вибірки 48 бітів з вмісту регістрів (Рис.5.7).

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

Рис.5.7. Матриця вибірки з вмісту регістрів

3. **Розщеплення.** Результат сумування з підключем розщеплюється на 6 частин по 8 бітів в кожному, кожна з яких передається в один з восьми S-блоків. 4) **S-блок** . Перетворює набір з 6 бітів в набір з 4 бітів.

S-блоки є нелінійними компонентами алгоритму DES, які і забезпечують криптостійкість шрифту. Кожний S-блок представляє собою пошукову таблицю з чотирьох рядків і шістнадцяти стовпців (Таблиця 5.1). Шість бітів, що входять до блоку, визначають який рядок і стовпець необхідно використовувати для заміни. Перший і шостий біт задають номер рядка, а інші – номер стовпця. Вихід S-блоку – бітове представлення числа відповідної комірки таблиці.

Таблиця 5.1. **Підстановки в S-блоках**

S1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	6	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	12
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

4) **Р-блок**. Вихід S-блоків з восьми 4-бітових елементів надходить до Р-блоку, в якому відбувається перестановка (Рис.5.8).

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

Рис.5.8. Матриця Р-перестановки

Розшифровування в алгоритмі DES відбувається аналогічно шифруванню з тією різницею, що вибірка ключової послідовності в раундах розшифровування буде оберненою, тобто k_{16}, k_{15} … k_1.

5.4. Стійкість DES

Основними недоліками DES, що суттєво зменшують рівень його безпеки, є такі:

- **Наявність слабких ключів.** Їх виникнення пов'язане з тим, що в кожному раунді при генерації підключа використовуються два регістри зсуву. При цьому може виникати ситуація, коли для кожного підключа буде генеруватись одна, або лише дві, чотири ключових послідовності. Такі ключі називаються слабкими. Приклд слабкого ключа: 1F1F1F1F 0E0E0E0E (з урахуванням бітів парності); однакові підключі будуть генеруватись у всіх 16 раундах.
- **Невелика довжина ключа**. На сучасному рівні розвитку мікропроцесорних засобів довжини ключа 56 біт (або 64 біти з котролем парності) не забезпечує достатнього рівня захисту.
- **Надлишковість ключа**. Наявність контролю парності кожного байту ключа, що і приводить до надлишковості, дозволяє відновлювати ключ при виникненні помилок в памяті. **Використання статичних підстановок в S-блоках**.

Слабкість алгоритму DES, що пов'язана з невеликою довжиною ключа, стала очевидною в 1990-их роках. Основні відомі атаки на DES:

1. **Диференційний криптоаналіз.** Метод запропонований в 1991 р. ізраїльськими вченими Елі Бахамом та Еді Шаміром. Ключ

шифрування обчислюється за умови наявності у атакуючого можливості генерації 247 спеціально вибраних пар «відкритий-зашифрований» тексти.

2. **Лінійний криптоаналіз.** Метод запропонований в 1993 р. японцем Міцуру Мацуі. Доведена можливість обчислення ключа DES за умови наявності у атакуючого 247 пар «відкритийзашифрований» тексти.
3. **Метод Девіса.** Метод в 1994 р. запропоновали Елі Біхам і Алекс Бірюков. Він дозволяє обчислити 6 бітів ключа при наявності 250 пар або 24 біти при наявності 252 пар «відкритийзашифрований» тексти.

Таким чином всі відомі атаки на DES вимагають великої кількості пар «відкритий-зашифрований» тексти, отримання яких у свою чергу є працезатратним процесом. Тому найбільш простим способом взлому DES вважається перебір всіх варіантів ключа шифрування. В 1998 році була створена пошукова машина під назвою DES Cracker вартістю 250 000 дол., яка знаходила ключі за 56 годин.

5.5. Похідні від DES шифри

3DES

З метою ефективного збільшення довжини ключа, в 1978 році був запропонований алгоритм **3DES** або **потрійний DES** (**TripleDES**).

Існує 3 типи алгоритму 3DES:

- DES-EEE3 (**шифрування-шифрування-шифрування**): шифрування виконується три рази з трьома різними ключами.
- DES-EDE3 (**шифрування-розшифрування-шифрування**): виконується операції шифровкарозшифровка-шифровка з трьома різними ключами.
- DES-EEE2 (**шифрування-шифрування**): як DES-EEE3, в якому на першому і третьому кроці використовується однаковий ключ
- DES-EDE2 (**шифрування-розшифрування**): як DES-EDE3, в якому на першому і третьому кроці використовується однаковий ключ.

Найбільш популярний різновид 3DES - це DES-EDE3 (Рис.5.9), для нього алгоритм виглядає так: $C = E_{K3}(E_{K}{}^{-}{}_{2}{}^{1}(E_{K1}(p))), p = E_{K}{}^{-}{}_{1}{}^{1}(E_{K2}(E_{K}{}^{-}{}_{3}{}^{1}(C)))$.

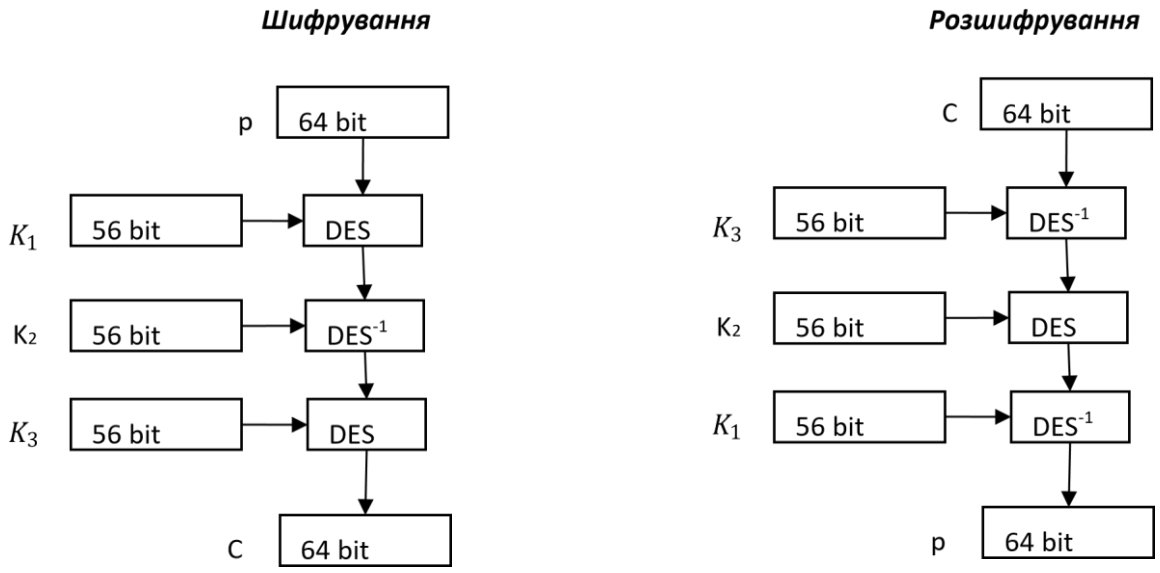


Рис.5.9. Схема шифрування/розшифрування для 3DES

При цьому, як неважко порахувати, довжина ключа стає рівною 168 бітам.

Логічним виглядає запитання при використання «подвійного» 2DES, в якому криптотекст C отримується з вхідного тексту P в результаті «двокрокового» перетворення $C= E_{K2}(E_{K1}(P))$, де K_1 і K_2 - ключі шифрування. Такий алгоритм можливий, але він вразливий до атаки «зустріч посередені». За допомогою цієї атаки криптоаналітик може отримати обидва ключі K_1 і K_2 для 2DES за наявності лише двох пар «відкритий-зашифрований» тексти P_1-C_1, P_2-C_2 наступним способом:

1. Виконуємо шифрування $E_{KX}(P_1)$ для всіх X ключів.
2. Виконуємо розшифрування $D_{KY}(C_1)$ для всіх Y ключів.
3. Порівнюємо результати виконання п.1 і п.2. Якщо вони співпали, то $K_X=K_1$ і $K_Y=K_2$.

Але число таких співпадінь може бути більшим і оцінюється як ~248. Тому для виключення «хибних» ключів необхідно повторити п.1-п.3 з парою P_2-C_2.

Ймовірність наявності більш ніж одного співпадіння ~2^{-16}. Тобто така атака вимагає приблизно в 2 рази більше обчислень, ніж при переборі ключів DES.

DESX

В 1996 р. запропонований алгоритм **DESX**. Суть алгоритму полягає в тому, що перед виконанням одноразового DES і після нього на дані операцією XOR накладаються різні 64-бітні фрагменти ключа: $C = K_3 \oplus DES_{K1}(K_2 \oplus p)$, $p = K_2 \oplus DES_{K^{-1}1}{}^{1}(K_3 \oplus c)$

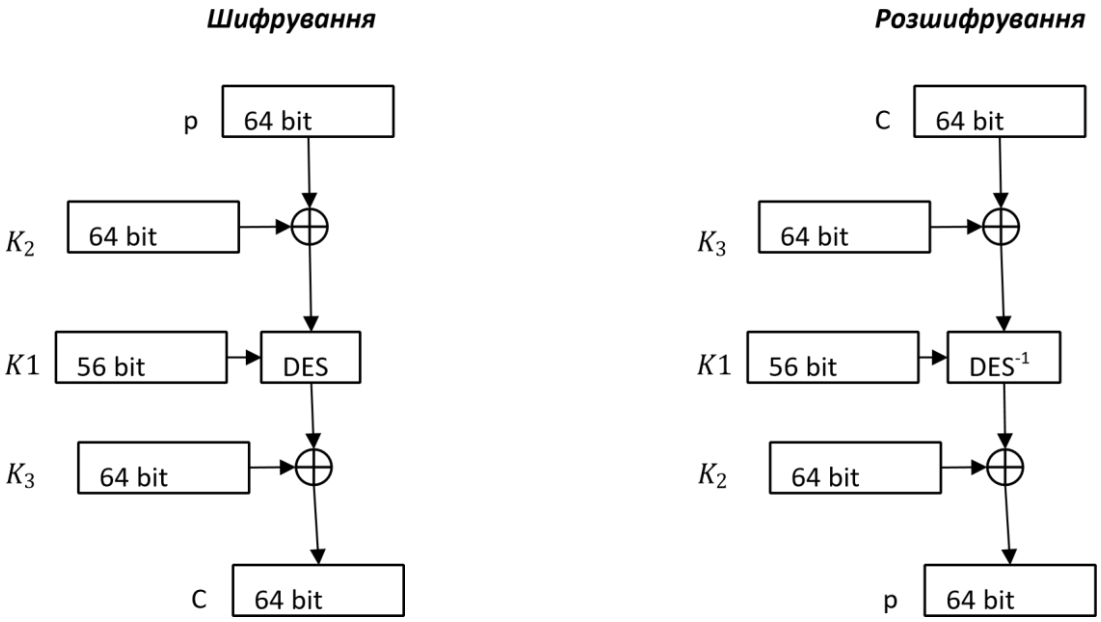


Рис.5.10. Схема шифрування/розшифрування для DESX

Операція накладання фрагмента ключа на вхід і / або вихід алгоритму шифрування називається **відбілюванням**, сенс якого полягає в тому, щоб перешкодити криптоаналітику отримати для дослідження алгоритму пари «відкритий текст – шифротекст».

DESX повністю сумісний з алгоритмом DES в разі, якщо $K_2 = K_3 = 0$.

Оскільки K_2 та K_3 мають розмір по 64 біта, а K1 - це звичайний 56-бітний ключ DES, то повний розмір ключа DESX становить 184 біта.

Запропоновано ряд різновидів алгоритму DESX:

- З використанням змінного розміру ключа шифрування, попередньо застосувавши до нього хешування за алгоритмом SHA-1.

- З використанням 120-бітного ключа шифрування (при цьому K_3 встановлюється рівним K_2). Замість обох операцій XOR виконується складання по модулю 264.

Загалом DESX вважається кращим, ніж DES. Проте в алгоритмі DESX було знайдено декілька уразливостей, що перешкодило широкому поширенню цього алгоритму.

5.6. DES і шифрована файлова система EFS

Шифрована файлова система (Encrypting File System, EFS) - це базова технологія, що реалізує шифрування на рівні файлів в операційних системах Microsoft Windows NT (починаючи з Windows 2000 і вище, за винятком «домашніх» версій - Windows XP Home Edition, Windows Vista Basic і Windows Vista Home Premium і т.п.).

Система надає можливість «прозорого шифрування» даних, що зберігаються на розділах з файловою системою NTFS, для захисту конфіденційних даних від несанкціонованого доступу при наявності фізичного доступу до комп'ютера.

В Windows 2000 EFS використовувала один алгоритм шифрування - це DESX, який є спеціальною модифікацією широко поширеного стандарту DES. В інших версіях Windows[3] EFS використовує різні симетричні алгоритми шифрування (Таблиця 5.2).

EFS використовує симетричне шифрування для захисту файлів, а також шифрування, засноване на парі відкритий / закритий ключ для захисту випадково згенерованого ключа шифрування для кожного файлу. За замовчуванням закритий ключ користувача захищений за допомогою шифрування користувальницьким паролем, і захищеність даних залежить від стійкості пароля користувача.

---

[3] Починаючи з Windows XP доступна теоретична можливість використання сторонніх бібліотек для шифрування даних.

Таблиця 5.2. **Алгоритми симетричного щифрування в ОС Windows**

Операційна система	**Алгоритм шифрування за замовчуванням**	**Альтернативні доступні алгоритми**
Windows 2000	DESX	(none)
Windows XP RTM	DESX	3DES
Windows XP SP1	AES	3DES, DESX
Windows Server 2003	AES	3DES, DESX
Windows Vista	AES	3DES, DESX
Windows Server 2008	AES	3DES, DESX (?)

Процедура шифрування здійснюється драйвером EFS і полягає у такому (Рис.5.11):

1. Для кожного файлу (каталогу), що шифрується, випадковим чином генерується ключ симетричного шифрування - так званий FEK (File Encryption Key).
2. Файл (каталог) шифрується за допомогою алгоритму симетричного шифрування з використанням в якості ключа FEK.
3. FEK захищається шляхом асиметричного шифрування за алгоритм RSA відкритим ключем користувача.
4. Зашифрований FEK зберігається в заголовку зашифрованого файлу (каталогу).

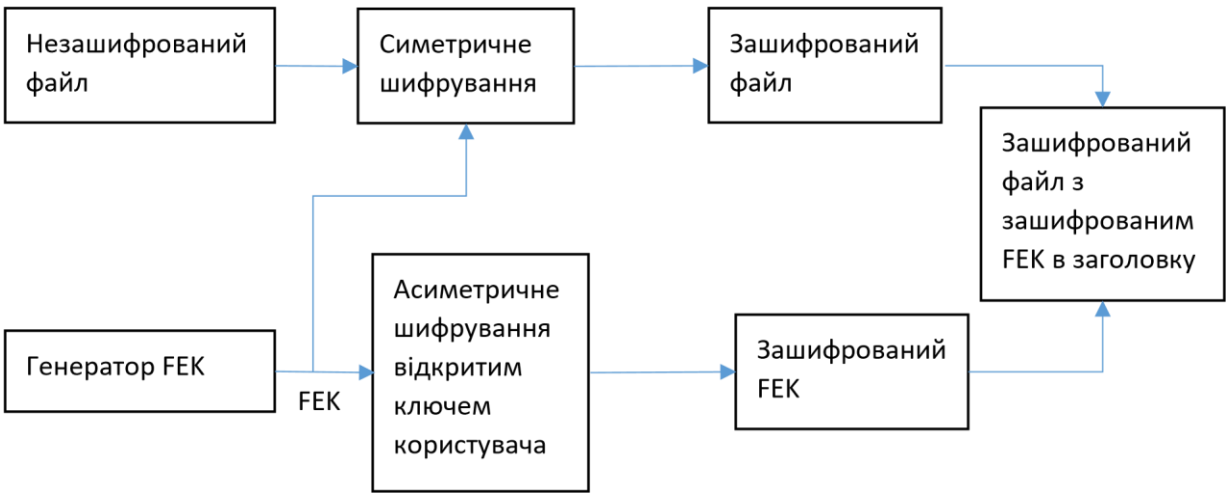


Рис.5.11. Схема шифрування при використанні EFS

Процедура розшифрування також здійснюється драйвером EFS полягає у такому (Рис.5.12):

1. FEK зчитується з заголовка зашифрованого файлу зашифрований і розшифровується закритим ключем користувача.
2. За допомогою розшифрованого FEK розшифровується необхідний файл (каталог).

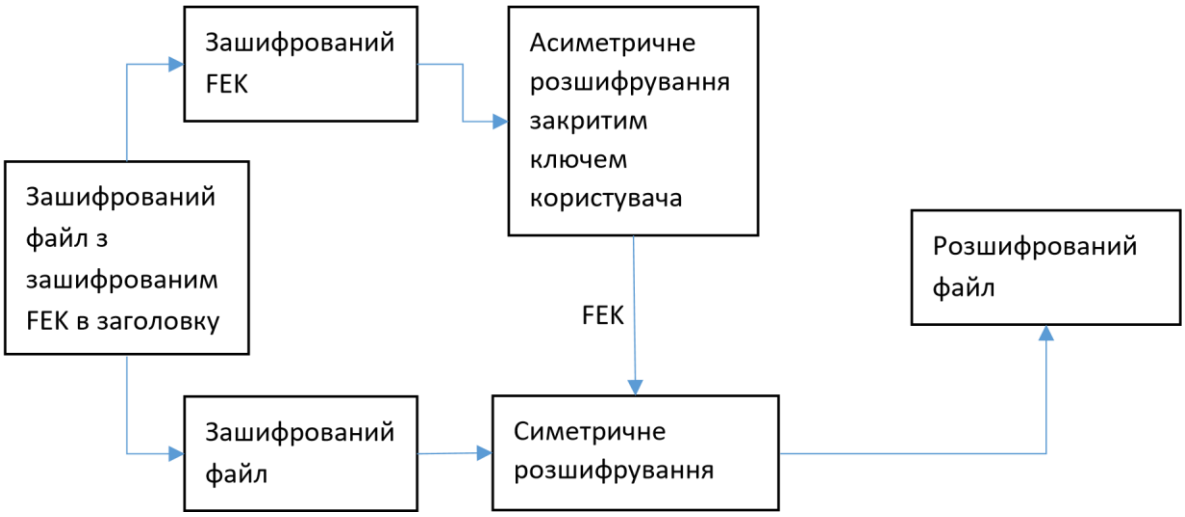


Рис.5.12. Схема розшифрування при використанні EFS

Оскільки шифрування / розшифрування файлів відбувається за допомогою драйвера файлової системи (по суті надбудови над NTFS), воно відбувається прозоро для користувача і додатків.

Варто зауважити, що EFS не шифрує файли, передані по мережі, тому для захисту переданих даних необхідно використовувати інші протоколи захисту даних (IPSec або WebDAV).

За замовчуванням EFS зконфігурована таким чином, що користувач може відразу почати використовувати шифрування файлів. Операції шифрування і розшифрування підтримуються для файлів і каталогів. У тому випадку, якщо шифрується каталог, автоматично шифруються всі файли і підкаталоги цього каталогу. Якщо зашифрований файл переміщується або перейменовується з

зашифрованого каталогу в незашифрований, то він все одно залишається зашифрованим.

Зашифровані файли зберігаються на диску в зашифрованому вигляді. При читанні файлу дані автоматично розшифровуються, а при записі - автоматично шифруються.

Операції шифрування /розшифрування можна виконати двома різними способами - використовуючи Windows Explorer або через інтерфейс командного рядка.

Для того щоб зашифрувати каталог з **Windows Explorer**, користувачеві потрібно просто вибрати один або декілька каталогів і встановити прапорець шифрування у вікні розширених властивостей каталогу (Рис.5.13).

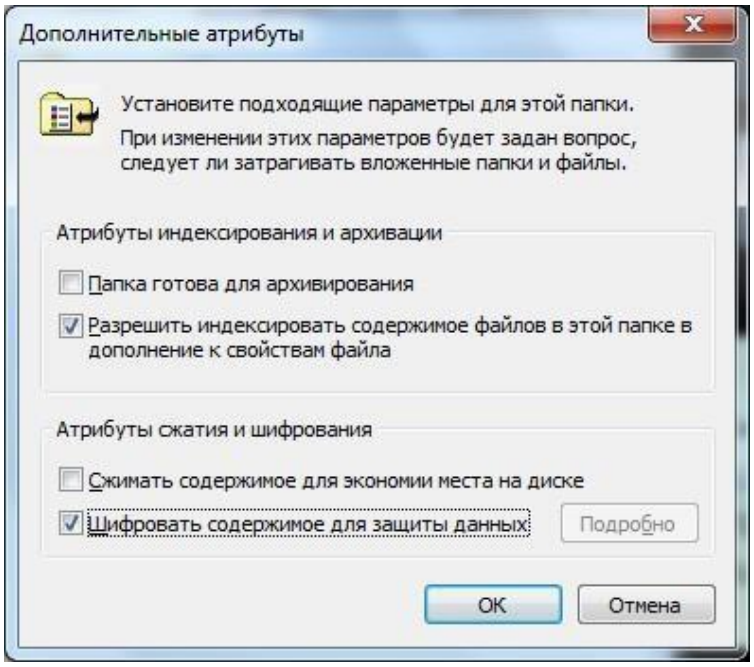


Рис.5.13. Діалог для шифрування за допомогою EFS

Всі створювані пізніше файли і підкаталоги в цьому каталозі будуть також зашифровані. Таким чином, зашифрувати файл можна, просто скопіювавши (або перенісши) його в "зашифрований" каталог.

Для роботи з EFS через інтерфейс командного рядка призначена утиліта **cipher**. При виконанні даної команди без параметрів буде виведено вміст поточної папки з міткою U перед файлом якщо він не зашифрований, і E якщо зашифрований.

Команда шифрування має вигляд: **cipher / E** <шлях до папки>.

Команда розшифрування має вигляд: **cipher / D** <шлях до папки>.

Дана утиліта має ряд інших можливостей, серед яких перешифруваннння файлів з новим ключем, генерація нового ключа шифрування, додавання агента відновлення і т. д.

Важливо пам'ятати, що зашифровані файли не можуть бути стиснуті засобами Windows, і навпаки, якщо каталог стиснутий, його вміст не може бути зашифровано.

Запитання для самоконтролю

1. Які основні характеристики шифру DES?
2. Які дії реалізуються в алгоритмі DES?
3. Як побудована функція шифрування DES?
4. Яка роль S-блоків в функції шифрування DES?
5. Які основні недоліки DES?
6. Як побудований шифр 3DES? 7. Як побудований шифр DESX?
8. Яке призначення шифрованої файлової системи у ОС Windows?
9. Як здійснюється шифрування.розшифрування з використанням з використанням шифрованої файлової системи?

Результати навчання

Знання про принципи побудови і особливості реалізації першого відкритого симетричного шифру DES, результати аналізу рівня криптостійкості. Розуміння напрямків усунення основних слабкостей шифру. Уміння застосовувати можливості шифрованої файлової системи для забезпечення безпеки операційних систем Windows.

Розділ 6. Сучасні симетричні криптосистеми

6.1. ДСТУ ГОСТ 28147:2009

В 1989 р. в СРСР був прийнятий державний стандарт шифрування даних - ГОСТ 28147-89. З моменту прийняття стандарт мав гриф «Для службового користування» до кінця існування самого СРСР. Тільки в 1994 році він був формально оголошений повністю відкритим.

В 2009 році стандарт прийнятий (за методом підтвердження) як стандарт України з наданням національного позначення ДСТУ ГОСТ 28147:2009.

Шифр ДСТУ ГОСТ 28147:2009, як і DES, є блоковим. Дані шифруються за схемою Фейстеля 64бітними блоками з використанням 256-бітного ключа шифрування. При цьому виконується 32 раунди перетворень.

У структурі алгоритму розрізняють:

- Основний крок – послідовність дій, що виконується в кожному базовому циклі (з різними значеннями підключів)
- Базові цикли [4] (*«32-З», «32-Р»,* (*«16-З»,*) - відрізняються числом повторень основного кроку і порядком використання елементів ключа.
- Режими роботи – спеціальні методи забезпечення криптостійкості, що використовують результати шифрування попередніх блоків для шифрування наступних.

Основний крок складається з таких операцій (Рис.6.1):

1. Один з 32-бітових субблоків даних сумується з 32-бітним значенням ключа раунду K_i по модулю 2^{32}.
2. Результат попередньої операції розбивається на 8 фрагментів по 4 біта, які паралельно «проганяються» через 8 таблиць замін S1 ... S8.

[4] У позначенні базових циклів число (32 або 16) вказує на число повторень основного кроку, буква (З або Р) вказує на різновид порядку використання елементів ключа.

3. 4-бітові фрагменти (після замін) об'єднуються назад у 32-бітний субблок.
4. Значення отриманого якого субблоку циклічно зсувається вліво на 11 біт.

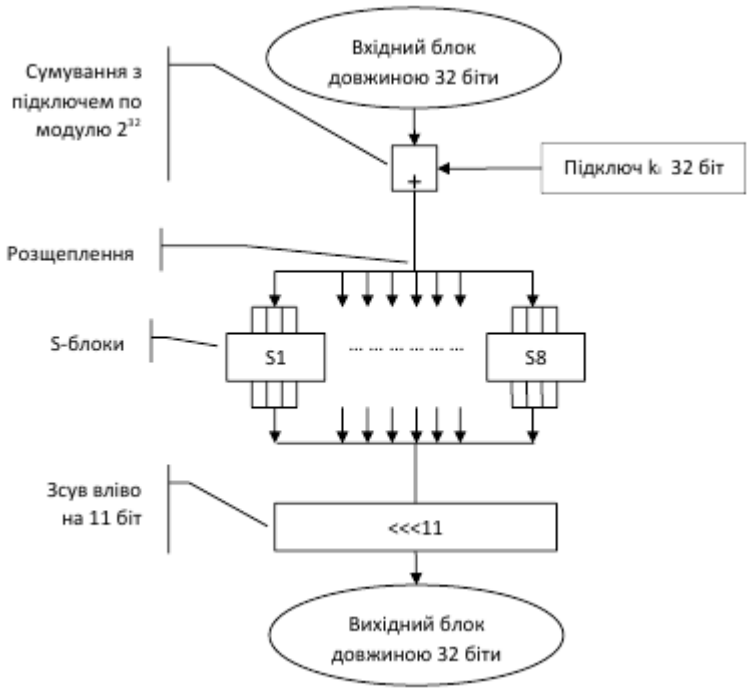


Рис.6.1. Структура функції шифрування для ГОСТ 28147-89

Тоді структуру алгоритму можна описати багатораундовою схемою (Рис.6.2).

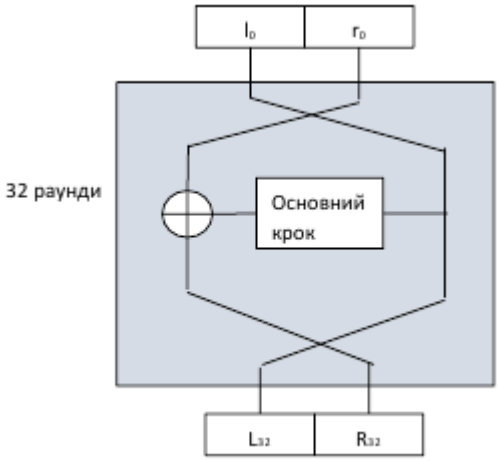


Рис.6.2. Структура алгоритму ГОСТ 28147-89

Генерація ключів проста. 256-бітний ключ розбивається на вісім 32-бітових підключів. Оскільки алгоритм має 32 раунди, кожен підключ використовується в чотирьох раундах за схемою на Рис.6.3.

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

Рис.6.3. Схема використання підключів при шифруванні в ГОСТ 28147-89

Тобто в раундах шифрування послідовно використовуються 32-бітові фрагменти K_1 ... K_8 вихідного 256-бітного ключа шифрування в наступному порядку: K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, - за винятком останніх 8 раундів - в раундах з 25-го по 31-й фрагменти використовуються в зворотному порядку. Така послідовність використання підключів при за шифруванні отримала назву *базовий цикл «32З».*

Вважається, що стійкість алгоритму визначається структурою S-блоків. Входом і виходом S-блоків є 4-бітні числа, тому кожен S-блок може бути представлений у вигляді рядка чисел від 0 до 15, розташованих в деякому порядку. Тоді порядковий номер числа буде вхідним значенням S- блоків, а саме число - вихідним значенням S- блоків.

Довгий час структура S- блоків в відкритій пресі не публікувалася.

Усі вісім S-блоків можуть бути різними. Фактично, вони можуть бути додатковим ключовим матеріалом, але частіше є параметром схеми, загальним для певної групи користувачів. Для цілей тестування рекомендуються наведені такі S-блоки[5], які показані в Таблиці 6.1.

[5] ГОСТ Р 34.11-94 – такі S-блоки використовуються в додатках Центрального Банку РФ і вважаються досить сильними

Таблиця 6.1. **Рекомендовані S-блоки**

Номер S-блоку	Значення															
1	4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
2	14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
3	5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11
4	7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
5	6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
6	4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
7	13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
8	1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

Розшифрування повністю аналогічно шифруванню, але з іншим порядком використання фрагментів ключа: в прямому порядку K_1 ... K_8 - у перших 8 раундах; в інших раундах - у зворотному порядку K_8... K_1 (Рис.6.4).

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

Рис.6.4. Схема використання підключів при розшифруванні в ГОСТ 28147-89

Така послідовність використання підключів при розшифруванні отримала назву *базовий цикл «32-Р».*

Стандарт також передбачає і описує різні режими застосування алгоритму:

- *режим простої заміни* – є аналогом режиму ECB в DES;
- *режим гамування* – є аналогом режиму OFB в DES;
- *режим гамування зі зворотним зв'язком* або *режим гамування із зціпленням блоків* – є аналогом режиму CFB в DES.
- *режим обчислення імітовставки* – є аналогом режиму CBC в DES.

Режим простої заміни приймає на вхід дані, розмір яких кратний 64-м бітам. Результатом шифрування є вхідний текст, перетворений блоками по 64

біта у випадку зашифрування циклом «32-З», а в разі розшифрування - циклом «32-Р».

Режим гамування приймає на вхід дані будь-якого розміру, а також додатковий 64-бітовий параметр *синхропосилку*. В ході роботи синхропосилка перетвориться в циклі «32-З», результат ділиться на дві частини. Перша частина складається по модулю 2^{32} з постійним значенням 101010116. Якщо друга частина дорівнює 2^{32}-1, то її значення не змінюється, інакше вона складається по модулю 2^{32}-1 з постійним значенням 101010416. Отримане об'єднанням обох перетворених частин значення, зване *гамою шифру*, надходить у цикл «32-З», його результат порозрядно складається по модулю 2 з 64-розрядним блоком вхідних даних. Якщо останній менше 64-х розрядів, то зайві розряди отриманого значення відкидаються. Отримане значення подається на вихід. Якщо ще є вхідні дані, то дія повторюється: складений з 32-розрядних частин блок перетвориться по частинах і так далі.

Режим гамування зі зворотним зв'язком також приймає на вхід дані будь-якого розміру і синхропосилку. Блок вхідних даних поразрядно сумується по модулю 2 з результатом перетворення в циклі «32-З» синхропосилки. Отримане значення подається на вихід. Значення синхропосилки замінюється в разі зашифрування вихідним блоком, а в разі розшифрування - вхідним, тобто зашифрованим. Якщо останній блок вхідних даних менше 64 розрядів, то зайві розряди гами (виходу циклу «32-З») відкидаються. Якщо ще є вхідні дані, то дія повторюється: з результату зашифрування заміненого значення утворюється гамма шифру і т.д.

Режим вироблення імітовставки приймає на вхід дані, розмір яких становить не менше двох повних 64-розрядних блоків, а повертає 64-розрядний блок даних, що називається *імітовставкою*. Тимчасове 64-бітове значення встановлюється в 0, далі, поки є вхідні дані, воно поразрядно складається по модулю 2 з результатом виконання циклу «16-З», на вхід якого подається блок вхідних даних. Для *базового циклу «16-З»* двічі

використовуються елементи ключа з першого по останній. Після закінчення вхідних даних тимчасове значення повертається як результат.

У відкритій літературі можна знайти досить мало робіт, присвячених криптоаналізу алгоритму ДСТУ ГОСТ 28147:2009. Це особливо помітно в порівнянні з величезним числом робіт, присвячених криптоаналізу DES і AES або криптоаналізу деяких з широко використовуваних алгоритмів шифрування , наприклад, IDEA або SAFER. За результатами відкритих робіт можна зробити висновок про досить високу криптостійкість вітчизняного стандарту шифрування.

Основними перевагами алгоритм ДСТУ ГОСТ 28147:2009 є:

- ефективність реалізації і відповідно висока швидкодія.
- безперспективність силової атаки;
- наявність захисту від нав'язування помилкових даних (вироблення імітовставки).

6.2. AES (Advanced Encryption Standard)

В 1998 р. Національний інститут стандартів і технологій відмовився сертифікувати DES як стандарт Уряду США. Після декількох років обговорень в 2000 р. він затвердив замість DES новий стандарт блокового симетричного алгоритму шифрування - AES.

Новий стандарт був прийнятий на основі відкритого конкурсу. Переможцем став алгоритм Rijndael, розроблений бельгійськими криптографами Вінсентом Ременом і Іоном Дамен. Назва алгоритму утворена з перших букв прізвищ його авторів, тому в транскрипції з фламандської вона вимовляється приблизно як «Рендал».

AES відноситься до SP-мереж і ґрунтується на новій архітектурі квадрата (square), для якої характерно:

- уявлення блоку, що шифрується, у вигляді двовимірного байтового масиву;
- шифрування за один раунд всього блоку даних;

- виконання криптографічних перетворень, як над окремими байтами масиву, так і над його рядками і стовпцями, що забезпечує дифузію даних одночасно в двох напрямках - по рядках і по стовпцях.

За архітектурою квадрата побудовані також такі щифри, як SQUARE, CRYPTON, SERPENT.

Загальні характеристики AES:

- Для шифрування і розшифрування використовуються 128-бітові блоки даних.
- Дозволяється використовувати три різних довжини ключа - 128,192 або 256 біт.
- Від розміру ключа залежить число раундів шифрування: довжина 128 біт-10 раундів; довжина 192 біта - 12 раундів; довжина 256 біт -14 раундів.
- Всі раунди, крім останнього, ідентичні.

Обмежимось далі розглядом AES-128. В цьому випадку вхідними даними для алгоритму є масив з

16 байт in_0 , in_1 , ..., in_{15} . Перед початком шифрування байти цього масиву розміщують ся послідовно в стовпців и матриці **InputBlock** (зверху вниз). Усередині алгоритму операції виконуються над матрицею станів **State**. Кінцеве значення матриці стану **OutputBlock** є виходом алгоритму і перетворюється в послідовність байтів шифротекста out_0 , out_1 , ..., out_{15} .

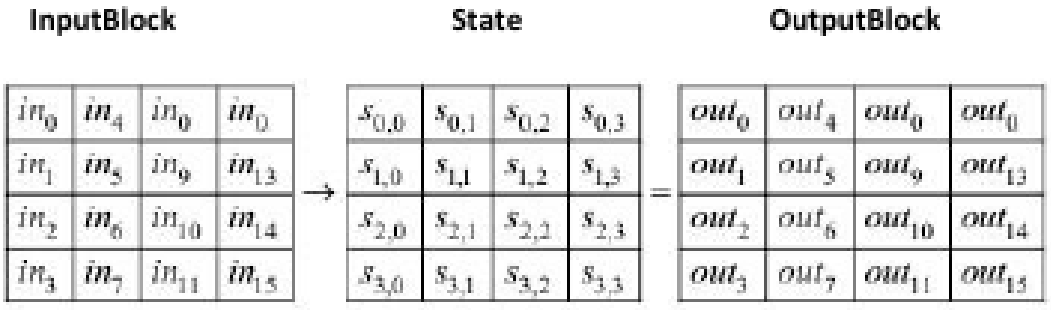


Рис.6.5. Основні матриці перетворень в AES-128

Аналогічно в стовпці матриці **InputKey** потрапляють і 16 байтів k_0, k_1, ..., k_{15} ключа шифру. (Рис.6.5)

k_0	k_4	k_8	k_{12}
k_1	k_5	k_9	k_{13}
k_2	k_6	k_{10}	k_{14}
k_3	k_7	k_{11}	k_{15}

Рис.6.5. Матриця ключа в AES-128

Кожен байти ключа **InputKey** утворює слово , тобто фактично ключ шифру - це чотири слова w_0, w_1, w_2, w3, де $w_0 = k_0\ k_1\ k_2\ k_3$, $w_1 = k_4\ k_5\ k_6\ k_7$ і т.д. З цих слів за допомогою спеціального алгоритму (розширення ключа) утворюється послідовність з 44 слів: w_0, w_1, w_2,…,w_{43} (кожне слово по 32 біта).

На кожен раунд шифрування подаються по чотири слова цієї послідовності. Вони і будуть грати роль раундового ключа . Матриця, яка надходить на вхід кожного раунду, називається матрицею **InputState** , а на виході раунду утворюється матриця **OutputState** . Очевидно, на в ході першого раунду **InputState** = **InputBlock**, а на виході останнього раунду **OutputState** = **OutputBlock**.

Схема перетворення даних показана на Рис.6.6.

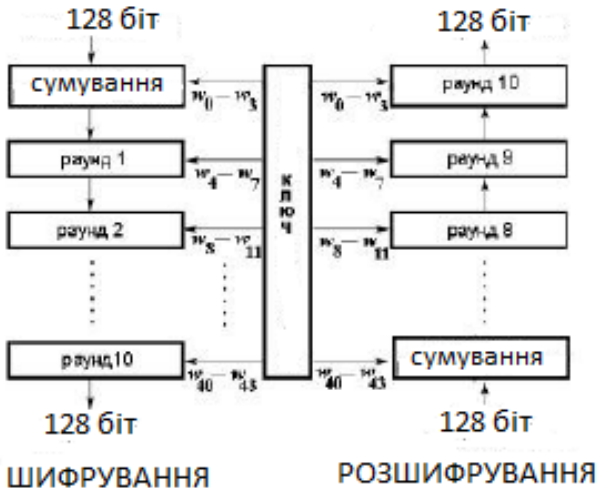


Рис.6.6. Схема перетворення даних в AES-128

Перед першим раундом виконується операція **AddRoundKey** (підсумовування по модулю 2 з початковим ключем шифру). Перетворення, виконані в одному раунді, позначають **Round (State, RoundKey)**, де змінна **State** - матриця, що описує дані на вході раунду і на його виході після шифрування ; змінна **RoundKey** - матриця, що містить раундовий ключ.

Раунд шифрування складається з 4 різних перетворень (Рис.6.7):

1. **SubBytes** - побайтна підстановка а в S-боксі з фіксованою таблицею замін;
2. **ShiftRows** - побайтний зсув рядків матриці **State** на різну кількість байт;
3. **MixColumns** - перемішування байтів в стовпчиках;
4. **AddRoundKey** - складання з раундовим ключем (операція XOR) .

Останній раунд дещо відрізняється від попередніх тим, що НЕ активізує функцію **MixColumns** .

При розшифруванні в кожному раунді виконуються зворотні операції: **InvShiftRows**, **InvSubBytes**, **AddRoundKey** і **InvMixColumns**.

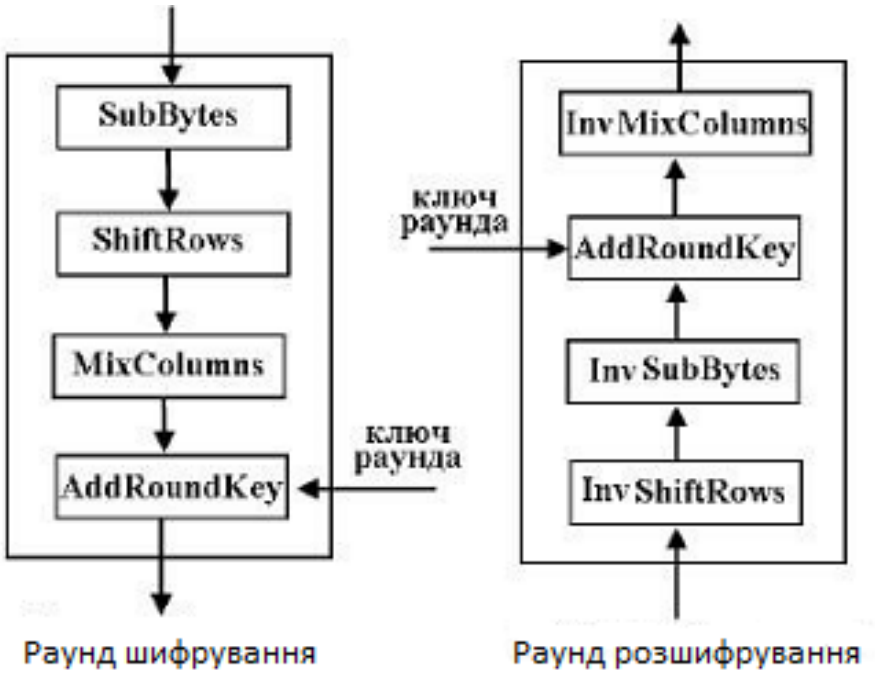


Рис.6.7. Операції в раундах шифрування/розшифрування AES-128

Розглянемо докладніше перетворення раунда шифрування.

1. **SubBytes()**

Процедура SubBytes() обробляє кожен байт стану незалежно (Рис.6.8), проводячи нелінійну заміну байтів використовуючи таблицю замін (S-box).

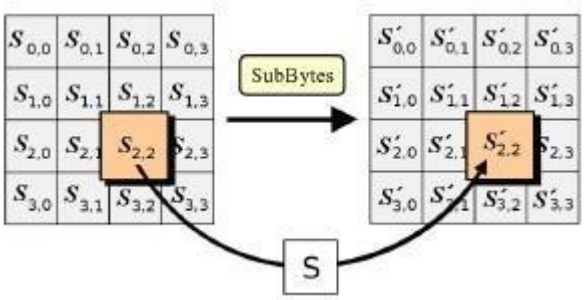


Рис.6.8. Операція SubBytes

Така операція забезпечує нелінійність алгоритму шифрування.

S-box можна зобразити таблицею простої підстановки (Рис.6.9).

S-box

\	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Рис.6.9. S-box

2. **ShiftRows()**

Операція застосовується до рядків матриці State - її перший рядок нерухомий, а елементи нижніх трьох рядків циклічно зсуваються вправо на 1, 2 і 3 байти відповідно (Рис.6.10).

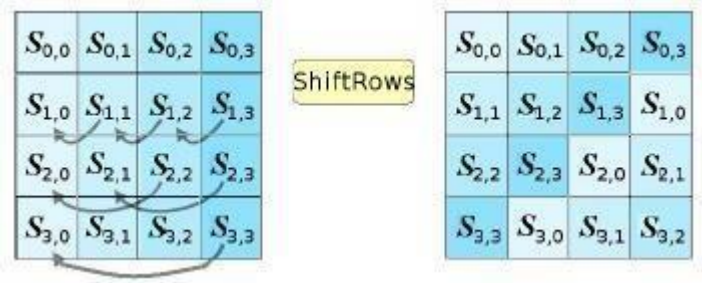


Рис.6.10. Операція ShiftRows

3. MixColumns()

За допомогою цієї операції виконується перемішування байтів в стовпчиках матриці **State** (Рис.6.11). Кожен стовпець цієї матриці приймається за многочлен над полем GF(2^8) і множиться на фіксований многочлен c(x) по модулю многочлена x^4+1.

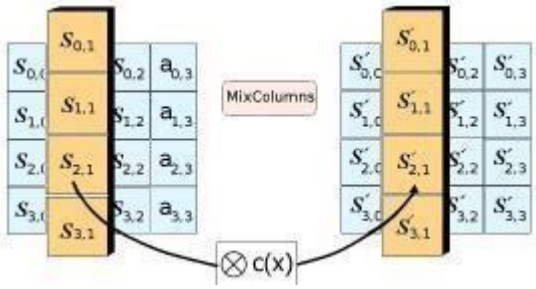


Рис.6.11. Операція MixColumns

4. AddRoundKey()

Функція побітово складає елементи **RoundKeyi** елементи **State (**Рис.6.12).

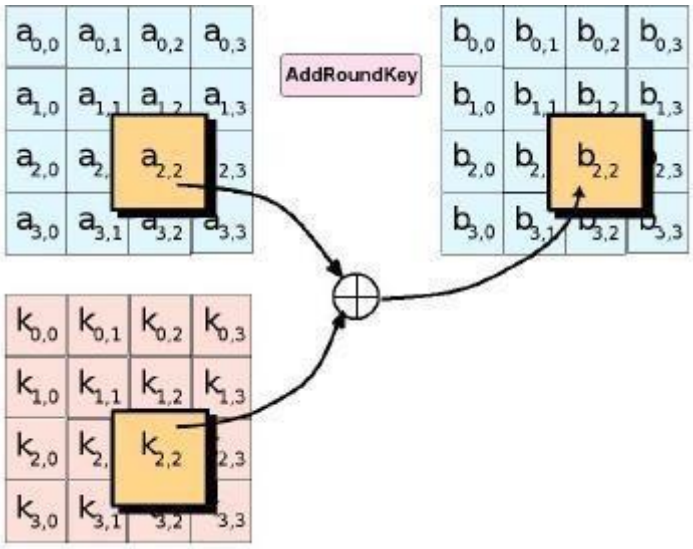


Рис.6.12. Операція AddRoundKey

6.3. Програмна реалізація криптографічних алгоритмів засобами .NET

На платформі .NET передбачена можливість використання ряду криптографічних алгоритмів для симетричного шифрування: DES; TripleDES; Rijndael. Реалізуються вони за допомогою об'єктів двох класів з простору імен System.Security.Cryptography (Таблиця 6.1):

- **CryptographicServiceProvider** class – клас, що надає криптопровайдери для кожного з вказаних алгоритмів.
- **CryptoStream** class – клас для роботи з криптографічним потоком.

Застосування цих основних об'єктів вимагає використання об'єкту **FileStream** з простору імен **System.IO**.

Крім того, для бітового представлення текстових даних необхідні об'єкти класів **UnicodeEncoding** (або **ASCIIEncoding**) з простору імен **System.Text**.

Таблиця 6.1. **Класи .NET для використання з криптопровайдерами**

Клас об'єктів	**Простір імен**
CryptographicServiceProvider CryptoStream	System.Security.Cryptography
FileStream	System.IO
UnicodeEncoding (ASCIIEncoding)	System.Text

Схема шифрування з використанням криптопровайдера наведена на Рис.6.13.

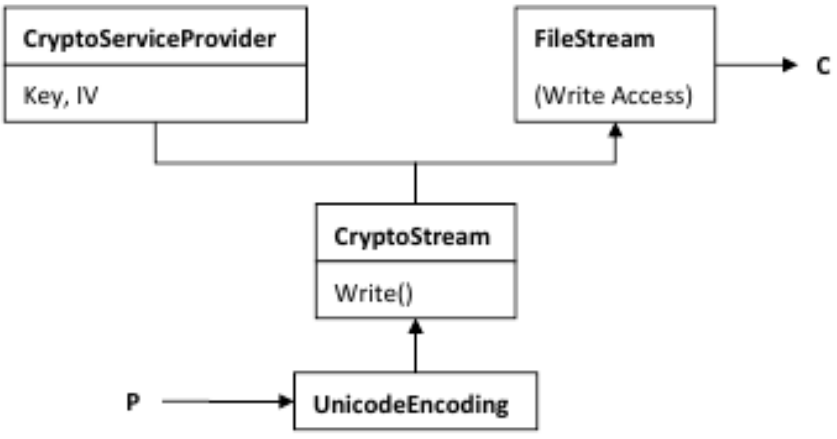


Рис.6.13. Схема шифрування з використанням криптопровайдера

Порядок шифрування за її допомогою такий:

1. Створюється потрібний крипто провайдер і задаються його ключ і вектор ініціалізації :

```
DESCryptoServiceProvider cryptic = new DESCryptoServiceProvider();
//DES cryptic = new DESCryptoServiceProvider(); cryptic.Key
= UnicodeEncoding.UTF8.GetBytes("ABCDEFGH"); cryptic.IV =
UnicodeEncoding.UTF8.GetBytes("ABCDEFGH");
```

Тут для бітового представлення ключа і вектора ініціалізації використаний клас

UnicodeEncoding з простору імен System.Text. Довжина ключа і вектора ініціалізації має точно відповідати вимогам алгоритму. Зокрема для DES вони мають бути 64-бітними, тобто складатися з 8 символів ASCII-кодів.

2. Відкривається звичайний файловий потік для запису зашифрованих даних:

```
 FileStream stream = File.OpenWrite(@"d:\test.txt")
/*FileStream stream = new FileStream(@"d:\test.txt",
FileMode.OpenOrCreate,FileAccess.Write)*/
```

3. Відкритий файловий потік трансформується в крипто потік для запису, якому і передаються дані для шифрування (в бітовій формі) і після шифрування обидва потоки закриваються:

```
   CryptoStream crStream = new CryptoStream(fs,
cryptic.CreateEncryptor(),CryptoStreamMode.Write);
```

4. Дані для шифрування перетворюються у бітову послідовність і всі біти (від 0 до data.Length) записуються в крипто потік за допомогою методу Write ():

```
   byte[] data = UnicodeEncoding.UTF8.GetBytes("Hello World!");
crStream.Write(data,0,data.Length);
```

5. Використані файловий і крипто – потоки закриваються:

```
 crStream.Close(); fs.Close();
```

Схема шифрування з використанням криптопровайдера наведена на Рис.6.14.

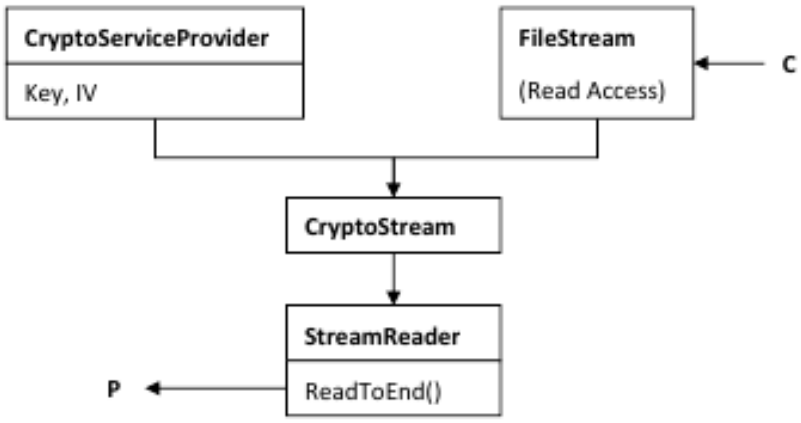


Рис.6.14. Схема розшифрування з використанням криптопровайдера

Порядок розшифрування полягає у такому:

1. Створюється потрібний крипто провайдер і задаються його ключ і вектор ініціалізації :

```
 DESCryptoServiceProvider cryptic = new DESCryptoServiceProvider();
//DES cryptic = new DESCryptoServiceProvider(); cryptic.Key
= UnicodeEncoding.UTF8.GetBytes("ABCDEFGH"); cryptic.IV =
UnicodeEncoding.UTF8.GetBytes("ABCDEFGH");
```

2. Відкривається звичайний файловий потік для читання зашифрованих даних:

```
FileStream fs= File.OpenRead (@"d:\test.txt")
//FileStream stream = new FileStream(@"d:\test.txt", FileMode.Open,FileAccess.Read)
```

3. Відкритий файловий потік трансформується в крипто потік для читання:

```
CryptoStream crStream = new CryptoStream(stream,
cryptic.CreateDecryptor(),CryptoStreamMode.Read).
```

4. Дані з крипто потоку зчитуються за допомогою об’єкта StreamReader і присвоюються текстовій змінній:

```
StreamReader reader = new StreamReader(crStream); string
data = reader.ReadToEnd();
```

5. Значення текстової змінної виводиться на екран і зчитувач та потік закриваються:

```
    Console.WriteLine(data);    reader.Close();
stream.Close();
```

Лістинг програми, що виконує шифрування і розшифрування за алгоритмом DES наведений в додатку 1.

Запитання для самоконтролю

1. Які основні характеристики шифру ДСТУ ГОСТ 28147:2009?
2. Які дії реалізуються в алгоритмі ДСТУ ГОСТ 28147:2009?
3. Як побудована функція шифрування ДСТУ ГОСТ 28147:2009?
4. Яка роль S-блоків в функції шифрування ДСТУ ГОСТ 28147:2009?
5. Як побудований шифр AES?
6. Які сучасні симетричні шифри реалізовані на платформі .Net?
7. Як здійснюється шифрування/розшифрування з використанням класів криптопровайдерів .Net?

Результати навчання

Знання про принципи побудови і особливості реалізації вітчизняного стандарту симетричного шифрування ДСТУ ГОСТ 28147:2009, результати дослідження його криптостійкості. Знання принципів функціонування шифру AES. Уміння використовувати можливості платформи .Net з реалізації криптографічних алгоритмів.

Розділ 7. Модель асиметричної системи

7.1. Передумови виникнення асиметричних систем

Традиційні криптографічні системи мають два суттєвих недоліки:

- **Проблема розподілення ключів в управління ними.** Система з n учасниками вимагає використання n/2 ключів і стільки ж безпечних каналів для їх розповсюдження. При зміні ключа одним з учасників доводиться генерувати і розподіляти (n-1) ключ. А додавання нового учасника вимагає генерування і розподілення n нових ключів.
- **Проблема автентифікації.** Традиційні криптосистеми не забезпечують потребу користувачів у використанні електронного еквіваленту підпису: будь-яке повідомлення, що відправлене одним з них, може бути відправлене і іншим.

Намагання усунути ці недоліки спонукало дослідників до пошуку криптографічних систем нового типу. Ідея такої системи була опублікована в 1976 р. у піонерській роботі У. Діффі і Д.Хеллмана «Нові напрями в криптографії». Криптографічні системи нового типу отримали назву **криптосистем з публічними ключами** або **асиметричних криптосистем**.

Наведемо приклади проблем, які вирішуються за допомогою асиметричних систем.

- **Проблема зберігання паролів на комп'ютері.** Якщо зберігати паролі на магнітному диску, то зловмисник може прочитати їх і використати для несанкціонованого доступу. Тому необхідно організувати зберігання паролів на диску так, щоб унеможливити таке зчитування.
- **Проблема радіолокації в ППО.** При перетині літаком кордону у нього запитується пароль. Зловмисник може перехопити пароль і використати його для нелегального перетину кордону.
- **Проблема віддаленої взаємодії.** Так, при взаємодії банка з клієнтом на початку сеансу банк запитує клієнта ім'я і секретний пароль, який при використанні відкритого каналу зв'язку також може бути перехоплений.

7.2. Модель криптосистеми з публічними ключами

Ідея У. Діффі і Д.Хеллмана полягала у такому. Кожний користувач U криптосистеми створює (або отримує з надійного джерела) пару узгоджених алгоритмів P_u і S_u (Рис.7.1). Алгоритм P_u оголошується публічним, а алгоритм S_u утримується в секреті. Ці алгоритми в залежності від застосування мають задовольняти окремим з наступних умов:

У1. Алгоритми P_u і S_u ефективні, тобто не вимагають занадто великого часу обчислень і великих обсягів пам'яті.

У2. $S_u(P_u(m))=m$ для будь-якого користувача U і будь-якого повідомлення m.

У3. Неможливо виходячи з алгоритму P_u знайти такий алгоритм S_u^*, що $S_u^*(P_u(m))=m$ для всіх m.

У4. $P_u(S_u(m))=m$ для будь-якого користувача U і будь-якого повідомлення m.

У5. Неможливо виходячи з алгоритму P_u знайти такий алгоритм S_u^*, що $P_u(S_u^*(m))=m$ для всіх m.

Умови **У3** і **У5** сформовані неточно і їх зміст має уточнюватись в залежності від області застосування

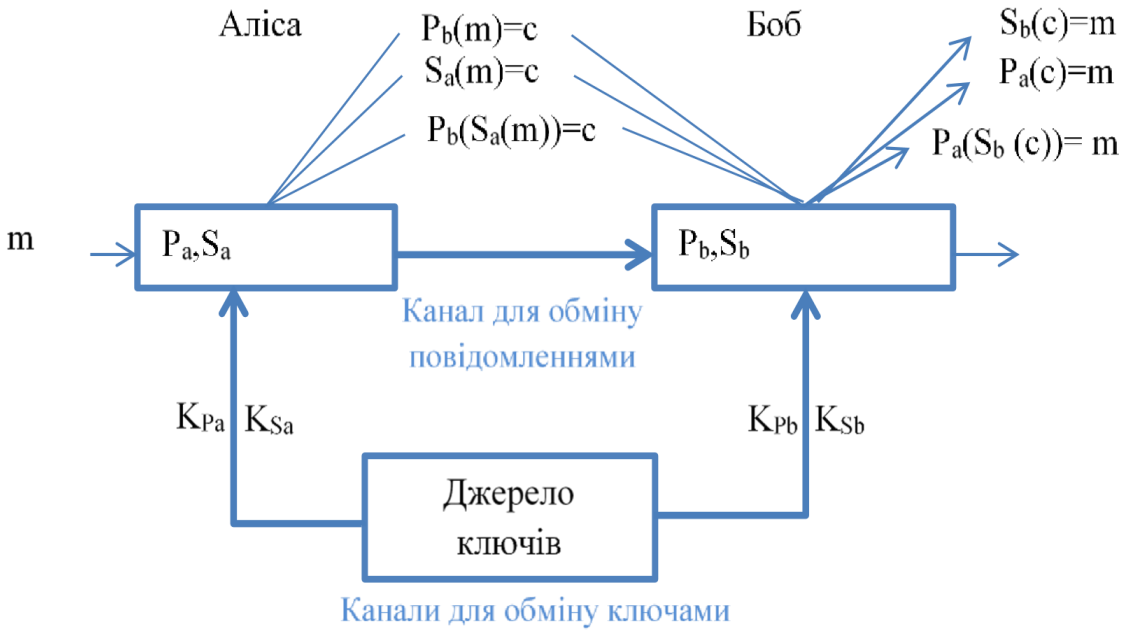


Рис.7.1. Модель асиметричної криптосистеми (Діффі-Хеллмана)

Розглянемо окремі випадки застосування цих умов.

1. Нехай виконуються умови **У1- У3**.

В цьому випадку забезпечується можливість **шифрування** повідомлень (Рис.7.2). Дійсно, якщо Аліса хоче надіслати Бобу зашифроване повідомлення m, то вона спочатку знаходить публічний ключ Боба P_B і за його допомогою шифрує m: c= P_B(m). Боб може відновити m, скористувавшись своїм секретним алгоритмом S_B, оскільки за умовою **У2** S_B(c)= S_B(P_B(m))= m. Виконання умови **У1** забезпечує практичність системи. А виконання умови **У3** дозволяє публікування алгоритму утворення відкритого ключа.

Рис.7.2. Шифрування в асиметричній криптосистемі

2. Нехай виконуються умови **У1,У4 і У5**.

В цьому випадку забезпечується можливість використання **цифрового підпису** (Рис.7.3). Дійсно, якщо Аліса хоче надіслати Бобу підписане повідомлення m, то вона може спочатку застосувати до нього секретний алгоритм S_A і відправити c= S_A(m). Боб може відновити m, скористувавшись публічним алгоритмом P_A, оскільки за умовою **У4** P_A(c)= P_A (S_A(m))= m. Виконання умови **У1** забезпечує практичність системи. А виконання умови **У5** дозволяє публікування алгоритму утворення цифрового підпису.

Рис.7.3. Цифровий підпис в асиметричній криптосистемі

3. Нехай виконуються всі умови **У1-У5**.

В цьому випадку забезпечується можливість **шифрування підписаних повідомлень** (Рис.7.4). Дійсно, якщо Аліса хоче надіслати Бобу повідомлення m в зашифрованому вигляді зі своїм підписом, то вона може спочатку підписати його за допомогою власного секретного алгоритму S_A, потім

зашифрувати за допомогою публічного алгоритму Боба P_B і відправити c= P_B (S_A(m)). Боб може відновити m, скористувавшись публічним алгоритмом P_A і своїм секретним S_B, оскільки за умовою **У2,У4** $P_A(S_B(c))= P_A(S_B(P_B(S_A(m))))=))= P_A(S_A(m))= m$.

Рис.7.4. Шифрування підписаного повідомлення в асиметричній криптосистемі

Виконання умови **У1** забезпечує практичність системи. А виконання умови **У3,У5** дозволяє публікування алгоритми шифрування і утворення цифрового підпису.

7.3. Поняття односторонньої функції-пастки

У. Діффі і Д.Хеллман запропонували використовувати для шифрування одностороннюю функцію-пастку.

Означення. Односторонньою функцією називається функція f: A→B з такими властивостями (Рис.7.5):

1. f(a) легко обчислюється для будь-якого a∈A.
2. Чисельно неможливо знайти f^{-1}(b) майже для всіх b B

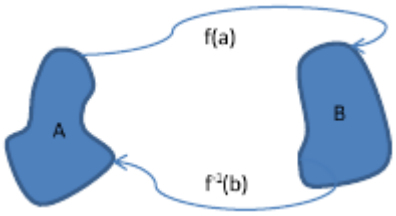


Рис.7.5. Відображення f і оберненого f^{-1}

Означення. Односторонньою функцією-пасткою (trapdoor function) називається така одностороння функція f(a), для якої f^{-1}(b), b ∈ B легко обчислюється, якщо відома деяка додаткова інформація (існує лазівка).

Означення. Одностороннюю функцію-пастку називають криптографічною функцією.

Приклад (криптографічної функції). Візьмемо телефонний довідник. Для шифрування кожної букви відкритого повідомлення із довідника випадково вибирається прізвище, що починається з такої букви. Відповідний телефонний номер утворює шифр для даного випадку появи букви; таким чином, система шифрування є поліалфавітною. Результат шифрування слова «шифр» може бути таким:

Буква	Прізвище	Шифр
ш	Шарапов	4013435
и	Ивліч	5556758
ф	Фоменко	4517890
р	Рудченко	6788399

Рис.7.6. Приклад шифрування з використанням телефонного довідника

Криптотекст утворюється написанням всіх номерів з правого стовпчика.

Отримати телефон за прізвищем - проста задача, а ось обернена задача визначення прізвища за номером значно складніша. Але вона теж може бути легко розв'язана, якщо відомий деякий секрет- лазівка. Таким секретом може бути обернений телефонний довідник.

7.4. Задача рюкзака

Першим алгоритмом для узагальненого шифрування з відкритим ключем став алгоритм рюкзака, розроблений Ральфом Меркле і Мартіном Хеллманом.

Проблему рюкзака можна сформулювати так: *Нехай задано множину натуральних чисел* $A = (a_1, a_2, \ldots, a_n)$ *і натуральне число S; потрібно встановити, чи існує така підмножина множини A, сума елементів якої б дорівнювала S.*

Еквівалентним є наступне формулювання: *Чи існує такий набір чисел* x_i *з (0,1),* $i \le n$*, для якого* $\Sigma a_i x_i = S$ $(1 <= i <= n)$*?*

Свою назву задача отримала в зв'язку з тим, що може бути поставлена також в наступному вигляді. Є набір предметів з відомими вагами і рюкзак, який

може витримати вагу, що не перевищує задану. Чи можна вибрати набір предметів для навантаження в рюкзак так, щоб вони в точності мали максимально допустиму вагу?

Приклад. Для набору з 10 цифр A = (43, 129, 215, 473, 903, 302, 561, 1165, 697, 1523) і S=3231 методом перебору можна знайти, що 3231 = 129 + 473 + 903 + 561 + 1165, тобто проблема рюкзака розв'язується.

В принципі рішення завжди може бути знайдено повним перебором підмножин A і перевіркою, яка з їх сум дорівнює S. Але при великих n доведеться перебрати 2^n варіантів. Навіть для n = 300 пошук серед 2^{300} підмножин не піддається обробці. Суть тут у тому, що невідомі алгоритми, які мають суттєво меншу складність у порівнянні з повним перебором.

"Проблема рюкзака" виявляється досить складною, її рішення з поліноміальною складністю в даний час не відомо.

Ідея побудови системи шифрування на основі проблеми рюкзака полягає у виділенні деякого підкласу задач про укладання рюкзака, що розв'язуються порівняно легко, і "маскування" задач цього класу (за допомогою деякого перетворення параметрів) під загальний випадок. Параметри підкласу визначають секретний ключ, а параметри модифікованої задачі - відкритий ключ.

В якості задачі, що легко розв'язується, Р. Меркль і М. Хеллман в 1978 р. запропонували задачу про укладання "суперзростаючого" рюкзака. Її суть полягає у такому.

Означення. Назвемо суперзростаючою послідовність таких натуральних чисел (b_1, b_2 ,..., b_n), що $b_i > \Sigma b_j$, для $1 <= j <= i-1$, $2 < i < n$.

Неважко переконатися в тому, що задача рюкзака для суперзростаючої послідовності може бути вирішена за допомогою виконання наступного алгоритму.

Алгоритм розв'язання задачі «суперзростаючого» рюкзака:

Введення: натуральні числа n,S, суперзростаюча послідовність натуральних чисел B=(b_1, b_2 ,..., b_n), набір чисел x_i з (0,1).

Виведення: такі b_i і x_i, для яких $\Sigma b_i x_i = S$.

Крок 1. Покласти i = n.

Крок 2. Якщо i> 1, то покласти x_i рівним 1 і S рівним (S - b_i), якщо S> b_i, і покласти x_i рівним 0 в іншому випадку.

Крок 3. Покласти і рівним (i-1) і повернутися до кроку 2.

Приклад. Для суперзростаючої послідовності з 5 цифр B = (43, 129, 215, 473, 903) і S=1161 знаходимо: x_5=1 - оскільки 903<1161; x_4=0 - оскільки 1161-903=258<473; x_3=1 - оскільки 215<258; x_2=0 - оскільки 258-215=43<129; x_1=1 - оскільки 43=43. Таким чином, X=(1,0,1,0,1). Дійсно, 43+215+903=1161.

Набір A з n значень визначає криптографічну функцію f (x) так. Будь-яке число x в інтервалі $0 \leq x \leq 2n - 1$ може бути задано двійковим представленням з n розрядів, в якому за необхідності добавляються початкові нулі. Таким чином, 1, 2 і 3 представляються в вигляді 0. . . 01, 0. . . 010, 0. . . 011, тоді як 1. . . 111 є представленням для (2n – 1). Тепер визначимо f (x) як число, що отримується з A сумуванням всіх таких a_i, у яких відповідний розряд в двійковому поданні x дорівнює 1. Так, f(1) = f (0... 001) = a_n, f(2)=f (0... 010) = a_n-1, f (3) = f (0... 011) = a_n-1 + a_n, і т. д.

Використовуючи векторне множення, можна записати, що **f** (x) = A**X**, де **X** - вектор-стовпець двійкового представлення x.

"Рюкзачні вектори" A можуть бути використані в якості основи криптосистеми. Для цього пропонується наступний механізм шифрування. Початкове повідомлення спочатку кодується і розбивається на n-розрядні блоки. Якщо це необхідно, останній блок доповнюється наприкінці нулями. Кожен з n-розрядних блоків шифрується за допомогою обчислення значення функції f для цього блоку.

Алгоритм шифрування в задачі рюкзака:

Введення: натуральне число n>1, послідовність натуральних чисел A=(a_1, a_2 ,..., a_n), вхідне повідомлення p.

Виведення: шифротекст C.

Крок 1. Представити p у вигляді бінарної послідовності.

Крок 2. Розбити отриману бінарну послідовність на n-розрядні блоки $p_i=p_{i1}p_{i2}\ldots p_{in}$.

Крок 3. Зашифрувати кожний блок за допомогою перетворення $C_i = \Sigma\, p_{ij} \cdot a_j$, j = 1 ... n. Крок 4. Отримати шифротекст C= (C_1; C_2; …; C_i)

Приклад. Якщо вихідне повідомлення написано по-англійськи, природним способом кодування є заміна однієї літери двійковим представленням її порядкового номера в алфавіті. Для цих цілей буде потрібно п'ять бітів. Співставим кожному символу 5-бітний код: пробіл -00000, A – 00001, B – 00010, C – 00011 і т.д.

Розглянемо попередній 10-набір і вихідне повідомлення HEALTH. Так як блоки шифрування складаються з 10 розрядів, то поділ на блоки вихідного тексту дасть:HE AL TH.

Відповідні 3 двійкових послідовності: 0100000101, 0000101100, 1010001000.

Для них f(0100000101) = 129 + 1165 + 1523 = 2817, f(0000101100) = 903 + 561 + 1165 = 2629, f(1010001000) = 43 + 215 + 561 = 819.

Отже, криптотекстом буде набір (2817, 2629, 819).

Визначену таким чином на основі функції рюкзака f (x) криптосистему можна використовувати як класичну. Тоді криптоаналітик знаходить n-набір A і після цього вирішує ще й задачу про рюкзак. Але для розшифрування легальний одержувач також повинен розв’язувати задачу про рюкзак. Це означає, що розшифрування однаково важко і для криптоаналітика, і для легального одержувача. Це означає, що криптосистема дуже недосконала. В

хороших криптосистемах розшифрування має бути значно важче для криптоаналітика, ніж для легального одержувача.

Удосконалити таку криптосистему і перетворити її на систему з відкритим ключем можна у такий спосіб.

Якщо розглядати суперзростаючий набір B як основу криптосистеми, то розшифрування буде однаково легким як для крипто аналітика, так і легального одержувача. Щоб уникнути цього, ми "збовтаємо" B таким чином, щоб результуючий набір A не був суперзростаючим і виглядав як довільний вектор рюкзака.

Для цього виберемо ціле $m > \Sigma b_i$. Так як B – суперзростаючий набір, то m велике в порівняно з усіма числами з B.

Виберемо інше ціле t, що не має спільних множників з m. Такий вибір t гарантує, що знайдеться інше ціле t^{-1}, таке, що $tt^{-1} \equiv 1 (\mathrm{mod}\ m)$. Ціле t^{-1} можна назвати зворотним до t. Зворотний елемент може бути легко обчислений по t і m.

Тепер утворюємо добутки tb_i, i = 1,. . . , n, і зведемо їх за модулем m: нехай a_i - найменший додатній залишок tb_i по модулю m. Результуючий вектор $A = (a_1, a_2..., a_n)$ розглядається як відкритий ключ для шифрування. Алгоритм шифрування для n-розрядних блоків вихідного повідомлення описаний вище.

Приклад. Нехай B = (1, 3, 5, 11, 21, 44, 87, 175, 349, 701), m=1590, t=43. Відмітимо, що t і m взаємно прості, оскільки $43 \cdot 37 = 1591 \equiv 1 \pmod{1590}$ і тому $t^{-1} = 37$.

Знаходимо значення A: $43 \cdot 1 = 43$, $43 \cdot 3 = 129$, $43 \cdot 5 = 215$, $43 \cdot 11 = 473$, $43 \cdot 21 = 903$, $43 \cdot 44 = 1892 = 1590 + 302$, $43 \cdot 87 = 3741 = 2 \cdot 1590 + 561$, $43 \cdot 175 = 7525 = 4 \cdot 1590 + 1165$, $43 \cdot 349 = 15007 = 9 \cdot 1590 + 697$, $43 \cdot 701 = 30143 = 18 \cdot 1590 + 1523$.

Числа t, t^{-1} і m зберігаються як секретна лазівка. Тоді легкий алгоритм розшифрування для легального одержувача полягатиме у такому. Оскільки легальний одержувач знає t^{-1} і m, він у змозі знайти A з відкритого ключа B. Після отримання блоку C криптотекста, який є цілим числом, легальний одержувач обчислює $C'=t^{-1}C$ і його найменший додатній залишок C' (mod m). При розшифруванні він вирішує легку задачу укладання рюкзака для значень B і C'. Рішенням є єдина послідовність p з n бітів. Вона також є і правильним блоком вихідного повідомлення, тому що будь-яке рішення p' задачі укладання рюкзака, яке визначається B і C', повинна дорівнювати p. Дійсно, $C \equiv t^{-1}C' = t^{-1}Bp' \equiv Ap'$ (mod m).

Алгоритм розшифрування в задачі рюкзака:

Введення: натуральне число n>1, суперзростаюча послідовність натуральних чисел $B=(b_1, b_2 ,..., b_n)$, натуральні числа $m> \Sigma b_i$, і $t\equiv1$(mod m), шифротекст $C= (C_1; C_2; \ldots; C_i)$.

Виведення: відкрите повідомлення p.

Крок 1. Знайти таке дійсне t^{-1}, що $tt^{-1} \equiv 1$(mod m).

Крок 2. Для кожного блоку шифротексту обчислити $C_i' \equiv t^{-1}C_i$ (mod m).

Крок 3. Розв'язати задачу «суперзростаючого» рюкзака для B і кожного C_i', отримавши відповідну бінарну послідовність p_i з n бітів.

Крок 4. Шляхом декодування p_i отримати текст повідомлення p.

Приклад. Розшифруємо отриманий раніше крипто текст (2817, 2629, 819). Множимо ці цифри на $t^{-1} = 37$ і знаходимо залишок по модулю 1590:

$37 \cdot 2817= 104229= 65 \cdot 1590 + 879 \equiv 879$ (mod 1590), $37 \cdot 2629= 372629 = 61 \cdot 1590 + 283 \equiv 283$ (mod 1590), $37 \cdot 819= 30303= 19 \cdot 1590 + 569 \equiv 93$ (mod 1590), Число 879 і суперзростаючий набір B дають 10-розрядну послідовність 0100000101. Дійсно, так як 879 > 701. Числа з A тепер порівнюються з різницею 879 - 701 = 178. Перше число, справа наліво, менше ніж 178, є 175. Наступне число 3 дорівнює різниці 178 - 175 = 3. Позиції 3, 175, і 701 у A є відповідно 2, 8 і 10. Декодуючи послідовність 0100000101, отримуємо перші дві літери тексту: НЕ.

Число 283 дає послідовність 00001011000. Перше число, справа наліво, менше ніж 283, є 175.

Наступне число менше різниці 283 - 175 = 108 є 87. Наступне число 21 дорівнює різниці 108 - 87 = 21. Позиції 21, 87, і 175 у A є відповідно 5, 7 і 8. Декодуючи послідовність 0100000101, отримуємо перші дві літери тексту: AL.

Декодуючи всі три послідовності, легальний одержувач обчислює вихідне повідомлення: HEALTH.

Запитання для самоконтролю

1. Які основні проблеми не вирішуються в класичній криптографії?
2. У чому сутність моделі криптосистеми з відкритим ключом, запропонованої Діффі і Хеллманом? Яку функціональність вона забезпечує?
3. Які функції називаються криптографічними?
4. У чому сутність задачі рюкзака?
5. Як на основі задачі рюкзака побудувати асиметричну криптосистему?

Результати навчання

Знання архітектури асиметричних криптосистем, можливостей асиметричного шифрування і цифрового підпису. Розуміння принципів і алгоритмів роботи асиметричної системи на основі задачі рюкзака.

Розділ 8. Протоколи асиметричної криптографії

8.1 Протокол Діффі-Хеллмана

Проблема розподілу ключів була вирішена в тій же основній роботі Діффі і Хеллмана, в якій була розроблена схема шифрування з відкритим ключем. Цей протокол розподілу ключів, названий протоколом Діффі-Хеллмана обміну ключами, дозволяє двом сторонам узгодити секретний ключ по відкритому каналу зв'язку. Стійкість протоколу ґрунтується на складності проблеми дискретного логарифмування в кінцевій абелевій групі.

В алгоритмі Діффі-Хеллмана симетричний сеансовий ключ не генерується і не розподіляється між учасниками. Алгоритм забезпечує формування одного й того ж секрету двома сторонами, який можна використовувати для побудови сеансового ключа в симетричному алгоритмі. Ця процедура називається *погодженням ключа*: сторони погоджують ключ, який будуть використовувати. Вона полягає у такому: кожна із сторін отримує секретне і відкрите значення (пара ключів ДіффіХеллмана); об’єднання секретного значення однієї із сторін з відкритим значенням іншої забезпечує створення одного й того ж самого секретного значення.

Протокол **Діффі-Хеллмана** спрощується, якщо зв’язок здійснюється тільки між двома абонентами. В цьому випадку:

1. Спочатку генеруються два великих простих числа n і q. Ці два числа не обов'язково зберігати в секреті.
2. Один з партнерів A генерує випадкове число *x* і посилає іншому учаснику B значення $L=Q^x(\text{mod } N)$.
3. Після отримання L партнер B генерує випадкове число *y* і посилає A обчислене значення $M=Q^y \text{ (mod } N)$.
4. Партнер A, отримавши M, обчислює $K_x = M^x \text{ (mod } N)$, а партнер B обчислює $K_y = L^y \text{ mod } N$.

Алгоритм гарантує, що $K_y=K_x$ і можуть бути використані в якості секретного ключа для шифрування. Адже навіть перехопивши числа L і M, важко обчислити K_x або K_y.

Основні повідомлення в протоколі Діффі-Хеллмана представляються наступною діаграмою :

A↔B: N,Q;

A: x; A→B: $L=Q^x(\text{mod } N)$;

B: y; B→A: $M=Q^y \text{ (mod } N)$;

A: $K_x = M^x \text{ (mod } N)$;

B: $K_y = L^y \text{ mod } N)$;

Приклад. Візьмемо N = 2147483 659 і G = 2. Тоді маємо:

A	**B**
x=12	y=34
$L=2^{12}(\text{mod } 2\ 147\ 483\ 659)= 4096$	$M=2^{34}(\text{mod } 2\ 147\ 483\ 659)= 2147483571$
$K_x = 2147483571^{12}(\text{mod } 2\ 147\ 483\ 659)= 1082609919$	$K_y = 4096^{34} (\text{mod } 2\ 147\ 483\ 659)= 1082609919$

Розглянута система має суттєвий недолік: у Аліси немає впевненості, що вона листується саме з Бобом. Це може привести до наступної атаки, яка умовно називається «людина посередині»:

- Аліса домовляється про ключ з Євою, думаючи, що переписується з Бобом;
- Боб веде переговори про ключ з Євою, вважаючи, що переписується з Алісою;

- Єва може вивчати повідомлення, т.я. вони проходять через неї як через комутатор. Оскільки вона не вносить змін в відкритий текст, її дії не можуть бути виявлені.

Отже, можна зробити висновок про те, що самого по собі протоколу Діффі-Хеллмана не достатньо для забезпечення секретності розподілення ключів. Але на основі протоколу ДіффіХеллмана можна створити справжню криптосистему, яка називається системою Ель-Гамаля.

8.2 Шифр Шаміра

Перший шифр, який дозволив організувати обмін секретними повідомленнями по відкритим лініям для осіб, які не мають ніяких захищених каналів і секретних ключів, був запропонований Шаміром (Adi Shamir). Наведемо його опис.

Нехай абонент А хоче передати повідомлення m абоненту В по відкритій лінії зв'язку. Для цього: абонент А вибирає випадкове велике просте число p і відкрито передає його абоненту В. Потім абонент А вибирає два секретні числа c_A і d_A такі, що $c_A d_A \bmod (p-1) = 1$. Абонент В також вибирає два секретні числа c_B і d_B такі, що $c_B d_B \bmod (p-1) = 1$.

Після цього абонент А передає своє повідомлення m за допомогою три ланкового протоколу. Якщо m<p, то повідомлення m передається зразу; якщо ж m>=p, то повідомлення представляється у вигляді $m_1, m_2, \ldots, m_t$, де всі m_i<p, а потім передаються послідовно $m_1, m_2, \ldots, m_t$. При цьому для кожного m_i рекомендується вибирати випадково нові пари (c_A ,d_A) і (c_B ,d_B), в іншому випадку надійність системи стає нижчою. Такий протокол може використовуватись для передачі секретних ключів.

Далі розглядається тільки випадок m<p.

Протокол Шаміра складається з таких кроків:

1. Абонент А обчислює число $x_1 = m^{c_A} \bmod p$ і пересилає його абоненту В.

2. Абонент В, отримавши x_1, обчислює $x_2 = x_1^{c_B} \, mod \, p$ і пересилає його абоненту А.
3. Абонент А, отримавши x_2, обчислює число $x_3 = x_2^{d_A} \, mod \, p$ і пересилає його абоненту В.
4. Абонент В, отримавши x_3, обчислює $x_4 = x_4^{d_B} \, mod \, p$ і пересилає його абоненту А.

Неважко довести, що: (а)x_4=m; (б) зловмисник не може знати m.

Дійсно, будь-яке ціле число e>=0 можна представити у вигляді e=k(p-1)+r, r=e mod (p-1).

За теоремою Ферма: $x^e \, mod \, p = x^{k(p-1)+r} mod \, p = x^r mod \, p.$

Тому $x_4 = x_{3d_B} \, mod \, p = (x_{2d_A})_{d_B} \, mod \, p = (x_{1c_B})_{d_A d_B} \, mod \, p = (m_{c_A})_{c_B d_A d_B} \, mod \, p =$

$m_{c_A c^B d^A d^B} \, mod \, p = m_{c_A c^B d^A d^B \, mod \, (p-1)} \, mod \, p = m_{c_A c^B d^A d^B \, mod \, (p-1)} \, mod \, p =$

$m_{c_A c_B \, mod \, (p-1) d^A d^B \, mod \, (p-1)} \, mod \, p = m_{1*1} \, mod \, p = m.$

Зловмиснику для знаходження m доведеться розв'язувати задачу дискретного логарифмування, що для великих p неможливо.

Для знаходження пар (c_A ,d_A) і (c_B ,d_B) можна запропонувати наступний спосіб. Перше число (c_A)вибирається випадково так, щоб воно було взаємно простим з (p-1), а друге число (d_A) знаходиться за допомогою розширеного алгоритму Евкліда.

Приклад. Нехай А хоче передати В m=10. А вибирає p=23, c_A=7 (оскільки gcd(7,22)=1) і обчислює d_A=19. Аналогічно, В вибирає c_B=5 і обчислює d_B=9. Переходимо до протоколу Шаміра:

1. $x_1=10^7$ mod 23=14.
2. $x_2=14^5$ mod 23=15.
3. $x_3=15^{19}$ mod 23=19.
4. $x_4=19^9$ mod 23=10.

Таким чином, В отримав m=10.

8.3 Шифр Ель-Гамаля

Шифр, запропонований Ель-Гамалем (Taher ElGamal), як і шифр Шаміра, вирішує задачу передачі між абонентами зашифрованих повідомлень по незахищених каналах, але використовує при цьому лише одну пересилку повідомлення. Фактично тут використовується протокол ДіффіХеллмана для формування секретного ключа і далі повідомлення шифрується шляхом множення його на цей ключ.

Для групи абонентів A,B,C,… вибирається велике просте число p і деяке число q, 1<q<p-1, таке, що всі числа з множини {1,2,…, p-1} можуть бути представлені як різні степені числа q mod p. Числа p і q передаються всім абонентам у відкритому вигляді.

Потім кожний абонент вибирає своє секретне число c_i, 1< c_i <p-1 і обчислює відповідне йому число

$$d_i = q^{c_i} \bmod p$$

В результаті отримуємо таблицю ключів:

Абонент	Секретний ключ	Відкритий ключ
A	c_A,	d_A
B	c_B	d_B
C	c_C	d_C
…	…	…

Покажемо, як A пересилає B повідомлення m:

1. Абонент A вибирає випадкове число k, 1<=k<=p-2, обчислює числа $d_A = q^{c_A} \bmod p$, $e = m \cdot d_B^{c_A} \bmod p$ p і передає пару чисел (r,e) абоненту B.
2. Абонент B, отримавши (r,e), обчислює $m' = e \cdot d_A^{\,p-1-c_B} \bmod p$.

Неважко довести, що: (а) m' = m; (б) зловмисник не може знати m. Дійсно, $m' = e \cdot d_A^{\,p-1-c_B} \bmod p = m \cdot d_B^{c_A} \cdot d_A^{\,p-1-c_B} \bmod p = m \cdot (q^{c_B})^{c_A} \cdot$

$(q^{c_A})^{p-1-c_B} \bmod p = m \cdot q^{c_B c_A + k(p-1) - c_A c_B} \bmod p = m \cdot q^{c_A(p-1)} \bmod p = m \cdot 1^{c_A} = m.$

Зловмиснику для знаходження m доведеться розв'язувати задачу дискретного логарифмування, що для великих p неможливо.

Приклад. Нехай m=15. Візьмемо p=23, q=5. Нехай абонент B обрав для себе секретне число c_B=13 і обрахував $d_B=5^{13} \bmod 23=21$. Абонент A вибирає випадкове k=7 і обчислює $r=5^7 \bmod 23=17$, $e=15*21^7 \bmod 23=12$. Абонент A надсилає абоненту B пару чисел (17,12). Абонент B знаходить $m'=12*17^{23-1-13} \bmod 23=12*17^9 \bmod 23=12*7 \bmod 23 =15$.

Зауважимо, що об'єм шифру в 2 рази перевищує об'єм повідомлення, але вимагається лише одна передача даних.

8.4 Шифр RSA

Система RSA, названа на честь його розробників Ріверса (Ron Rivers), Шаміра (Adi Shamir) і Адлемана (Leonard Adleman).

Система ґрунтується на використанні іншої односторонньої функції. В цій системі використовуються наступні факти з теорії чисел:

1. Задача перевірки числа на простоту є порівняно простою.
2. Задача розкладання числа n=p*q, де p і q – прості числа, на множники є дуже складною задачею, якщо ми знаємо тільки n, а p і q – великі числа (задача факторизації).

В групі абонентів A,B,C,… кожний абонент:

1. Вибирає випадково два великих простих числа P і Q і обчислює $N= PQ$.
2. Обчислює число $f=(P-1)(Q-1)$.
3. Вибирає деяке число d<f, взаємно просте з f, і за розширеним алгоритмом Евкліда знаходить таке c, що $cd \bmod f=1$.

В результаті отримуємо таблицю ключів:

Абонент	Секретний ключ	Відкритий ключ
A	P_A, Q_A,c_A	N_A, d_A,
B	P_B, Q_B,c_B	N_B, d_B,
C	P_C, Q_C,c_C	N_C, d_C,
…	…	…

Нехай A хоче передати B повідомлення m< N_B. Протокол передачі складається з таких кроків:

1. Абонент A шифрує повідомлення, використовуючи відкриті параметри абонента B: $\boxed{e = m^{d_B} \ mod \ N_B}$
2. . Абонент B, отримавши зашифроване повідомлення, обчислює $\boxed{m' = e^{c_B} \ mod \ N_B}$.

Неважко довести, що $m' = m$. Дійсно, $m' = e^{c_B} \ mod \ N_B = m^{d_B c_B} \ mod \ N_B$. Але $d_B c_B = k \cdot f_B + 1$, а $f_B = (P_B - 1)(Q_B - 1) = \varphi(N_B)$, де φ – функція Ейлера. Оскільки за теоремою Ейлера $m^{\varphi(NB)} \equiv 1 (mod \ N_B)$, то $m' = m_{k\varphi(NB)+1} \ mod \ N_B = m$.

Відмітимо, що для RSA важливо, щоб кожний абонент вибирав власну пару простих чисел P і Q, тобто всі N мають різнитись,інакше один абонент міг би читати повідомлення, призначені для іншого. Проте інший відкритий параметр d може бути однаковим у всіх абонентів. Часто рекомендується d=3. Тоді шифрування виконується максимально швидко – всього за дві операції множення.

Приклад. Нехай абонент A передає абоненту B повідомлення m=15. Нехай абонент B обрав для себе секретне число P_B=3, Q_B=11, N_B=33 і d_B=3.За допомогою розширеного алгоритму Евкліда знаходимо c_B=7. Тоді e=15^3 mod 33=9. Це число передається абоненту B. Тільки абонент B знає c_B, тому він розшифровує m'=9^7 mod 33=15.

8.5 Програмна реалізація алгоритму Діффі-Хеллмана засобами .NET

Платформа .NET надає реалізацію CNG алгоритму Діффі-Хеллмана на еліптичних кривих (ECDH)

через використання об’єктів класу **ECDiffieHellmanCng** з простору імен **System.Security.Cryptography**.

Клас **ECDiffieHellmanCng** дозволяє двом сторонам обмінюватися матеріалом закритих ключів, навіть якщо взаємодія здійснюється по відкритим каналам. Обидві сторони можуть обчислити одне і те ж таємне значення, яке називається **секретною угодою** в керованих класах

Діффі-Хеллмана. Секретна угода може надалі використовуватися для різних цілей, в тому числі як симетричний ключ. Проте, замість прямого представлення секретної угоди клас **ECDiffieHellmanCng** робить деяку її обробку перед наданням значення. Ця постобробка називається **функцією формування ключа** (key derivation function, KDF). Можна вибрати, яку функцію формування ключа використовувати, і задати її параметри через набір властивостей в екземплярі об'єкта Діффі-Хеллмана. Результат передачі секретної угоди через функцію створення ключа представляє собою масив байтів, який можна використовувати як матеріал ключа. Кількість байтів створеного матеріалу ключа залежить від функції формування ключа; наприклад SHA-256 створить 256 бітів матеріалу ключа, а SHA-512 - 512 бітів матеріалу ключа.

Обмін ключами ECDH відбувається наступним чином:

1. Аліса і Боб створюють пару ключів для операції обміну ключами Діффі-Хеллмана.
2. Аліса і Боб налаштовують функцію формування ключа з використанням обумовлених параметрів.
3. Аліса відправляє свій відкритий ключ Бобу.
4. Боб відправляє свій відкритий ключ Алісі.

5. Аліса і Боб використовують відкриті ключі один одного для створення секретної угоди і застосовують функцію формування ключа для створення матеріалу ключа.

Порядок узгодження ключа шифрування між Алісою і Бобом за допомогою об’єктів класу **ECDiffieHellmanCng** такий:

1. Аліса створює сховище ключів і експортує з нього свій публічний ключ для передачі Бобу: `CngKey aliceCngKey = CngKey.Create(CngAlgorithm.ECDiffieHellmanP256);`

`byte[] alicePublicKeyBlob = aliceCngKey.Export(CngKeyBlobFormat.EccPublicBlob);`

2. Боб створює сховище ключів і експортує з нього свій публічний ключ для передачі Алісі:

```
CngKey bobCngKey = CngKey.Create(CngAlgorithm.ECDiffieHellmanP256);
byte[] bobPublicKeyBlob = aliceCngKey.Export(CngKeyBlobFormat.EccPublicBlob);
```

3. Аліса імпортує публічний ключ Боба в окреме сховище:

```
CngKey bobPubCngKey = CngKey.Import(bobPublicKeyBlob,
CngKeyBlobFormat.EccPublicBlob);
```

4. Аліса створює екземпляр класу ECDiffieHellmanCng з ключами, що беруться з її сховища ключів:

```
ECDiffieHellmanCng aliceAlgorithm = new ECDiffieHellmanCng(aliceCngKey)
```

5. Аліса отримує секретний ключ з ключового матеріалу:

```
byte[] aliceKey = aliceAlgorithm.DeriveKeyMaterial(bobPubCngKey);
```

6. Боб імпортує публічний ключ Аліси в окреме сховище і використовує його для отримання секретного ключа з ключового матеріалу:

```
CngKey alicePubCngKey = CngKey.Import(alicePublicKeyBlob,
CngKeyBlobFormat.EccPublicBlob);
ECDiffieHellmanCng bobAlgorithm = new ECDiffieHellmanCng(bobCngKey); byte[] bobKey
= bobAlgorithm.DeriveKeyMaterial(alicePubCngKey);
```

Слід зауважити, що обмінюватись публічними ключами зручніше, якщо вони представлені у форматі XML.

Експортувати з алгоритму публічний ключ у XML-формат дозволяє метод **ToXmlString():**

```
ECDiffieHellmanCng bobAlgorithm = new ECDiffieHellmanCng(bobCngKey); string xmlBobPublicKey = bobAlgorithm.PublicKey.ToXmlString();
```

Імпортувати до алгоритму публічний ключ з XML-формату дозволяє метод **FromXmlString(),** після чого він може бути конвертований в байтовий масив за допомогою методу **ToByteArray():**

ECDiffieHellmanCng bobAlgorithm = new ECDiffieHellmanCng(bobCngKey); bobAlgorithm.FromXmlString(xmlBobPublicKey,ECKeyXmlFormat.Rfc4050); byte[] bobPublicKeyBlob = bobAlgorithm.PublicKey.ToByteArray(); Схематично описаний порядок узгодження показано на Рис.8.1.

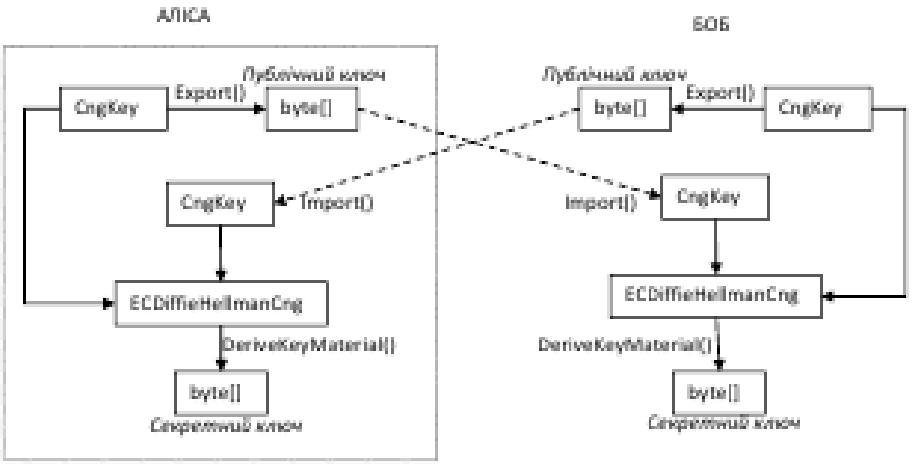


Рис.8.1. Схема реалізації алгоритму Діффі-Хеллмана

Лістінг програми узгодження спільного секрету за допомогою протоколу Діффі-Хеллмана наведений в додатку 2.

8.6 Програмна реалізація алгоритму RSA засобами .NET

На платформі .NET алгоритм RSA реалізується за допомогою об’єктів класу **RSACryptoServiceProvider** з простору імен **System.Security.Cryptography**.

Генерація набору, що складається з відкритого та закритого ключів, здійснюється при створенні нового екземпляра класу.

Після створення нового екземпляра класу можна отримати інформацію про ключ одним із двох способів:

1. Метод **ToXMLString** – повертає інформацію про ключ в форматі XML.
2. Метод **ExportParameters** – повертає структуру **RSAParameters**, що містить ключові відомості.

Обидва методи приймають як параметр логічне значення, яке показує:

- **false** – слід повертати відомості тільки про відкритий ключ;
- **true** – слід повертати відомості і про відкритий, і про закритий ключі.

Ініціалізація класу RSACryptoServiceProvider може бути здійснена також двома шляхами:

1. Метод **FromXmlString** – використовує дані ключа з рядка XML.
2. Метод **ImportParameters** – використовує дані структури RSAParameters.

Асиметричні закриті ключі ніколи не повинні зберігатися в роздрукованому вигляді або у вигляді простого тексту на локальному комп'ютері. Якщо необхідно зберігати закритий ключ, слід використовувати для цього *контейнер ключа*.

Контейнер ключа представляє собою екземпляр класу **CspParameters** (з простору імен

System.Security.Cryptography). В полі **CspParameters.KeyContainerName** задається ім’я контейнера.

Створити екземпляр класу **CspParameters** і зберегти в ньому згенерований ключ можна так:

```
CspParameters cp = new CspParameters(); cp.KeyContainerName
= “ContainerName”;
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(cp);
Console.WriteLine("Key added to container: \n {0}", rsa.ToXmlString(true));
```

Передача ключів провайдеру здійснюється через передачу імені контейнера конструктору об’єкта класу **RSACryptoServiceProvider**.

Отримати ключ з контейнеру можна так:

```
CspParameters cp = new CspParameters(); cp.KeyContainerName
= “ContainerName”;
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(cp);
Console.WriteLine("Key retrieved from container : \n {0}", rsa.ToXmlString(true));
```

Добавлений ключ можна видалити з контейнера. Для цього в екземплярі класу

RSACryptoServiceProvider необхідно спочатку відмовитись від збереження ключа, присвоївши властивості **PersistKeyInCSP** значення **false**, а потім скористатись методом **Clear**() для вивільнення всіх ресурсів класу і видалення контейнера ключа.

```
CspParameters cp = new CspParameters(); cp.KeyContainerName = ContainerName;
   RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(cp);
rsa.PersistKeyInCsp = false; rsa.Clear();
   Console.WriteLine("Key deleted.");
```

Порядок розшифрування за допомогою об’єктів класу **RSACryptoServiceProvider** такий:

1. Створюється контейнер для збереження ключів:

```
CspParameters cp = new CspParameters(); cp.KeyContainerName
= "Key Name";
```

2. Створюється екземпляр криптопровайдера з розміщенням ключів у контейнері:

```
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(cp)
```

3. Публічний ключ експортується для передачі іншій стороні:

```
string pubKey = rsa.ToXmlString(false);
Console.WriteLine("Public Key: \n {0}", pubKey);
```

4. Після отримання байтових даних *byte[] EncryptBytes*, зашифрованих за допомогою публічного ключа, здійснюється їх розшифрування за допомогою закритого ключа:

```
byte[] DecryptBytes = rsa.Decrypt(EncryptBytes, false); string decryptStr = Encoding.Unicode.GetString(DecryptBytes);
//string decryptStr =BitConverter.ToString(DecryptBytes);
Console.WriteLine("Decripted string: \n {0}", decryptStr);
```

Порядок шифрування полягає у такому:

1. Створюється екземпляр криптопровайдера :

```
RSACryptoServiceProvider rsa1 = new RSACryptoServiceProvider()
```

2. Імпортується публічний ключ: rsa1.FromXmlString(pubKey);

3. Текст повідомлення перетворюється у байтову послідовність і зашифровується публічним ключем:

```
string dataToEncrypt = "Data to encrypt"; byte[] byteToEncript = Encoding.Unicode.GetBytes(dataToEncrypt); byte[] EncryptBytes = rsa1.Encrypt(byteToEncript, false);
```

4. Зашифрована байтова послідовність відправляється стороні, яка має для розшифрування відповідний закритий ключ.

Описаний процес шифрування за алгоритмом RSA схематично показано на Рис.8.2.

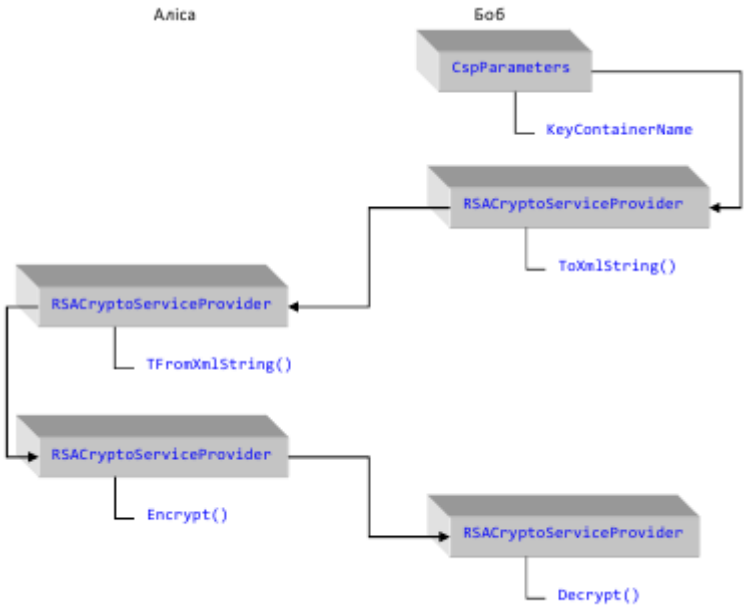


Рис.8.2. Схема реалізації алгоритму RSA

Зауваження. Для перетворення текстових даних в байтові і навпаки можна також скористуватись екземпляром класу UnicodeEncoding з простору імен System.Text. Цей клас надає кодування символів Юнікоду у форматі UTF-16. Наприклад, перетворити текстові дані в байтові можна так:

```
UnicodeEncoding ByteConverter = new UnicodeEncoding(); byte[]
byteToEncript = ByteConverter.GetBytes(dataToEncrypt);
```

Запитання для самоконтролю

1. Яке призначення протоколу Діффі-Хеллмана? У чому його зміст?
2. Як здійснюється шифрування за протоколом Шаміра?
3. Як здійснюється шифрування за протоколом Ель-Гамаля?
4. Які асиметричні алгоритми шифрування реалізовані на платформі .Net?
5. Які класи .Net забезпечують прикладну реалізацію протокола Діффі-Хеллмана?
6. На складності розвязання якої задачі грунтується алгоритм RSA?
7. У чому полягає зміст алгоритму RSA?

8. Який клас криптопровайдера .Net містить реалізацію алгортиму RSA?

Результати навчання

Знання класичних протоколів з відкритим ключем (Діффі-Хеллмана, Шаміра, Ель-Гамаля) та уміння з реалізації на платформі .Net протоколу Діффі-Хеллмана. Знання принципів асиметричного шифрування на основі алгоритму RSA. Уміння з його програмної реалізації засобами платформи .Net.

Розділ 9. Цифровий (електронний) підпис

9.1. Загальна схема використання

Електронно-цифровий підпис (ЕЦП) – вид електронного підпису, отриманого за результатом криптографічного перетворення набору електронних даних, який додається до цього набору або логічно з ним поєднується і дає змогу підтвердити його цілісність та ідентифікувати підписувача.

ЕЦП накладається за допомогою особистого ключа та перевіряється за допомогою відкритого ключа.

Схема застосування цифрового підпису (Рис.9.1):

1. Беремо вихідне повідомлення і створюємо 160-бітовий геш (дайджест повідомлення) за допомогою алгоритму SHA-l.
2. Потім дайджест повідомлення шифрується за допомогою секретного ключа, відомого тільки його власнику, тобто відправнику повідомлення.
3. Будь-який бажаючий може розшифрувати геш, користуючись загальнодоступним відкритим ключем.
4. Результат цього шифрування і називають цифровим підписом.
5. Підписане повідомлення формується об'єднанням вихідного повідомлення, його цифрового підпису та відкритого ключа.

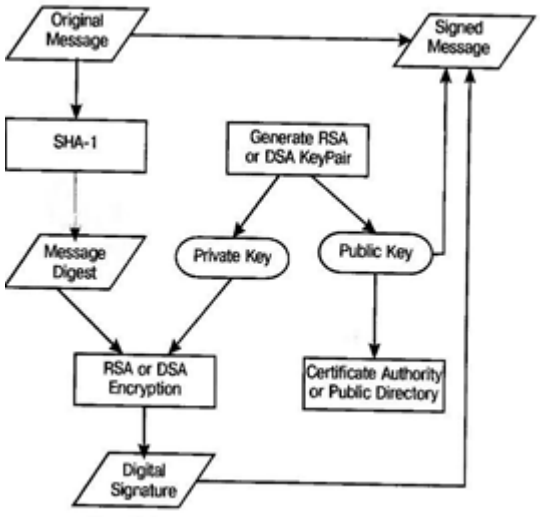


Рис.9.1. Схема створення цифрового підпису

Схема перевірки цифрового підпису (рис.9.2):

1. Отримане повідомлення розбивається на три компоненти: на вихідне повідомлення, відкритий ключ і цифровий підпис.
2. Заново обчислюється геш повідомлення.
3. Якщо обчислений геш збігається з розшифрованим гешем, то перевірка підпису пройдена.

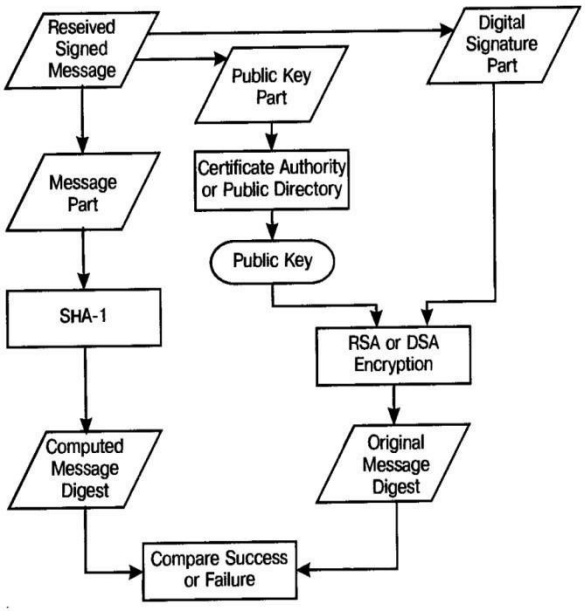


Рис.9.2. Схема перевірки цифрового підпису

9.2. Цифровий підпис на основі шифру RSA

Створити цифровий підпис можна на основі схеми RSA. Так, якщо Аліса має намір використовувати цифровий підпис, вона:

1. Вибирає два великих простих числа P і Q.
2. Обчислює N=PQ і f=(P-1)(Q-1).
3. Вибирає d, взаємно просте з f.
4. Обчислює $c=d^{-1} \bmod f$.
5. Зберігає c в якості секретного ключа (інші числа – P,Q і f більше не потрібні).

6. Публікує N і d в якості відкритого ключа.

Тепер Аліса готова до використання цифрового підпису. Нехай вона хоче підписати повідомлення $m= m_1, \dots , m_n$. Тоді вона:

1. Спочатку обчислює так називану геш-функцію $h =h(m_1, \dots , m_n)$. Найбільш важлива особливість цієї функції в тому, що практично неможливо змінити текст $m_1, \dots , m_n$, не змінивши h. Вважається, що алгоритм її обчислення загальновідомий.
2. Обчислює число $s= h^c \bmod N$, яке і є цифровим підписом.
3. Додає s до повідомлення m, формуючи підписане повідомлення <m,s> .

Тепер кожний, хто знає відкритий ключ Аліси, може перевірити її справжність. Для цього йому необхідно:

1. Обрахувати геш-функцію $h =h(m_1, \dots , m_n)$.
2. Обрахувати $w=s^d \bmod N$..
3. Перевірити виконання рівності w = h.

Неважко переконатись у тому, що у випадку справжнього підпису w = h. Дійсно, $w=s^d \bmod N= h^{cd} \bmod N= h$.

Приклад. Нехай P=5 і Q=11. Тоді N=5*11=55, f=4*10=40. Візьмемо d=3, т.я. gcd(40,3)=1. За допомогою розширеного алгоритму Евкліда знаходимо $c=3^{-1} \bmod 40 =27$.

Нехай значення геш-функції повідомлення y=13. Тоді Аліса знаходить $s = 13^{27} \bmod 55=7$.

Для перевірки підпису обраховуємо $w=7^3 \bmod 55=13$. Оскільки це значення співпадає зі значенням геш-функції, підпис справжній.

Цифровий RSA-підпис має два суттєвих недоліки:

1. Він є дуже великим, а іноді необхідно, щоб підпис займав мало місця.
2. Генерація RSA-підпису – надто дорога операція, що унеможливлює його використання в окремих додатках.

9.3. Цифровий підпис на основі шифру Ель-Гамаля

У той час, як цифровий підпис RSA ґрунтується на складності розв'язання задачі факторизації, цей підпис ґрунтується на складності розв'язання задачі дискретного логарифмування.

Нехай Аліса має намір використовувати цифровий підпис. Для цього вона:

1. Вибирає велике просте число p.
2. Вибирає таке число q, що різні степені q є різними по модулю p.
3. Публікує p і q у відкритому виді.
4. Вибирає випадкове число x, $1<x<p-1$, і зберігає його в якості секретного ключа.
5. Обчислює число $y=q^x \bmod p$.
6. Публікує y в якості відкритого ключа.

Тепер Аліса може користуватись цифровим підписом. Нехай вона хоче підписати повідомлення $m= m1, \dots , m_n$. Тоді вона:

1. Обчислює значення геш-функції $h=h(m)$, яке має задовольняти нерівності $1<h<p$.
2. Вибирає випадкове число k ($1<k<p-1$), взаємно просте з (p-1), і обчислює число $r=q^k \bmod p$.
3. Обчислює числа $u=(h-xr) \bmod (p-1)$, $s=k^{-1}u \bmod (p-1)$. Тут під k^{-1} розуміється число, що задовольняє рівняння $k^{-1}k \bmod (p-1)$ і знаходиться за розширеним алгоритмом Евкліда.
4. Формує підписане повідомлення <m, r, s>.

Тепер кожний, хто знає відкритий ключ Аліси, може перевірити її справжність. Для цього йому необхідно:

1. Обрахувати геш-функцію $h=h(m)$.

2. Перевірити виконання рівності $y^r r^s = q^h \bmod p$.

Неважко переконатись у тому, що у випадку виконання рівності $y^r r^s = q^h \bmod p$ підпис є справжнім.

Дійсно, $y_r r_s = (q_x)_r (q_k)_s = q_{xr} q_{k(k^{-1}(h-xr))} = q_{xr} q_h q_{-xr} = q_h \, mod \, p.$

Приклад. Нехай загальними параметрами для деякої спільноти вибрані p=23 і q=5. Аліса вибирає свій секретний ключ x=7 і обчислює $y=5^7 \bmod 23=17$.

Нехай значення хеш-функції повідомлення h(m)=3. Тоді Аліса генерує випадкове число k, наприклад, k = 5. Тоді $r=5^5 \bmod 23=20$, $u=(3-7*20) \bmod 22=17$.

Далі Аліса знаходить $k^{-1} \bmod 22 = 5^{-1} \bmod 22 = 9$ і $s = 9*17 \bmod 22 = 21$, а потім формує повідомлення<m,20,21>

Для перевірки підпису Боб знаходить значення хеш-функції і обраховуємо $y^r r^s = 17^{20} * 20^{21} \bmod 23 = 16*15 \bmod 23 = 10$ і після цього $q^h \bmod p = 5^3 \bmod 23 = 10$. Оскільки ці значення співпадають, підпис справжній.

Розглянутий метод складніше RSA, але на його основі може бути побудований більш ефективний алгоритм, в якому час обрахунків значно скорочується за рахунок використання «коротких» показників степенів. Він розглядається далі.

9.4. Стандарт DSA

В DSA використовуються наступні параметри домена:

1. p – просте число p, де $2^{L-1} < p < 2^L$, $512 =< L =< 1024$ и L кратно 64;
2. q – простий дільник p-1, при цьому $2159 < q < 2160$;
3. g = h(p-1)/q mod p, де h – будь-яке ціле число $1 < h < p - 1$ таке, що h(p-1)/q mod p > 1;
4. x – випадкове ціле число, $0 < x < q$; 5. y = gx mod p;

6. k – випадкове ціле число, $0 < k < q$.

Генерація підпису виконується наступним способом:

1. $r = (g^k \bmod p) \bmod q$
2. $s = (k^{-1}(SHA(M) + xr)) \bmod q$.
3. Якщо r = 0 або s = 0, має бути згенеровано нове k і обчислений новий підпис.

Тут SHA (M) - 160-бітний бінарний рядок, що отримується в наслідок застосування геширування за алгоритмом SHA-1.

Перевірка підпису:

Числа p, q, g і відкритий ключ знаходяться у відкритому доступі. Нехай M ', r' і s 'отримані версії M, r і s, відповідно, і нехай y - відкритий ключ. При перевірці підпису спочатку потрібно подивитися, чи виконуються наступні нерівності: 0 <r '<q, 0 <s '<q. Якщо хоча б одне нерівність не виконано, підпис повинна бути відкинута. Якщо умови нерівностей виконані, виробляються наступні обчислення:

1. $w = (s')^{-1} \bmod q$
2. $u_1 = ((SHA (M ')\ w) \bmod q$
3. $u_2 = ((r')\ w) \bmod q$
4. $v = (((g)^{u1} (y)^{u2}) \bmod p) \bmod q$.

Якщо v = r ', то справжність підпису підтверджена.

Приклад. Виберем такі параметри домену: q=13, p=4q+1=53 і $g=2^4 \bmod 53=16$. Нехай ключова пара користувача приймає значення x=3, $y=16^3 \bmod 53 =15$.

Будемо вважати, що геш-функція повідомлення SHA(M)=5. В якості ефемерного ключа виберемо k=2.

Генеруємо підпис: $r = (16^2 \bmod 53) \bmod 13 =5$, $s = ((5 + 3*5)/2) \bmod 13=10$.

Перевірка підпису: $w = (10)^{-1} \bmod 13 = 4$, $u_1 = (5*4) \bmod 13=7$, $u_2 = (5*4) \bmod 13=7$, $v = (16^7*15^7) \bmod 53) \bmod 13=5$.

9.5. Стандарт ГОСТ Р34.10-94

Параметри домена теж схожі на параметри домена в алгоритмі Ель-Гамаля:

1. Вибираються два простих числа – p довжиною 1024 біт і q довжиною 256 біт, між якими виконується співвідношення $p=bq+1$ для деякого цілого числа b.
2. Вибирається таке число a>1, що $a^q \bmod p=1$.
3. Публікуються у відкритому виді p, q і a.
4. Кожний користувач вибирає випадкове число x, $1<x<q$, і зберігає його в якості секретного ключа.
5. Обчислює число $y=a^x \bmod p$ і публікує його в якості відкритого ключа.

Генерація підпису виконується наступним способом:

1. Обчислюється значення геш-функції h=h(m), яке має задовольняти нерівності 1<h<p.
2. Вибирається випадкове число k (1<k< q).
3. Обчислюється число $r=(a^k \bmod p) \bmod q$; якщо r=0, повертаємося до кроку 2.
4. Обчислюється число $s=(kh+xr) \bmod q$; якщо s =0, повертаємося до кроку 2.
5. Формується підписане повідомлення <m, r, s>.

Перевірка підпису виконується наступним способом:

1. Обчислюється геш-функцію h=h(m).
2. Перевіряється виконання нерівності $0<r<q$, $0<s<q$.
3. Обчислюється $u_1=sh^{-1} \bmod q$, $u_2=-r\ h^{-1} \bmod q$
4. Обчислюється $v = (a^{u_1} y^{u_2} \bmod p) \bmod q$

5. Перевіряється виконання рівності v=r.

Якщо хоча б одна з цих перевірок не виконується, підпис вважається несправжнім.

Можна стверджувати, що якщо підпис був створений власником секретного ключа, то v=r. Дійсно,

$v = (a_{u1} y_{u2} \bmod p) \bmod q = (a_{sh-1} y_{-rh-1} \bmod p) \bmod q = (a_{(kh+xr)\, h-1} a_{-xrh-1} \bmod p) \bmod q = (a^k \bmod p) \bmod q = r.$

Для знаходження числа a рекомендується наступний метод. Спочатку вибирається випадкове число g>1 і обчислюється a= $g^{(p-1)/q}$ mod p. Якщо a>1, то воно нас влаштовує, оскільки a^q mod p= $g^{(p-1)}$ mod p=1. Якщо ж a=1, треба вибрати інше g.

Приклад. Виберемо такі спільні параметри: q=11, p=6q+1=67. Візьмемо g=10 і обрахуємо a=10^6 mod 67=25.

Виберемо секретний ключ x=6 і обрахуємо відкритий ключ y=25^6 mod 67=62.

Сформуємо підпис для повідомлення m. Нехай для нього геш-функція приймає значення h(m)=3. Візьмемо випадкове число k=8. Обрахуємо r=(25^8 mod 67) mod 11=24 mod 11=2, s=(8*3+6*2) mod 11=36 mod 11=3. Отримуємо підписане повідомлення <m,2,3>.

Для перевірки підпису знаходимо: $h^{-1}=3^{-1}$ mod 11=4, u_1=3*4 mod 11=1, u_2=-2*4 mod 11=-8 mod 11=3, v=(25^1 * 62^3 mod 67) mod 11=(25 * 9 mod 67) mod 11= 24 mod 11=2, тобто v=r.

Як бачимо, відмінності вітчизняного стандарту від американського полягають у такому:

1. Довжина q становить 160 біт.
2. В якості гещ-функції використовується алгоритм SHA-1.
3. При генерації підпису (крок 4) параметр s розраховується за формулою $s=k^{-1}(h+xr) \bmod q$.
4. При перевірці підпису (крок 3) параметри u_1 і u_2 розраховується за формулами $u_1=hs^{-1} \bmod q$, $u_1=rs^{-1} \bmod q$

9.6. Програмна реалізація цифрового підпису RSA засобами .NET

На платформі .NET цифровий підпис RSA реалізується за допомогою об'єктів класу **RSACryptoServiceProvider** з простору імен **System.Security.Cryptography**.

Порядок створення цифрового підпису полягає у такому:

1. Створюється контейнер з ключами за замовчуванням:

```
CspParameters signParam = new CspParameters(); signParam.KeyContainerName
= "Bob";
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(signParam);
//Для контролю можна виввести згенерований секретний ключ
//Console.WriteLine("---Secret key:\n{0}\n", rsa.ToXmlString(false));
```

2. Публічний ключ експортується для передачі іншій стороні в XML-форматі:

```
string pubKey = rsa.ToXmlString(false);
Console.WriteLine("---Public key:\n{0}\n", pubKey);
```

3. Вхідне повідомлення представляється у вигляді байтової послідовності:

```
byte[] message = Encoding.UTF8.GetBytes("Hello world!");
//Console.WriteLine("---HexMessage: \n{0}\n", BitConverter.ToString(message));
```

4. Створюється дайджест повідомлення за алгоритмом SHA-1:

```
SHA1 sha1 = new SHA1CryptoServiceProvider(); byte[] hmessage
= sha1.ComputeHash(message);
//Console.WriteLine("---Hesh: \n{0}\n", BitConverter.ToString(hmessage));
```

5. Створюється підпис для дайджесту, наприклад, в 16-му представленні:

```
byte[] signature = rsa.SignHash(hmessage, "sha1");
Console.WriteLine("---Signature: \n{0}\n", BitConverter.ToString(signature));
```

6. Підпис додається до повідомлення.

Порядок перевірки цифрового підпису полягає у такому:

1. Вхідне повідомлення представляється у вигляді байтової послідовності:

```
byte[] message = Encoding.UTF8.GetBytes("Hello world!");
Console.WriteLine("---HexMessage: \n{0}\n ", BitConverter.ToString(message));
```

2. Створюється дайджест повідомлення за алгоритмом SHA-1:

```
SHA1 sha1 = new SHA1CryptoServiceProvider(); byte[] hmessage =
sha1.ComputeHash(message); Console.WriteLine("---Hesh: \n{0}\n",
BitConverter.ToString(hmessage));
```

3. Створюється об'єкт RSA, до якого експортується публічний ключ з контейнера:

```
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(); string
pubKey =
"<RSAKeyValue><Modulus>mJ32CjZjzD2Qw4oRc/3O4xyVBLLPHeXELhIa9kwPhPCERkhuvROalKgfod
SgOrVr2K9TgTAMw3Ua4Q9O/vWhJRzsQPcSEYc5wJuHU9QY0a9jPn7WJQY91uCwfA54GsVYTLo2VI60DEt
uES7Dhlj4VK5KemvudEkmHiY8/BfO0XM=</Modulus><Exponent>AQAB</Exponent><P>z9JDG1Avu9
q5LlBNG6NQcxdhZc+HnJsQahjmza/fZb6T/K0tThAjAM18dH2gCCac05oJn2QEyCbtW5RgQ3lgvQ==</P
><Q>u/9yCtIK1GjfS16K5pNkSqdVZr2QDHbjy0cHiO7LfKBtWugUyN3FCSSUBooF2DmNFvRKDm1mZ4iPP
7nMsqYV7w==</Q><DP>burHyjIX5+kq4J8XKGMXsvWNlCsZM+pG7nlv5eOyFbmLflZnUbynEeyaOgo6eV
8yYHVcIWfebXzpPfGJFzoW+Q==</DP><DQ>hZeph6zoyzZW7u0ZEW7NxwsP8flk4qadizdHUHQ4n7A05X
OkSXTmbm/SzK7KJnQHIbeo5IWzToFJIkS7BHxnew==</DQ><InverseQ>jbhzN1BGEp4rA8njVFoHv8nY
oywX0fG0tVoeMkuu+V5St81pD5oY5rz+OMksTxD+scpSF8DnhEewtBPMDOTJDg==</InverseQ><D>P2u
U7NWBT0RePgPIEO1t5c7g1h0AGKp8hbCcZ7Ff2ZyhOxuqeQQGfrQGwRc8pmjxsg/ZoZu4Ehk93DyiVSz5
k3AfGe7vW+Mwk6jM71deoSyBdKpP1S/cpssOJHY781Tcx8jqqmP/gN19jEUkU7mJcmv4ahkis79THvo/F
vAmkJE=</D></RSAKeyValue>"; rsa.FromXmlString(pubKey);
```

4. Вхідне повідомлення представляється у вигляді байтової послідовності:

```
byte[] message = Encoding.UTF8.GetBytes("Hello world!");
Console.WriteLine("---HexMessage: \n{0}\n ", BitConverter.ToString(message));
```

5. Перевіряється підпис для дайджесту:

```
string hexSign =
"4725D1135F715C2FCFA2C6E93C839B106F1F8B2481CD599A5062652C39D2B63675D5254377B7400A
F85E3338F8023AA53D59AEC0144815C76FD02E22C0F3D8B976CDCDCF1DE42BAF7D4314C3124B54E4B
17B114A6E226001913AE15939584001AA1E98FA81C1F435F0F272DFEED1C27476B6F2C3E5A7AF968C
AC149892B7A198"; byte[] signature = StringToByteArray(hexSign); bool match =
rsa.VerifyHash(hmessage, "sha1", signature); Console.WriteLine("Verdict:\n{0}\n",
match);
```

Тут перетворення 16-вого цифрового підпису в байтовий масив використовується функція:

```
public static byte[] StringToByteArray(string hex)
{ return Enumerable.Range(0, hex.Length)
   .Where(x => x % 2 == 0)
```

```
    .Select(x => Convert.ToByte(hex.Substring(x, 2), 16)) .ToArray();
}
```

Можна обійтись без перетворення 16-вого цифрового підпису в байтовий масив. Для цього цифровий підпис треба зберегти, наприклад, у форматі Base64: `byte[] signature = rsa.SignHash(hmessage, "sha1");`

```
Console.WriteLine("---Signature: \n{0}\n", Convert.ToBase64String(signature));
```

Тоді перевірити підпис можна так:

```
string base64Sign =
"RyXRE19xXC/PosbpPIObEG8fiySBzVmaUGJlLDnStjZ11SVDd7dACvheMzj4AjqlPVmuwBRIFcdv0C4i
wPPYuXbNzc8d5CuvfUMUwxJLVOSxexFKbiJgAZE64Vk5WEABqh6Y+oHB9DXw8nLf7tHCdHa28sPlp6+Wj
KwUmJK3oZg="; byte[] signature = Convert.FromBase64String(base64Sign);

Console.WriteLine("---Signature: \n{0}\n", Convert.ToBase64String(signature));
```

Лістинг програми зі створення і перевірки ЕЦП RSA наведений в додатку 3.

9.7. Програмна реалізація цифрового підпису DSA засобами .NET

На платформі .NET цифровий підпис DSA реалізується за допомогою об’єктів класу **DSACryptoServiceProvider** з простору імен **System.Security.Cryptography**.

Цей алгоритм підтримує ключі довжиною від 512 до 1024 біт з приростами по 64 біта.

Нажаль, **DSACryptoServiceProvider** не підтримує роботу з контейнером. Тому порядок накладання підпису DSA дещо відрізняється від порядка накладання підпису RSA.

Порядок створення цифрового підпису полягає у такому:

1. Створюється об’єкт DSA з ключами за замовчуванням:

```
DSACryptoServiceProvider dsa = new DSACryptoServiceProvider();
//Для контролю виводиться згенерований секретний ключ string prKey
= dsa.ToXmlString(true);
Console.WriteLine("---Secret key:\n{0}\n", dsa.ToXmlString(true));
```

2. Публічний ключ експортується для передачі іншій стороні в XML-форматі:

```
string pubKey = rsa.ToXmlString(false);
Console.WriteLine("---Public key:\n{0}\n", pubKey);
```

3. Вхідне повідомлення представляється у вигляді байтової послідовності:

```
byte[] message = Encoding.UTF8.GetBytes("Hello world!");
Console.WriteLine("---HexMessage: \n{0}\n", BitConverter.ToString(message));
```

4. Створюється дайджест повідомлення за алгоритмом SHA-1:

```
SHA1 sha1 = new SHA1CryptoServiceProvider(); byte[] hmessage
= sha1.ComputeHash(message);
//Console.WriteLine("---Hesh: \n{0}\n", BitConverter.ToString(hmessage));
```

5. Створюється підпис для дайджесту, наприклад, у форматі Base64:

```
byte[] signature = dsa.SignHash(hmessage, "sha1");
//Console.WriteLine("---Signature: \n{0}\n", Convert.ToBase64String(signature));
```

6. Підпис додається до повідомлення.

Порядок перевірки цифрового підпису полягає у такому:

1. Вхідне повідомлення представляється у вигляді байтової послідовності:

```
byte[] message = Encoding.UTF8.GetBytes("Hello world!");
Console.WriteLine("---HexMessage: \n{0}\n ", BitConverter.ToString(message));
```

2. Створюється дайджест повідомлення за алгоритмом SHA-1:

```
SHA1 sha1 = new SHA1CryptoServiceProvider(); byte[] hmessage =
sha1.ComputeHash(message); Console.WriteLine("---Hesh: \n{0}\n",
BitConverter.ToString(hmessage));
```

3. Створюється об’єкт DSA, до якого експортується публічний ключ з контейнера:

```
DSACryptoServiceProvider dsa = new DSACryptoServiceProvider();
string pubKey =
```

```
"<DSAKeyValue><P>jmQh1u62F7h3RMkRs+uhoGn4fQMhih8GyZPWmJJHRwcOVxmJC5d5cR0k7PFf0x0+
PvkHr+wFSTvB2ZregiasXJ8WnHUvEcFCHbJ+iRiKQD1Rd75HfN8o8ViqL36Ia6PM1ZwI2Fqyc0zJshoZq
g2CQX8cDwO7T/Ogea+S+5bnrrE=</P><Q>68r+49IOAxNZNJkG43A5I+Ma9hE=</Q><G>g9RWkaYfOs1t
OewwxFHZFgN9NFTDz6FW5UWN2bakbfyOOZ2xeveYQ87nJEg5nXv4ZSTo7mxP2AhuL8lFKKmGcPLDgPFwj
cWqTAeMTd3x6zKDb1Ev0N4VTzzHD0GNLfTfdtQpTYuzzifp+gh2rtkmqF29g8DqOgG1uI9pVYfvYF0=</
G><Y>Q6SrbyDo6/T0n8aZrWowe3YJKgEzEZhT7qAY8nZ2O7CvCIM7Y3M8/9DkddrAosk1X4pSye5bmBTE
RpDB1+I7TlAqS+sOeUx69UL6OJlRGTfdai51g9T/qy7nPqwgY/jBQx4UvPjLIsFaovCjjRDLTje9X8Q2i
+5c/OSdHN8/mfQ=</Y><J>mpgEUdSn4XqlWugO+TKaF4r7WFQkux4wRPRIJHoL/wBgjIY8oA9Ji8RgVOG
gb9TI61ll5BTM4mfR/ucCgMHtoKRbi8dOzOgoj6hdT1BWDg4vW23t6v8Up3/APn/tO+ZAOaL26rZuscya
J1Ow</J><Seed>U1+1KTe5EGYeuLYQojzYeoTfGnk=</Seed><PgenCounter>Ag==</PgenCounter>
< /DSAKeyValue>"; dsa.FromXmlString(pubKey);
```

4. Перевіряється підпис для дайджесту:

```
string base64Sign = "mxdOkPaZrDdDnl15sEL8GQim6B4/c2fbd2ksc4ByqxOVS7TA1ouXqg==";
byte[] signature = Convert.FromBase64String(base64Sign);
//Console.WriteLine("---Signature: \n{0}\n", Convert.ToBase64String(signature));
bool match = dsa.VerifyHash(hmessage, "sha1", signature);
Console.WriteLine("---Verdict:\n{0}\n", match);
```

Лістинг програми зі створення і перевірки ЕСП DSA наведений в додатку 4.

Запитання для самоконтролю

1. Що представляє собою цифровий підпис?
2. Які властивості інформації захищає цифровий підпис?
3. Як реалізується цифровий підпис на основі шифру RSA?
4. Як реалізується цифровий підпис на основі шифру Ель-Гамаля?
5. Як реалізується цифровий підпис за алгоритмом DSS?
6. Які класи криптопровайдерів .Netмістять реалізацію цифрового підпису на основі шифру RSA? Як ними користуватись?
7. Які класи криптопровайдерів .Netмістять реалізацію цифрового підпису на основі алгоритму DSA? Як ними користуватись?

Результати навчання

Знання основних алгоритмів цифрового підпису на основі шифрів RSA, Ель-Гамаля, стандартів DSS та ГОСТ Р34.10-94. Знання можливостей платформи.Net з програмної реалізації цифрових підписів і уміння їх застосування в задачах прикладного програмування.

Комп’ютерний практикум №1

Тема: Шифр Цезаря

Мета: Розробити криптосистему на основі шифру Цезаря

Базові відомості

Шифр Цезаря - один з найдавніших шифрів, названий на честь римського імператора Гая Юлія Цезаря, який використовував його для секретного листування. При шифруванні кожен символ замінюється іншим, віддаленим від нього в алфавіті на фіксоване число позицій.

Якщо зіставити кожному символу алфавіту його порядковий номер, то шифрування і розшифрування можна виразити формулами модульної арифметики:

$y=(x+k) \bmod n$ $x=(y+n-(k \bmod n)) \bmod n$, де x - символ відкритого тексту, y - символ шифрованого тексту, n - потужність алфавіту, а k - ключ.

З прикладами використання шифру Цезаря можна ознайомитись на чисельних сайтах відповідної тематики, наприклад:
https://ciox.ru/caesar-cipher http://questhint.ru/shifr-tsezarya/
http://hostciti.net/calc/it/cipher-ceaser.html

Хід виконання роботи

1. Розробіть інтерфейс криптографічної системи симетричного шифрування, передбачивши в ньому використання меню та/або панелі інструментів для виконання таких команд:
    a. створення, відкривання, збереження, друкування файлів,
    b. шифрування і розшифрування файлів українською та англійською мовами,
    c. виведення відомостей про розробника та
    d. виходу з системи.

2. Розробіть систему класів для реалізації симетричного шифрування методом Цезаря, передбачивши в них методи валідації ключа, валідації, шифрування і розшифрування даних.
3. Виконайте тестування роботи системи.

Додаткові завдання:

1. Доповніть розроблену систему модулем для атаки на шифр Цезаря методом «грубої сили» (перебору).
2. Розширте можливості системи, забезпечивши можливість шифрування даних в будь-якому форматі, а не тільки текстових.

Комп’ютерний практикум №2

Тема: Шифр Тритеміуса

Мета: Розробити криптосистему на основі шифру Тритеміуса

Базові відомості

Шифр Тритеміуса - вдосконалений шифр Цезаря, в якому кожен символ повідомлення зміщується на символ, який відстає від даного на деякий крок. Але крок зміщення робиться змінним, тобто залежним від будь-яких додаткових чинників. Наприклад, можна задати закон зміщення у вигляді лінійної функції позиції літери, що шифрується, або за допомогою використання гасла – текстового рядка, який багаторазово записується під текстом повідомленням.

Таким чином, шифрування і розшифрування для шифру Тритеміуса можна виразити наступними рівняннями:

$y=(x+k) \bmod n$ $x=(y+n-(k \bmod n)) \bmod n$, де x - символ відкритого тексту, y - символ шифрованого тексту, n - потужність алфавіту.

Крок зміщення k розраховується:

- за лінійним рівнянням $k = Ap + B$;
- за нелінійним рівнянням $k = A^2 + Bp + C$;
- за гаслом.

Тут p - позиція букви в повідомленні. Ключем шифрування виступають відповідно коефіцієнти вказаних рівнянь та гасло.

Хід виконання роботи

1. Модифікуйте інтерфейс криптографічної системи симетричного шифрування з лабораторної роботи №1, забезпечивши можливість використання в якості ключа:
    a. 2-вимірного вектору для зберігання коефіцієнтів лінійного рівняння шифрування,

b. З-вимірного вектору для зберігання коефіцієнтів лінійного рівняння шифрування, с. Текстового рядка (гасла).

2. Доповніть систему класів з лабораторної роботи №1 класами та методами, необхідними для реалізації симетричного шифрування методом Тритеміуса, передбачивши в них методи валідації ключа, валідації шифрування і розшифрування даних.
3. Виконайте тестування роботи системи.

Додаткове завдання :

Доповніть систему модулем активної атаки на шифр Тритеміуса, який би забезпечував знаходження ключа шифрування у випадку, коли зловмиснику вдалось отримати пару повідомлень «незашифроване – зашифроване».

Комп’ютерний практикум №3

Тема: Шифр гамування

Мета: Розробити криптосистему на основі шифру гамування

Базові відомості

Метод полягає в тому, що символи тексту, який шифрується, послідовно складаються з символами деякої спеціальної послідовності, яка називається *гамою*. Іноді такий метод представляють як накладення гами на вхідний текст, тому він отримав назву «гамування». При цьому символи вихідного тексту і гамми замінюються цифровими еквівалентами, які потім складаються по модулю n, де n - число символів в алфавіті, тобто шифрування і розшифрування для шифру гамування можна виразити наступними рівняннями:

$y=(x+g) \bmod n$ $x=(y+n-(g \bmod n)) \bmod n$, де x - символ відкритого тексту, y - символ шифрованого тексту, g – символ гами.

Найбільш часто на практиці зустрічається двійкове гамування. При цьому використовується двійковий алфавіт, а складання здійснюється за модулем два: $z = x + g \pmod 2 = x \text{ XOR } g$.

Операція складання по модулю два в алгебрі логіки називається також "виключне АБО" або поанглійськи XOR. Операція XOR дуже швидко виконується на комп'ютері (на відміну від багатьох інших арифметичних операцій), тому накладення гами навіть на дуже великий відкритий текст виконується практично миттєво.

Цю ж саму операцію використовують і для розшифрування.

При використанні методу гамування ключем є послідовність, з якою проводиться складання - гамма. Якщо гамма коротше, ніж повідомлення, призначене для шифрування, гамма повторюється необхідну кількість разів. Чим довше ключ, тим надійніше шифрування методом гамування.

Розрізняють два різновиди гамування - з кінцевою і нескінченною гамами. При хороших статистичних властивостях гами якість шифрування визначається тільки довжиною періоду гами. При цьому, якщо довжина періоду гами перевищує довжину шифротексту, то такий шифр є абсолютно стійким, тобто його не можна розкрити за допомогою статистичної обробки зашифрованого тексту. При шифруванні за допомогою ЕОМ послідовність гами може формуватися за допомогою генератора псевдовипадкових чисел (ПВЧ).

Хід виконання роботи

1. Адаптуйте інтерфейс криптографічної системи симетричного шифрування з лабораторної роботи №1 або №2 для реалізації шифрування методом гамування.
2. Доповніть систему класів з попередніх лабораторних робіт класами та методами, необхідними для:
    a. генерації гами, період якої перевищує довжину вхідного тексту;
    b. реалізації симетричного шифрування методом гамування.
3. Виконайте тестування роботи системи.

Додаткове завдання:

Модифікуйте розроблену систему, забезпечивши можливість шифрування і розшифрування за допомогою шифроблокноту, як це передбачено в шифрі Вернама.

Комп’ютерний практикум №4

Тема: Книжковий шифр

Мета: Розробити криптосистему на основі використання віршованого фрагменту в якості ключа шифрування

Базові відомості

Книжковий шифр - вид шифру, в якому кожен елемент відкритого тексту (кожна буква або слово) замінюється на покажчик (наприклад, номер сторінки, рядки і стовпці) аналогічного елемента в додатковому тексті-ключі.

Суть методу книжкового шифру - це вибір будь-якого тексту з книги, де номери слів починаються на певну букву або координати (рядок, номер в рядку) самих букв виступають в якості шифру вихідного повідомлення. При цьому однією вихідної букві може відповідати декілька символів

Відомо кілька різновидів книжкового шифру. Найбільш простим з них є *шифрування з використанням вірша (віршований шифр)*.

У віршованому шифрі ключем є заздалегідь обумовлений вірш, яке записується в прямокутник узгодженого розміру. Цей прямокутник є ключовою сторінкою книжкового шифру.

Алгоритм віршованого шифрування:

1. Вибрати вірш для використання в якості ключа шифрування.
2. Пронумерувати всі стовпчики і рядки вибраного ключа шифрування двозначними цифрами: CC SS відповідно.
3. Символу M вхідного повідомлення поставити у відповідність 4-значний код CC/SS такого ж вибраного випадково символу ключа шифрування. Тут чисельник кожного дробу - номер рядка, а знаменник - номер стовпчика.
4. Код CC/SS занести до шифрограми і додати кому.
5. Повторити п.п.3-4 для кожного символу повідомлення, що шифрується

Алгоритм розшифрування з використанням вірша:

1. Для елемента коду CC/SS криптограми визначити номер стовпчика CC і рядка SS зашифрованого символу.
2. Знайти в ключі шифрування символ, що знаходиться на перетині CC колонки і SS-рядка.
3. Записати знайдений символ в якості розшифрованого символу.
4. Повторити п.п.1-3 для кожного елементу коду, відокремленого за допомогою ком.

Так в одному з таємних листувань революціонерів, ключем шифру був вірш Н. А. Некрасова «Школьник»: «Ну, пошел же ради бога ...». Вірш вписувався в квадрат розміром 10 на 10, якщо в рядку було більше 10 букв, то зайві літери викидалися:

	1	2	3	4	5	6	7	8	9	10
1	Н	У	П	О	Ш	Е	Л	Ж	Е	Р
2	Н	Е	Б	О	Е	Л	Ь	Н	И	К
3	Н	Е	В	Е	С	Е	Л	А	Я	Д
4	Э	Й	С	А	Д	И	С	Ь	К	О
5	Н	О	Г	И	Б	О	С	Ы	Г	Р
6	И	Е	Д	В	А	П	Р	И	К	Р
7	Н	Е	С	Т	Ы	Д	И	С	Я	Ч
8	Э	Т	О	М	Н	О	Г	И	Х	С
9	В	И	Ж	У	Я	В	К	О	Т	О
10	Т	А	К	У	Ч	И	Т	Ь	С	Я

Так, слово «Сообщите» по такій таблиці можна було зашифрувати декількома способами: «4/3,

5/2, 8/6, 2/3, 1/5, 7/7, 10/1, 6 / 2 ... »або« 10/9, 1/4, 8/3, 5/5, 1/5, 8/8, 9/9, 6/2 ... »і т. д. Так як в таблиці відсутня літера «Щ», то замість неї використовується літера «Ш», але це ніяк не заважає розшифровці повідомлення.

Помітною перевагою книжкового шифру є відсутність проблем, пов'язаних з підготовкою і передачею секретного ключа, адже кодовий текст відразу існує

в кількох примірниках. Проте цей шифр нестійкий до частотних методів криптоаналізу.

Хід виконання роботи

1. Розробіть інтерфейс криптографічної системи для реалізації шифрування з використанням вірша.
2. Доповніть систему класів з попередніх лабораторних робіт класами та методами, необхідними для шифрування і розшифрування віршованим шифром.
3. Виконайте тестування роботи системи.

Додаткове завдання:

Ознайомтесь з іншими різновидами книжкового шифру (див., наприклад, Wikipedia) та надайте їх програмну реалізацію.

Комп’ютерний практикум №5

Тема: Шифр DES

Мета: Ознайомитись з використанням криптопровайдерів в прикладному програмуванні

Базові відомості

Алгоритм симетричного шифрування DES (Data Encryption Standard) – стандарт симетричного шифрування США, розроблений у 1977 році, який згодом набув міжнародного застосування. Зараз DES вважається ненадійним в основному через малу довжину ключа.

З метою забезпечення більш високого рівня криптостійкості був запропонований модифікований метод з послідовним трициклічним шифруванням за алгоритмом DES, який отримав назву 3-DES (TripleDES). Криптостійкість методу виявилась значно кращою (про реалізацію успішних атак на 3DES не відомо), проте швидкість шифрування значно зменшилась у порівнянні з DES (приблизно в 3 рази).

На сьогодні алгоритми DES та 3-DES поступово витісняються новітнім алгоритмом шифрування AES (Advanced Encryption Standard), який забезпечує як високий рівень криптостійкості, так і прийнятну швидкість шифрування.

Нові можливості для розробки криптографічних додатків надає бібліотека класів .NET Framework, яка включає класи криптопровайдерів для реалізації симетричного шифрування за чотирма алгоритмами: DES, TripleDES і AES, а також RC2 (який є попередником AES і залишений для забезпечення сумісності з попередніми версіями додатків). Реалізуються вони за допомогою об’єктів двох класів з простору імен **System.Security.Cryptography**:

- **CryptographicServiceProvider** – клас, що надає криптопровайдери для кожного з вказаних алгоритмів.
- **CryptoStream**– клас для роботи з криптографічним потоком.

Застосування цих основних об’єктів вимагає використання об’єкту **FileStream** з простору імен **System.IO**. Крім того, для бітового представлення

текстових даних необхідні об’єкти класів **UnicodeEncoding** (або **ASCIIEncoding**) з простору імен **System.Text**.

Порядок шифрування за їх допомогою такий:

1. Створюється потрібний крипто провайдер і задаються його ключ і вектор ініціалізації:

```
DESCryptoServiceProvider cryptic = new DESCryptoServiceProvider();
cryptic.Key =
ASCIIEncoding.ASCII.GetBytes("ABCDEFGH"); cryptic.IV =
ASCIIEncoding.ASCII.GetBytes("ABCDEFGH"); cryptic.Mode =
CipherMode.CBC;
```

2. Відкривається звичайний файловий потік для запису зашифрованих даних:

```
FileStream stream = new FileStream(@"d:\test.txt",
FileMode.OpenOrCreate,FileAccess.Write)
```

3. Відкритий файловий потік трансформується в крипто потік для запису:

```
CryptoStream crStream = new CryptoStream(fs,
cryptic.CreateEncryptor(),CryptoStreamMode.Write);
```

4. Дані для шифрування перетворюються у бітову послідовність і всі біти (від 0 до data.Length) записуються в крипто потік за допомогою методу Write ():

```
byte[] data = ASCIIEncoding.ASCII.GetBytes("Hello World!");
crStream.Write(data,0,data.Length);
```

5. Використані файловий і крипто – потоки закриваються:

```
crStream.Close();
fs.Close();
```

Порядок розшифрування полягає у такому:

1. Створюється потрібний крипто провайдер і задаються його ключ і вектор ініціалізації :

   DESCryptoServiceProvider cryptic = new DESCryptoServiceProvider(); cryptic.Key =

   ASCIIEncoding.ASCII.GetBytes("ABCDEFGH");

   cryptic.IV = ASCIIEncoding.ASCII.GetBytes("ABCDEFGH");

   cryptic.Mode = CipherMode.CBC;

2. Відкривається звичайний файловий потік для читання зашифрованих даних:

   FileStream stream = new FileStream(@"d:\test.txt", FileMode.Open,FileAccess.Read)

3. Відкритий файловий потік трансформується в криптопотік для читання:

   CryptoStream crStream = new CryptoStream(stream, cryptic.CreateDecryptor(),CryptoStreamMode.Read).

4. Дані з крипто потоку зчитуються за допомогою об’єкта StreamReader і присвоюються текстовій змінній:

   StreamReader reader = new StreamReader(crStream); string data = reader.ReadToEnd();

5. Значення текстової змінної виводиться на екран і зчитувач та потік закриваються:

   reader.Close(); stream.Close();

Хід виконання роботи

1. Розробіть інтерфейс криптографічної системи для шифрування за допомогою DES з використанням всіх можливих режимів.

2. Ознайомтесь з описом класів CryptographicServiceProvider і CryptoStream бібліотеки .NET Framework.
3. Реалізуйте шифрування DES, використовуючи класи .NET Framework.
4. Виконайте тестування роботи системи.

Додаткові завдання

1. Модифікуйте створений програмний код для здійснення шифрування за алгоритмом TripleDES (або AES).
2. Узагальніть розроблений програмний код, забезпечивши можливість динамічного вибору одного з трьох шифрів – DES, TripleDES, AES.

Комп’ютерний практикум №6

Тема: Шифрування з відкритим ключем на основі задачі рюкзака

Мета: Ознайомитись з принципами побудови асиметричних криптосистем

Базові відомості

Першим алгоритмом для узагальненого шифрування з відкритим ключем став алгоритм рюкзака, розроблений Ральфом Меркле і Мартіном Хеллманом. Алгоритм демонструє можливість застосування задачі рюкзака (NP-повної проблеми) в криптографії з відкритими ключами. Задачу рюкзака можна сформулювати так:

Нехай задано множину натуральних чисел A = (a_1,a_2 ,..., a_n) і натуральне число S. Потрібно встановити, чи існує такий набір чисел x_i з (0,1), i ≤ n, для якого $\Sigma a_i x_i = S$ (1 <= i <= n)?

Ідея побудови системи шифрування на основі проблеми рюкзака полягає у виділенні деякого підкласу задач про укладання рюкзака, що розв'язуються порівняно легко – задачі «суперзростаючого» рюкзака, і "маскування" задач цього класу за допомогою деякого перетворення параметрів під загальний випадок. Параметри підкласу визначають секретний ключ, а параметри модифікованої задачі - відкритий ключ.

Суперзростаюча послідовність B = (b_1,b_2 ,..., b_n) - це послідовність, в якій кожній член більше суми всіх попередніх членів, тобто $b_i > \sum b_j$, j<i. Наприклад, послідовність {1,3,6,13,27,52} є суперзростаючою, а {1,3,4,9, 15,25} - ні.

Розв’язання задачі рюкзака для суперзростаючої послідовності знайти легко, використовуючи такий алгоритм:

Введення: натуральне число n>1, натуральне число S, суперзростаюча послідовність натуральних чисел B = (b_1,b_2 ,..., b_n).

Виведення: набір чисел xi з (0,1), i ≤ n, для якого Σaixi = S (1 <= i <= n)

Крок 1. Покласти i=n.

Крок 2.Порівняти S з найбільшим числом послідовності bi: якщо S< bi, то xi=0, інакше xi=1.

Крок 3. Зменшити S на bi, якщо xi=1.

Крок 4. Покласти i=n-1.

Крок 5. Якщо i>1, перейти до кроку 2, в іншому випадку повернути набір чисел xi.

Не суперзростаючі, або нормальні, рюкзаки представляють собою важку NP-проблему - швидкого алгоритму для них не знайдено. Алгоритм Меркле-Хеллмана заснований на цій властивості.

В якості закритого ключа вибирається суперзростаюча послідовність B = (b_1,b_2 ,..., b_n) та натуральні числа m> Σb_i, і t≡1(mod m). За ними будується послідовність нормального рюкзака A = (a_1,a_2 ,..., a_n) за наступним алгоритмом:

Введення: натуральне число n>1, суперзростаюча послідовність натуральних чисел B=(b_1, b_2 ,..., b_n), натуральні числа m> Σb_i, і t≡1(mod m).

Виведення: A = (a_1,a_2 ,..., a_n).

Крок 1. Покласти i=1.

Крок 2. Знайти a_i=b_i*t mod m.

Крок 3. Покласти i=i+1.

Крок 4. Якщо i>n, повернути A, в іншому випадку перейти до кроку 2.

Відкритий ключ A=(a_1, a_2 ,..., a_n) використовується для шифрування за таким алгоритмом:

Введення: натуральне число n>1, послідовність натуральних чисел A=(a_1, a_2 ,..., a_n), вхідне повідомлення p.

Виведення: шифротекст C.

Крок 1. Представити p у вигляді бінарної послідовності.

Крок 2. Розбити отриману бінарну послідовність на n-розрядні блоки $p_i=p_{i1}p_{i2}\ldots p_{in}$.

Крок 3. Зашифрувати кожний блок за допомогою перетворення $C_i = \Sigma\, p_{ij} \cdot a_j$, j = 1 ... n.

Крок 4. Отримати шифротекст C= (C_1; C_2; …; C_i)

Зашифроване повідомлення може розшифрувати власник закритого ключа, скористувавшись наступним алгоритмом:

Введення: натуральне число n>1, суперзростаюча послідовність натуральних чисел B=(b_1, b_2 ,..., b_n), натуральні числа m> Σb_i, і t≡1(mod m), шифротекст C= (C_1; C_2; …; C_i).

Виведення: відкрите повідомлення p.

Крок 1. Знайти таке дійсне t^{-1}, що $tt^{-1} \equiv 1(\text{mod } m)$.

Крок 2. Для кожного блоку шифротексту обчислити $C_i' \equiv t^{-1}C_i \,(\text{mod } m)$.

В принципі рішення задачі рюкзака завжди може бути знайдено повним перебором підмножин A і перевіркою, яка з їх сум дорівнює S. Але при великих n доведеться перебрати 2^n варіантів. Навіть для n = 300 пошук серед 2^{300} підмножин не піддається обробці.

Хід виконання роботи

1. Відшукайте в Інтернет-ресурсах чисельний приклад з використання «рюкзачного» алгоритму (наприклад, в Вікіпедії) та опрацюйте його.
2. Розробіть інтерфейс криптографічної системи для шифрування з використанням задачі рюкзака, передбачивши окремий діалог для формування відкритого ключа.
3. Розробіть методи, які б забезпечували:
    a. Генерацію пари «відкритий –закритий» ключі.
    b. Шифрування з використанням відкритого ключа.
    c. Розшифрування з використанням закритого ключа. При цьому значення t^{-1} вважати відомим.

4. Перевірте правильність роботи системи на основі використання даних з чисельного прикладу.

Додаткові завдання:

1. Ознайомтесь з можливостями он-лайн калькулятора для знаходження взаємно обернених чисел, використайте його для t^{-1} за відомими t і перевірте правильність функціонування системи в загальному випадку.
2. Ознайомтесь з розширеним алгоритмом Евкліда для знаходження взаємно обернених чисел і модифікуйте створений програмний код, додавши метод з реалізацією цього алгоритму і використання його для знаходження t^{-1} за відомими t і m.

Комп'ютерний практикум №7

Тема: Шифрування з відкритим ключем на основі алгоритму RSA

Мета: Ознайомитись з використанням криптопровайдерів .Net для побудови асиметричної криптосистеми

Базові відомості

Шифр RSA отримав назву на честь його розробників Ріверса (Ron Rivers), Шаміра (Adi Shamir) і Адлемана (Leonard Adleman). В RSA системі використовуються наступні факти з теорії чисел:

1. Задача перевірки числа на простоту є порівняно простою.
2. Задача розкладання числа n=p*q, де p і q – прості числа, на множники є дуже складною задачею, якщо ми знаємо тільки n, а p і q – великі числа (задача факторизації).

Основні повідомлення між сторонами B і A в протоколі RSA представляються наступною діаграмою:

A↔B: N=PQ, P,Q -прості;

B: f=(P-1)(Q-1); d<f, взаємно просте з f ; cd mod f=1;

B→A: d;

A:m; A→B: $e=m^d \bmod N$

B: y; B→A: $m'=e^c \bmod N$;

Алгоритм гарантує, що m'=m. Пара чисел (c,N) є секретним, а (d,N) – публічним ключем сторони B.

На платформі .NET алгоритм RSA реалізується за допомогою об'єктів класу **RSACryptoServiceProvider** з простору імен **System.Security.Cryptography**. Генерація відкритого та закритого ключів здійснюється при створенні нового екземпляра класу. Після створення нового екземпляра класу можна отримати інформацію про ключ одним із двох способів:

1. Метод **ToXMLString** – повертає інформацію про ключ в форматі XML.

2. Метод **ExportParameters** – повертає структуру RSAParameters, що містить ключові відомості.

Обидва методи приймають як параметр логічне значення, яке показує: **false** – слід повертати відомості тільки про відкритий ключ; **true** – слід повертати відомості і про відкритий, і про закритий ключі.

Ініціалізація класу RSACryptoServiceProvider може бути здійснена також двома шляхами:

1. Метод **FromXmlString** – використовує дані ключа з рядка XML.
2. Метод **ImportParameters** – використовує дані структури RSAParameters.

Асиметричні закриті ключі ніколи не повинні зберігатися в роздрукованому вигляді або у вигляді простого тексту на локальному комп'ютері. Якщо необхідно зберігати закритий ключ, слід використовувати для цього *контейнер ключа*. Контейнер ключа представляє собою екземпляр

класу **CspParameters** (з простору імен **System.Security.Cryptography)**. В полі **CspParameters.KeyContainerName** задається ім’я контейнера.

Порядок розшифрування за допомогою об’єктів класу **RSACryptoServiceProvider** такий:

1. Створюється контейнер для збереження ключів:

```
CspParameters cp = new CspParameters(); cp.KeyContainerName
= "Key Name";
```

2. Створюється екземпляр криптопровайдера з розміщенням ключів у контейнері:

```
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(cp)
```

3. Публічний ключ експортується для передачі іншій стороні:

```
string pubKey = rsa.ToXmlString(false);
Console.WriteLine("Public Key: \n {0}", pubKey);
```

4. Після отримання байтових даних byte[] EncryptBytes, зашифрованих за допомогою публічного ключа, здійснюється їх розшифрування за допомогою закритого ключа:

```
byte[] DecryptBytes = rsa.Decrypt(EncryptBytes, false); string decryptStr
= Encoding.Unicode.GetString(DecryptBytes);
//string decryptStr =BitConverter.ToString(DecryptBytes);
Console.WriteLine("Decripted string: \n {0}", decryptStr);
```

Порядок шифрування полягає у такому:

1. Створюється екземпляр криптопровайдера :

```
RSACryptoServiceProvider rsa1 = new RSACryptoServiceProvider()
```

2. Імпортується публічний ключ: `rsa1.FromXmlString(pubKey);`

3. Текст повідомлення перетворюється у байтову послідовність і зашифровується публічним ключем:

```
string dataToEncrypt = "Data to encrypt"; byte[] byteToEncript =
Encoding.Unicode.GetBytes(dataToEncrypt); byte[] EncryptBytes =
rsa1.Encrypt(byteToEncript, false);
```

4. Зашифрована байтова послідовність відправляється стороні, яка має для розшифрування відповідний закритий ключ.

Хід виконання роботи

1. Відшукайте в Інтернет-ресурсах чисельний приклад з використання алгоритму RSA (наприклад, в Вікіпедії) та опрацюйте його.
2. Розробіть інтерфейс криптографічної системи для шифрування з використанням RSA, передбачивши окремий діалог для формування відкритого ключа.
3. Розробіть методи, які б забезпечували:
   a. Генерацію пари «відкритий –закритий» ключі.
   b. Шифрування з використанням відкритого ключа.

c. Розшифрування з використанням закритого ключа.

4. Перевірте правильність роботи системи на основі використання даних з чисельного прикладу.

Додаткові завдання:

1. Ознайомтесь з можливостями он-лайн калькулятора для розкладання числа на прості множники і скористайтесь ним для проведення атаки на шифр RSA.
2. Визначте область значень параметрів шифру RSA, за яких така атака є реальною.

Комп’ютерний практикум №8

Тема: Електронно-цифровий підпис на основі алгоритму RSA

Мета: Ознайомитись з використанням криптопровайдерів .Net для створення і перевірки цифрового підпису

Базові відомості

Електронно-цифровий підпис (ЕЦП) – вид електронного підпису, отриманого за результатом криптографічного перетворення набору електронних даних, який додається до цього набору або логічно з ним поєднується і дає змогу підтвердити його цілісність та ідентифікувати підписувача.

ЕЦП накладається за допомогою особистого ключа та перевіряється за допомогою відкритого ключа.

Одним із видів електронно-цифрового підпису є ЕЦП на основі криптографічної системи RSA. Розглянемо принципи його функціонування.

Схема створення цифрового підпису RSA полягає у такому:

1. З вхідного повідомлення створюється 160-бітовий дайджест повідомлення (геш) за допомогою алгоритму SHA-l.
2. Дайджест повідомлення шифрується за допомогою секретного ключа RSA, відомого тільки відправнику повідомлення. Результат цього шифрування і називають цифровим підписом.
3. Підписане повідомлення формується об'єднанням вихідного повідомлення, його цифрового підпису та відкритого ключа.

Схема перевірки цифрового підпису RSA полягає у такому:

1. Отримане повідомлення розбивається на три компоненти: на вихідне повідомлення, відкритий ключ і цифровий підпис.
2. Знову обчислюється геш повідомлення.
3. Якщо геш збігається з розшифрованим гешем, то підпису вважається перевіреним.

На платформі .NET цифровий підпис RSA реалізується за допомогою об’єктів класу **RSACryptoServiceProvider** з простору імен **System.Security.Cryptography**.

Порядок створення цифрового підпису полягає у такому:

1. Створюється контейнер з ключами за замовчуванням:

```
CspParameters signParam = new CspParameters(); signParam.KeyContainerName
= "Bob";
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(signParam);
//Для контролю можна виввести згенерований секретний ключ
//Console.WriteLine("---Secret key:\n{0}\n", rsa.ToXmlString(false));
```

2. Публічний ключ експортується для передачі іншій стороні в XML-форматі:

```
string pubKey = rsa.ToXmlString(false);
Console.WriteLine("---Public key:\n{0}\n", pubKey);
```

3. Вхідне повідомлення представляється у вигляді байтової послідовності:

```
byte[] message = Encoding.UTF8.GetBytes("Hello world!");
//Console.WriteLine("---HexMessage: \n{0}\n", BitConverter.ToString(message));
```

4. Створюється дайджест повідомлення за алгоритмом SHA-1:

```
SHA1 sha1 = new SHA1CryptoServiceProvider(); byte[] hmessage
= sha1.ComputeHash(message);
//Console.WriteLine("---Hesh: \n{0}\n", BitConverter.ToString(hmessage));
```

5. Створюється підпис для дайджесту, наприклад, в 16-му представленні:

```
byte[] signature = rsa.SignHash(hmessage, "sha1");
Console.WriteLine("---Signature: \n{0}\n", BitConverter.ToString(signature));
```

6. Підпис додається до повідомлення.

Порядок перевірки цифрового підпису полягає у такому:

1. Вхідне повідомлення представляється у вигляді байтової послідовності:

```
byte[] message = Encoding.UTF8.GetBytes("Hello world!");
Console.WriteLine("---HexMessage: \n{0}\n ", BitConverter.ToString(message));
```

2. Створюється дайджест повідомлення за алгоритмом SHA-1:

```
SHA1 sha1 = new SHA1CryptoServiceProvider(); byte[] hmessage = sha1.ComputeHash(message); Console.WriteLine("---Hesh: \n{0}\n", BitConverter.ToString(hmessage));
```

3. Створюється об’єкт RSA, до якого експортується публічний ключ з контейнера:

```
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(); string pubKey =
"<RSAKeyValue><Modulus>mJ32CjZjzD2Qw4oRc/3O4xyVBLLPHeXELhIa9kwPhPCERkhuvROalKgfodSgOrVr2K9TgTAMw3Ua4Q9O/vWhJRzsQPcSEYc5wJuHU9QY0a9jPn7WJQY91uCwfA54GsVYTLo2VI60DEtuES7Dhlj4VK5KemvudEkmHiY8/BfO0XM=</Modulus><Exponent>AQAB</Exponent><P>z9JDG1Avu9q5LlBNG6NQcxdhZc+HnJsQahjmza/fZb6T/K0tThAjAM18dH2gCCac05oJn2QEyCbtW5RgQ3lgvQ==</P><Q>u/9yCtIK1GjfS16K5pNkSqdVZr2QDHbjy0cHiO7LfKBtWugUyN3FCSSUBooF2DmNFvRKDm1mZ4iPP7nMsqYV7w==</Q><DP>burHyjIX5+kq4J8XKGMXsvWNlCsZM+pG7nlv5eOyFbmLflZnUbynEeyaOgo6eV8yYHVcIWfebXzpPfGJFzoW+Q==</DP><DQ>hZeph6zoyzZW7u0ZEW7NxwsP8flk4qadizdHUHQ4n7A05XOkSXTmbm/SzK7KJnQHIbeo5IWzToFJIkS7BHxnew==</DQ><InverseQ>jbhzN1BGEp4rA8njVFoHv8nYoywX0fG0tVoeMkuu+V5St81pD5oY5rz+OMksTxD+scpSF8DnhEewtBPMDOTJDg==</InverseQ><D>P2uU7NWBT0RePgPIEO1t5c7g1h0AGKp8hbCcZ7Ff2ZyhOxuqeQQGfrQGwRc8pmjxsg/ZoZu4Ehk93DyiVSz5k3AfGe7vW+Mwk6jM71deoSyBdKpP1S/cpssOJHY781Tcx8jqqmP/gN19jEUkU7mJcmv4ahkis79THvo/FvAmkJE=</D></RSAKeyValue>"; rsa.FromXmlString(pubKey);
```

4. Вхідне повідомлення представляється у вигляді байтової послідовності:

```
byte[] message = Encoding.UTF8.GetBytes("Hello world!"); Console.WriteLine("---HexMessage: \n{0}\n ", BitConverter.ToString(message));
```

5. Перевіряється підпис для дайджесту:

```
string hexSign =
"4725D1135F715C2FCFA2C6E93C839B106F1F8B2481CD599A5062652C39D2B63675D5254377B7400AF85E3338F8023AA53D59AEC0144815C76FD02E22C0F3D8B976CDCDCF1DE42BAF7D4314C3124B54E4B17B114A6E226001913AE15939584001AA1E98FA81C1F435F0F272DFEED1C27476B6F2C3E5A7AF968CAC149892B7A198"; byte[] signature = StringToByteArray(hexSign); bool match = rsa.VerifyHash(hmessage, "sha1", signature); Console.WriteLine("Verdict:\n{0}\n", match);
```

Тут перетворення 16-вого цифрового підпису в байтовий масив використовується функція:

```
public static byte[] StringToByteArray(string hex)
{ return Enumerable.Range(0, hex.Length)
    .Where(x => x % 2 == 0)
    .Select(x => Convert.ToByte(hex.Substring(x, 2), 16)) .ToArray();
}
```

Можна обійтись без перетворення 16-вого цифрового підпису в байтовий масив. Для цього цифровий підпис треба зберегти, наприклад, у форматі Base64: `byte[] signature = rsa.SignHash(hmessage, "sha1");`

```
Console.WriteLine("---Signature:   \n{0}\n",   Convert.ToBase64String(signature));
```

Тоді перевірити підпис можна так:

```
string base64Sign =
"RyXRE19xXC/PosbpPIObEG8fiySBzVmaUGJlLDnStjZ11SVDd7dACvheMzj4AjqlPVmuwBRIFcdv0C4i
wPPYuXbNzc8d5CuvfUMUwxJLVOSxexFKbiJgAZE64Vk5WEABqh6Y+oHB9DXw8nLf7tHCdHa28sPlp6+Wj
KwUmJK3oZg="; byte[] signature = Convert.FromBase64String(base64Sign);

Console.WriteLine("---Signature: \n{0}\n", Convert.ToBase64String(signature));
```

Хід виконання роботи

1. Розробіть інтерфейс системи формування і перевірки ЕЦП RSA, передбачивши окремий діалог для формування відкритого ключа.
2. Розробіть методи, які б забезпечували:
    a. Генерацію пари «відкритий –закритий» ключі.
    b. Підписування довільного повідомлення з використанням закритого ключа.
    c. Перевірки ЕЦП з використанням відкритого ключа.
3. Перевірте правильність роботи системи на основі використання даних з чисельного прикладу.

Додаткові завдання:

1. Ознайомтесь з алгоритмом цифрового підпису DSA (Digital Signature Algorithm) та описом криптопровайдера .NET з реалізацією цього алгоритму **DSACryptoServiceProvider** з простору імен **System.Security.Cryptography**.
2. Побудуйте систему формування і перевірки ЕЦП за алгоритмом DSA.

Додаток 1. Лістинг програми, що виконує шифрування і розшифрування за алгоритмом DES.

```
using System;
using System.Security.Cryptography;
using System.IO; using System.Text;


   namespace DESExmp
{
       class Program
  {
       static   void   Main()
{
      byte[] keyArr= {1,2,3,4,5,6,7,8};//64-битный ключ
      byte[] ivArr = { 11, 22, 33, 44, 55, 66, 77, 88 };//64-битная синхропосылка       byte[]
date = UnicodeEncoding.UTF8.GetBytes("Hello world!");//Исходный текст
      //Создание криптопровайдера
      DESCryptoServiceProvider   des   =   new   DESCryptoServiceProvider();
des.Key = keyArr;       des.IV = ivArr;
      /***********Шифрование*********************************/
      //Открытие файлового потока
         FileStream fsIn = new FileStream(@"d:\test.txt", FileMode.OpenOrCreate,
   FileAccess.Write);
      //Трансформация в криптопоток
      CryptoStream   csIn   =   new   CryptoStream(fsIn,   des.CreateEncryptor(),
CryptoStreamMode.Write);       csIn.Write(date, 0, date.Length);
      //Результат в 16-ричных кодах
      Console.WriteLine(BitConverter.ToString(date));
csIn.Close();       fsIn.Close();
      /***********Расшифрование*********************************/
      //Открытие файлового потока
         FileStream    fsOut    =    new    FileStream(@"d:\test.txt",    FileMode.Open,
FileAccess.Read);
      //Трансформация в криптопоток
      CryptoStream    csOut    =    new    CryptoStream(fsOut,    des.CreateDecryptor(),
CryptoStreamMode.Read);
      StreamReader strRd = new StreamReader(csOut);       string
datd = strRd.ReadToEnd();
      //Результат
      Console.WriteLine(datd);
strRd.Close();             csOut.Close();
fsOut.Close();
    }
  }
}
```

Додаток 2. Лістінг програми узгодження спільного секрету по протоколу ДіффіХеллмана.

```
using System;
using System.Security.Cryptography; namespace
DiffHell
{
  class Program
  {
    static CngKey aliceCngKey;     static
CngKey  bobCngKey;          static  byte[]
alicePublicKeyBlob;         static  byte[]
bobPublicKeyBlob;


    //------------------------------------------------------------------------//    static
void PrintByteArray(byte[] arr)
    {
      for (int i = 0; i < arr.Length; i++)
      {
        Console.Write(arr[i]+"; ");
      }
    }
    //------------------------------------------------------------------------//    static
void GetAliceKey()
    {
      aliceCngKey = CngKey.Create(CngAlgorithm.ECDiffieHellmanP256);       bobCngKey
=  CngKey.Create(CngAlgorithm.ECDiffieHellmanP256);             alicePublicKeyBlob  =
aliceCngKey.Export(CngKeyBlobFormat.EccPublicBlob);              bobPublicKeyBlob  =
bobCngKey.Export(CngKeyBlobFormat.EccPublicBlob);
      CngKey          bobPubCngKey          =          CngKey.Import(bobPublicKeyBlob,
CngKeyBlobFormat.EccPublicBlob);


        ECDiffieHellmanCng  aliceAlgorithm  =  new  ECDiffieHellmanCng(aliceCngKey);
byte[] aliceKey = aliceAlgorithm.DeriveKeyMaterial(bobPubCngKey);
        Console.WriteLine("\n--->Alice sekret key:");
      PrintByteArray(aliceKey);
    }
    //------------------------------------------------------------------------//    static
void GetBobKey()
    {
      CngKey         alicePubCngKey         =         CngKey.Import(alicePublicKeyBlob,
CngKeyBlobFormat.EccPublicBlob);
      ECDiffieHellmanCng   bobAlgorithm   =   new   ECDiffieHellmanCng(bobCngKey);
byte[] bobKey = bobAlgorithm.DeriveKeyMaterial(alicePubCngKey);
        Console.WriteLine("\n--->Bob sekret key:");
      PrintByteArray(bobKey);
    }
////////////////////////////////////////////////////////////////////////////
static void Main()
    {
      GetAliceKey();
      GetBobKey();
    }
  }
```

Додаток 3. Лістинг програми зі створення і перевірки ЕЦП RSA.

```
   using      System;
using System.Text;
   using  System.Security.Cryptography;  using
System.Linq;

   namespace RSA_sign
   {
     class Program
     {

    //Функція створення підпису     static void
GetSign()
       {
       Console.WriteLine("\n<<<GetSign>>>:\n");

       //Створюється контейнер з ключами за замовчуванням      CspParameters signParam
= new CspParameters();
          signParam.KeyContainerName = "Bob";
          RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(signParam);
       //Для контролю виводиться згенерований секретний ключ
          Console.WriteLine("---Secret key:\n{0}\n", rsa.ToXmlString(false));
       //Публічний ключ експортується для передачі іншій стороні       string pubKey
= rsa.ToXmlString(false);
       Console.WriteLine("---Public key:\n{0}\n", pubKey);

       //Вхідне повідомлення представляється у вигляді байтової послідовності
          byte[] message = Encoding.UTF8.GetBytes("Hello world!");
          Console.WriteLine("---HexMessage: \n{0}\n", BitConverter.ToString(message));

       //Створюється дайджест повідомлення за алгоритмом SHA-1        SHA1
sha1  =  new  SHA1CryptoServiceProvider();             byte[]  hmessage  =
sha1.ComputeHash(message);
          Console.WriteLine("---Hesh: \n{0}\n", BitConverter.ToString(hmessage));
       //Створюється підпис для дайджесту
          byte[] signature = rsa.SignHash(hmessage, "sha1");
          Console.WriteLine("---Signature: \n{0}\n", Convert.ToBase64String(signature));
       }

    //Функція перевірки підпису     static void
VerifySign()
       {
       Console.WriteLine("\n<<<VerifySign>>>:\n");

       //Вхідне повідомлення представляється у вигляді байтової послідовності
          byte[] message = Encoding.UTF8.GetBytes("Hello world!");
          Console.WriteLine("---HexMessage: \n{0}\n ", BitConverter.ToString(message));

       //Створюється дайджест повідомлення за алгоритмом SHA-1
          SHA1 sha1 = new SHA1CryptoServiceProvider();
          byte[] hmessage = sha1.ComputeHash(message);
          Console.WriteLine("---Hesh: \n{0}\n", BitConverter.ToString(hmessage));
       //Створюється     об’єкт     RSA     з     розміщенням     ключів     у     контейнері
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();       string pubKey =
```

```
   "<RSAKeyValue><Modulus>mJ32CjZjzD2Qw4oRc/3O4xyVBLLPHeXELhIa9kwPhPCERkhuvROalKgfodSgOr
                                                    V
 r2K9TgTAMw3Ua4Q9O/vWhJRzsQPcSEYc5wJuHU9QY0a9jPn7WJQY91uCwfA54GsVYTLo2VI60DEtuES7Dhlj4V
 K5KemvudEkmHiY8/BfO0XM=</Modulus><Exponent>AQAB</Exponent><P>z9JDG1Avu9q5LlBNG6NQcxdhZ
 c+HnJsQahjmza/fZb6T/K0tThAjAM18dH2gCCac05oJn2QEyCbtW5RgQ3lgvQ==</P><Q>u/9yCtIK1GjfS16K
5pNkSqdVZr2QDHbjy0cHiO7LfKBtWugUyN3FCSSUBooF2DmNFvRKDm1mZ4iPP7nMsqYV7w==</Q><DP>burHyj
IX5+kq4J8XKGMXsvWNlCsZM+pG7nlv5eOyFbmLflZnUbynEeyaOgo6eV8yYHVcIWfebXzpPfGJFzoW+Q==</DP
><DQ>hZeph6zoyzZW7u0ZEW7NxwsP8flk4qadizdHUHQ4n7A05XOkSXTmbm/SzK7KJnQHIbeo5IWzToFJIkS7B
Hxnew==</DQ><InverseQ>jbhzN1BGEp4rA8njVFoHv8nYoywX0fG0tVoeMkuu+V5St81pD5oY5rz+OMksTxD+
scpSF8DnhEewtBPMDOTJDg==</InverseQ><D>P2uU7NWBT0RePgPIEO1t5c7g1h0AGKp8hbCcZ7Ff2ZyhOxuq
eQQGfrQGwRc8pmjxsg/ZoZu4Ehk93DyiVSz5k3AfGe7vW+Mwk6jM71deoSyBdKpP1S/cpssOJHY781Tcx8jqqm
P/gN19jEUkU7mJcmv4ahkis79THvo/FvAmkJE=</D></RSAKeyValue>";
         rsa.FromXmlString(pubKey);

      //Перевіряється  підпис  для  дайджесту           string
base64Sign =
   "RyXRE19xXC/PosbpPIObEG8fiySBzVmaUGJlLDnStjZ11SVDd7dACvheMzj4AjqlPVmuwBRIFcdv0C4iwPPY
                                                    u
 XbNzc8d5CuvfUMUwxJLVOSxexFKbiJgAZE64Vk5WEABqh6Y+oHB9DXw8nLf7tHCdHa28sPlp6+WjKwUmJK3oZg
=";
         byte[] signature = Convert.FromBase64String(base64Sign);
         Console.WriteLine("---Signature: \n{0}\n", Convert.ToBase64String(signature));

         bool match = rsa.VerifyHash(hmessage, "sha1", signature);       Console.WriteLine("-
--Verdict:\n{0}\n", match);

      }

//*************************************************************************//
static void Main()
      {
        GetSign();
        VerifySign();
      }
    }
  }
```

Додаток 4. Лістинг програми зі створення і перевірки ЕЦП DSA.

```
    using System; using System.Text;

   using  System.Security.Cryptography;  using
System.Linq;

   namespace DSA_sign
   {
     class Program
     {
    //Функція створення підпису     static void
GetSign()
       {
       Console.WriteLine("\n<<<GetSign>>>:\n");

       //Створюється об’єкт DSA з ключами за замовчуванням
          DSACryptoServiceProvider dsa = new DSACryptoServiceProvider();
       //Для контролю виводиться згенерований секретний ключ
          string prKey = dsa.ToXmlString(true);
          Console.WriteLine("---Secret key:\n{0}\n", dsa.ToXmlString(true));
       //Публічний ключ експортується для передачі іншій стороні        string pubKey
= dsa.ToXmlString(false);
       Console.WriteLine("---Public key:\n{0}\n", pubKey);

       //Вхідне повідомлення представляється у вигляді байтової послідовності
          byte[] message = Encoding.UTF8.GetBytes("Hello world!");
          Console.WriteLine("---HexMessage: \n{0}\n", BitConverter.ToString(message));

       //Створюється дайджест повідомлення за алгоритмом SHA-1        SHA1
sha1  =  new  SHA1CryptoServiceProvider();             byte[]  hmessage  =
sha1.ComputeHash(message);
          Console.WriteLine("---Hesh: \n{0}\n", BitConverter.ToString(hmessage));
       //Створюється підпис для дайджесту
          byte[] signature = dsa.SignHash(hmessage, "sha1");
          Console.WriteLine("---Signature: \n{0}\n", Convert.ToBase64String(signature));
       }

    //Функція перевірки підпису     static void
VerifySign()
       {
       Console.WriteLine("\n<<<VerifySign>>>:\n");

       //Вхідне повідомлення представляється у вигляді байтової послідовності
          byte[] message = Encoding.UTF8.GetBytes("Hello world!");
          Console.WriteLine("---HexMessage: \n{0}\n ", BitConverter.ToString(message));

       //Створюється дайджест повідомлення за алгоритмом SHA-1
          SHA1 sha1 = new SHA1CryptoServiceProvider();
          byte[] hmessage = sha1.ComputeHash(message);
          Console.WriteLine("---Hesh: \n{0}\n", BitConverter.ToString(hmessage));
       //Створюється об’єкт DSA, до якого експортується публічний ключ з контейнера
          DSACryptoServiceProvider dsa = new DSACryptoServiceProvider();       string pubKey
=
"<DSAKeyValue><P>jmQh1u62F7h3RMkRs+uhoGn4fQMhih8GyZPWmJJHRwcOVxmJC5d5cR0k7PFf0x0+PvkHr
+wFSTvB2ZregiasXJ8WnHUvEcFCHbJ+iRiKQD1Rd75HfN8o8ViqL36Ia6PM1ZwI2Fqyc0zJshoZqg2CQX8cDwO
```

```
7T/Ogea+S+5bnrrE=</P><Q>68r+49IOAxNZNJkG43A5I+Ma9hE=</Q><G>g9RWkaYfOs1tOewwxFHZFgN9NFT
Dz6FW5UWN2bakbfyOOZ2xeveYQ87nJEg5nXv4ZSTo7mxP2AhuL8lFKKmGcPLDgPFwjcWqTAeMTd3x6zKDb1Ev0
N4VTzzHD0GNLfTfdtQpTYuzzifp+gh2rtkmqF29g8DqOgG1uI9pVYfvYF0=</G><Y>Q6SrbyDo6/T0n8aZrWow
e3YJKgEzEZhT7qAY8nZ2O7CvCIM7Y3M8/9DkddrAosk1X4pSye5bmBTERpDB1+I7TlAqS+sOeUx69UL6OJlRGT
fdai51g9T/qy7nPqwgY/jBQx4UvPjLIsFaovCjjRDLTje9X8Q2i+5c/OSdHN8/mfQ=</Y><J>mpgEUdSn4XqlW
ugO+TKaF4r7WFQkux4wRPRIJHoL/wBgjIY8oA9Ji8RgVOGgb9TI61l15BTM4mfR/ucCgMHtoKRbi8dOzOgoj6h
dT1BWDg4vW23t6v8Up3/APn/tO+ZAOaL26rZuscyaJ1Ow</J><Seed>U1+1KTe5EGYeuLYQojzYeoTfGnk=</S
eed><PgenCounter>Ag==</PgenCounter></DSAKeyValue>";       dsa.FromXmlString(pubKey);

      //Перевіряється  підпис  для  дайджесту           string
base64Sign =
"mxdOkPaZrDdDnl15sEL8GQim6B4/c2fbd2ksc4ByqxOVS7TA1ouXqg==";
         byte[] signature = Convert.FromBase64String(base64Sign);
      //Console.WriteLine("---Signature: \n{0}\n", Convert.ToBase64String(signature));
         bool match = dsa.VerifyHash(hmessage, "sha1", signature);       Console.WriteLine("-
--Verdict:\n{0}\n", match);

       }

//**************************************************************************//
static void Main()
       {
      //GetSign();
         VerifySign();
       }
    }
  }
```

Література

1. Венбо Мао. Современная криптография. Теория и практика. М: Вильямс, 2005. – 768 с.
2. Бернет С., Пэйн С., Криптография. Официальное руководство RSA Security. Изд. 2-е, стереотипное. – М.: ООО «Бином-Пресс», 2007. – 384 с.: ил.
3. Аграновский А.В., Хади Р.А. Практическая криптография: алгоритмы и их программирование. – М.: СОЛОН-Пресс, 2002. – 256 с.
4. Адаменко М.В. Основы классической криптологии: секреты шифров и кодов. – М.: ДМК Пресс, 2012. – 256 с.
5. Бабаш А. В. Криптография. – М.: СОЛОН-Пресс, 2007. – 511 с.
6. Баричев С. Г., Гончаров В. В., Серов Р. Е. Основы современной криптографии: Учебное пособие. М.: Горячая Линия - Телеком, 2002. – 175 с.
7. Ростовцев А.Г., Маховенко Е.Б. Теоретическая криптография. М.: Издательство: АНО НПО "Профессионал", 2005. – 480 с.
8. Алферов А.П., Зубов А.Ю., Кузьмин А.С., Черемушкин А.В. Основы криптографии. – М.: Гелиос АРВ, 2001. – 480 с.
9. Маховенко Е. Б. Теоретико-числовые методы в криптографии. М.: Гелиос АРВ, 2006.–320 с.
10. Мухачев В.А., Хорошко В.А. Методы практической крипто-графии. К.: ООО Полиграф-Консалдинг, 2005. – 209 с.
11. Рябко Б.Я., Фионов А.Н. Криптографические методы защиты информации. М.: Горячая линия – Телеком, 2005. – 229 с.
12. Ростовцев А.Г. Алгебраические основы криптографии. М.: НПО «Мир и семья», ООО «Интерлайн», 2000. – 256 с.