СИМЕТРИЧНІ КРИПТОСИСТЕМИ

Функції криптосистем

Криптоалгоритми є методами перетворення невеликого блоку даних (від 4 до 32 байт) в закодований вигляд залежно від заданого двійкового ключа, що є основою криптографічних систем, але їх безпосереднє використання без яких-небудь модифікацій для кодування великих об'ємів даних не є оптимальним.

Всі недоліки безпосереднього використання криптоалгоритмів усуваються в криптосистемах. Криптосистема – це завершена комплексна модель, здатна здійснювати двосторонні криптоперетворення над даними довільного об'єму і підтверджувати час відправки повідомлення, яка володіє механізмом перетворення паролів і ключів, а також системою транспортного кодування. Таким чином, криптосистема виконує три основні функції:

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

Конкретна програмна реалізація криптосистеми називається криптопакетом.

Алгоритми створення ланцюжків

Перша задача, яка вирішується при шифруванні даних криптоалгоритмом, – це дані з довжиною, що не дорівнюють довжині 1 блоку криптоалгоритма. Ця ситуація матиме місце практично завжди.

Перша проблема: що можливо зробити, якщо необхідно зашифрувати 24 байти тексту, якщо використовується криптоалгоритм з довжиною блоку 8 байт? Можливо було б послідовно зашифрувати три рази по 8 байт і скласти їх у вихідний файл так, як вони містились в початковому файлі. Але якщо даних багато і деякі блоки по 8 байт повторюються, це означає, що у вихідному файлі ці ж блоки будуть зашифровані однаково - це призводить до зниження криптостійкості.

Друга проблема: якщо у файлі даних не 24, а 21 байт. Можливо було б не шифрувати останні 5 байт або чимось заповнювати ще 3 байти, а потім при дешифруванні їх викидати. Перший варіант взагалі неприємним, а другий застосовується, але виникає питання чим заповнювати останні байти?

Для вирішення цих проблем і були введені в криптосистеми алгоритми створення ланцюжків (англ. chaining modes). Найпростіший метод - метод ECB (Electronic Code Book), де шифрований файл тимчасово поділяється на блоки, рівні блокам алгоритму, кожний з них шифрується незалежно, а потім із зашифрованих пакетів даних компонується в тій же послідовності файл, який відтепер надійно захищений криптоалгоритмом. Назву алгоритм одержав через те, що він широко застосовувався в простих портативних пристроях для шифрування – електронних шифрокніжках. Схема цього методу наведена на рис.1.

У тому випадку, коли довжина пакету інформації, що пересилається, не кратна довжині блоку криптоалгоритму можливо розширення останнього (неповного) блоку байт до необхідної довжини або ж за допомогою генератора псевдовипадкових чисел, що не завжди безпечне відносно крипостійкості, або ж за допомогою хеш-суми інформації, що шифрується. Другий варіант є більш переважним, оскільки хеш-сума володіє

кращими статистичними показниками, а її апріорна відомість сторонній особі рівносильна знанню нею всієї інформації, що шифрується.

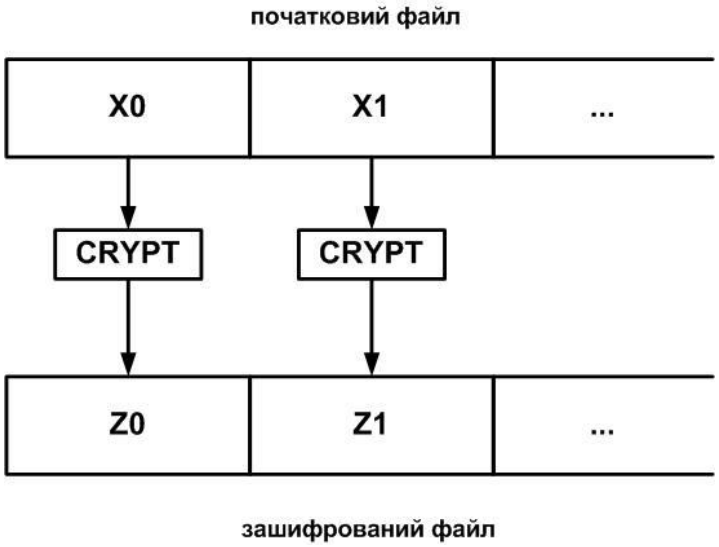


Рис.1. – Схема методу ECB

Вказаним вище недоліком цієї схеми є те, що при повторі в початковому тексті однакових символів протягом більш, ніж 2N байтів (де N – розмір блоку криптоалгоритма), у вихідному файлі присутні однакові зашифровані блоки. Тому, для підсилення захисту великих пакетів інформації за допомогою блокових шифрів застосовуються декілька зворотних схем "створення ланцюжків". Всі вони майже рівнозначні по крипостійкості, кожна має деякі переваги і недоліки, залежні від виду початкового тексту.

Всі схеми створення ланцюжків засновані на принципах залежності результуючого зашифровуваного блоку від попередніх блоків, або від позиції блоку в початковому файлі. Це досягається за допомогою блоку "пам'яті" – пакету інформації з довжиною, що дорівнює довжині блоку алгоритму. Блок пам'яті (до нього застосовують термін IV – англ. Initial Vector) обчислюється за певним принципом зі всіх минулих зашифрованих блоків, а потім накладається за допомогою якої-небудь зворотної функції (звичайно - XOR) на оброблюваний текст на одній із стадій шифрування. В процесі розкодування на приймальному боці операція створення IV повторюється на основі прийнятого і розшифрованого тексту, унаслідок чого алгоритми створення ланцюжків повністю зворотні.

Два найпоширеніших алгоритму створення ланцюжків – CBC і CFB. Їх структура наведена на рис.2 і рис.3. Метод CBC одержав назву від англійської абревіатури Cipher Block Chaining – об'єднання в ланцюжок блоків шифру, а метод CFB – від Cipher FeedBack – зворотний зв'язок за шифроблоком.

Метод OFB (англ. Output FeedBack – зворотний зв'язок за виходом) має дещо іншу структуру (рис.4.) : в ньому значення, що накладається на шифрований блок, не залежить від попередніх блоків, а тільки від позиції шифрованого блоку (в цьому значенні він повністю відповідає скремблерам), і через це він не поширює перешкоди на подальші блоки. Очевидно, що всі алгоритми створення ланцюжків є однозначно відновлюваними.

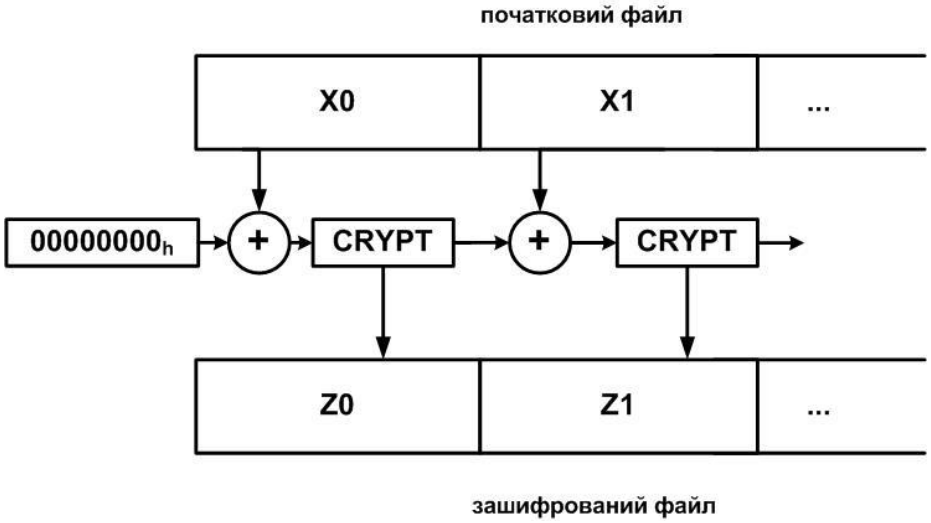


Рис.2. – Схема методу CBC

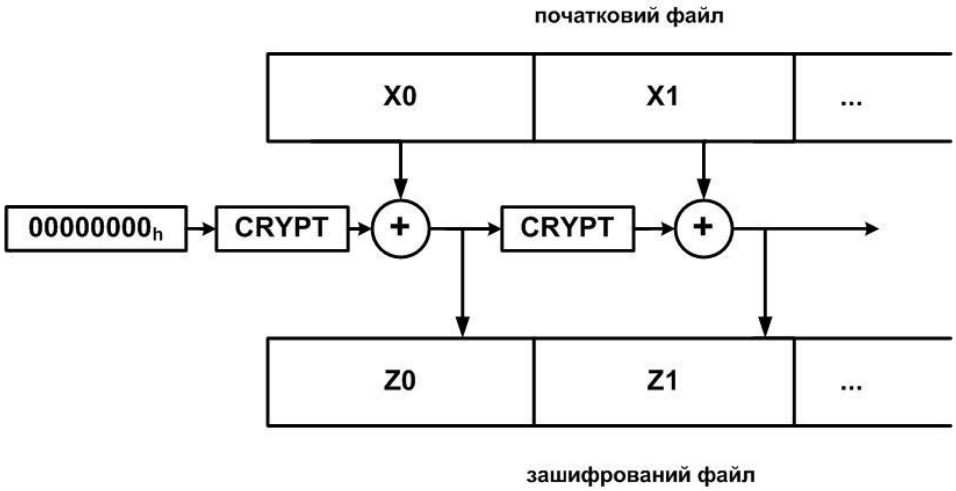


Рис.3. – Схема методу CFB

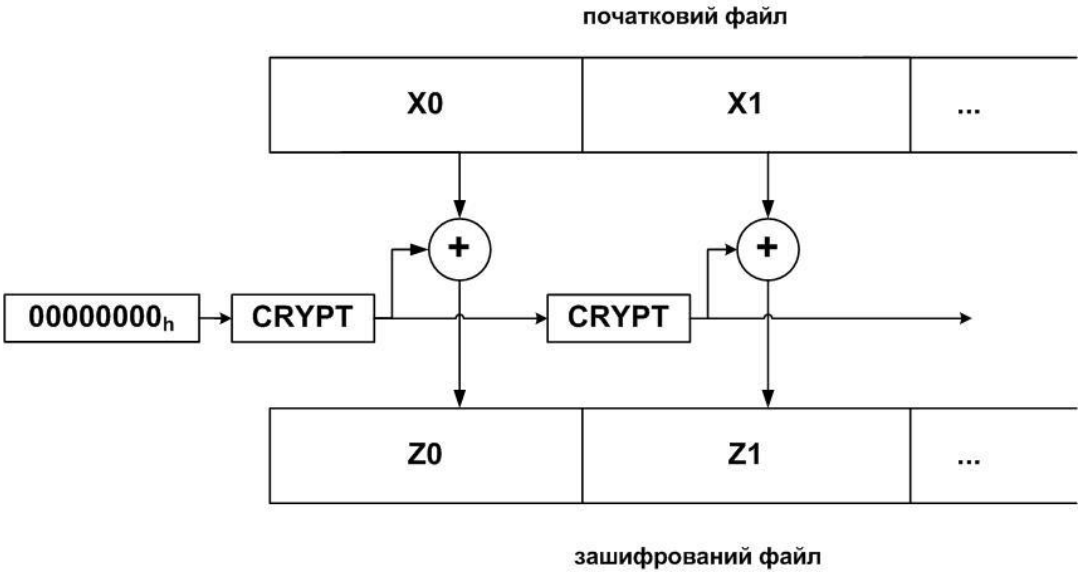


Рис.4. – Схема методу OFB

Порівняльна характеристика методів створення ланцюжків наведена у табл. 1.

Таблиця 1 - Порівняльна характеристика методів створення ланцюжків

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

Методи рандомізації повідомлень

Огляд методів рандомізації повідомлень

Ще одним вдосконаленням, направленим на підвищення стійкості всієї системи в цілому є створення ключів сеансу. Ця операція необхідна в тих випадках, коли здійснюється часте шифрування схожих блоків даних одним і тим же ключем. Наприклад, це має місце при передачі інформації або команд в автоматизованих системах управління, в банківських операціях і багатьох інших випадках передачі інформації, що має певний наперед відомий формат.

У цьому випадку необхідне застосування якої-небудь випадкової величини до процесу шифрування. Це можна зробити декількома способами:

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

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

Генератори випадкових і псевдовипадкових послідовностей

Найбільша проблема всіх методів рандомізації повідомлень – це створення дійсно випадкової послідовності біт. Річ у тому, що генератори випадкових послідовностей, що використовуються для загальних цілей, наприклад, в мовах програмування, є насправді псевдовипадковими генераторами. Це пояснюється тим, що у принципі існує кінцева, а не нескінченна множина станів ЕОМ, і, як би складно не формувалося в алгоритмі число, воно все одно має відносно небагато біт інформаційної насиченості.

Найчастіше в прикладних задачах випадковий (псевдовипадковий) результат формують з лічильника тиків – системного годинника. В цьому випадку дані про поточну годину несуть приблизно 32 біт інформації, значення лічильника тиків – ще 32 біт. Це дає нам 64 біти інформації. На сьогоднішній день межею стійкої криптографії є значення в 40 біт, при реальних довжинах ключів в 256 біт. Т.ч., подібний метод неоптимальним. До 64 біт можна додати ще 32 біт з надшвидкого таймера і цього ще недостатньо. Крім того, навіть якщо і можливо набрати довжину ключа в 256 біт (що дуже сумнівне), вона нестиме псевдовипадковий характер, оскільки заснована на стані тільки такої ЕОМ на момент початку шифрування. Джерелами по-справжньому випадкових величин можуть бути тільки зовнішні об'єкти, наприклад, людина.

Два найбільш поширених методів створення випадкових послідовностей за допомогою людини засновані на введенні з клавіатури. В обох випадках користувача просять, не замислюючись, понабирати на клавіатурі безглузді поєднання букв.

За першим методом над самими введеними значеннями проводяться дії, що підвищують випадковість вихідного потоку. Так, наприклад, обов'язково видаляються перші 3 біти введеного ASCII символу, часто віддаляються ще один перший і ще один останній біти. Потім, об'єм одержаної послідовності зменшується ще в три рази накладенням першого і другого біта, на третій, операцією XOR. Це генерує достатньо випадкову послідовність біт.

За другим методом на введені символи алгоритм не звертає ніякої уваги, зате запам’ятовує інтервали часу, через які відбулися натиснення. Запис моментів проводиться за відліками швидкого системного таймера або внутрішнього лічильнику процесора, що працює на частоті процесора. Оскільки перші і молодші біти мають певну кореляцію між символами (перші через фізичні характеристики людини, другі через особливості операційної системи), то вони відкидаються (звичайно віддаляються 0-8 старших біта і 4-10 молодших).

Як варіанти методів, що менш розповсюджені, можливо виділити 1) комбінацію обох клавіатурних методів і 2) метод, заснований на маніпуляторі "миша" - він виділяє випадкову інформацію із зсувів користувачем покажчика миші.

У потужних криптосистемах військового використання застосовуються дійсно випадкові генератори чисел, засновані на фізичних процесах. Вони є платою, або зовнішніми пристроями, що підключаються до ЕОМ через порт введення- виведення. При цьому двома основними джерелами білого Гауссовського шуму є високоточне вимірювання теплових флуктуацій і запис радіоефіру на частоті, вільній від радіомовлення.

Архівація

Загальні принципи архівації. Класифікація методів

Більшість сучасних форматів запису даних містять їх у вигляді, зручному для швидкого маніпулювання, для зручного зчитування користувачами. При цьому дані займають об'єм більший, ніж це дійсно потрібно для їх зберігання. Алгоритми, які усувають надмірність запису даних, називаються алгоритмами стиснення даних, або алгоритмами архівації. Сьогодні існує багато програм для стиснення даних, заснованих на декількох основних способах.

В сучасному криптоаналізі, тобто науці про протистояння криптографії, з доведено, що вірогідність злому криптосхеми за наявності кореляції між блоками вхідної інформації значно вища, ніж за відсутності такої. А алгоритми стиснення даних за визначенням і передбачають усунення надмірності, тобто кореляцій між даними у вхідному тексті.

Всі алгоритми стиснення даних якісно поділяються на :

1) алгоритми стиснення без втрат, коли дані відновлюються без щонайменших змін;

2) алгоритми стиснення з втратами, які видаляють з потоку даних інформацію, що незначно впливає на зміст даних, або взагалі на інформацію, що не сприймається людиною (такі алгоритми зараз розроблені тільки для аудіо- і відео- зображень).

В криптосистемах використовується тільки перша група алгоритмів.

Існує два основні методи архівації без втрат:

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

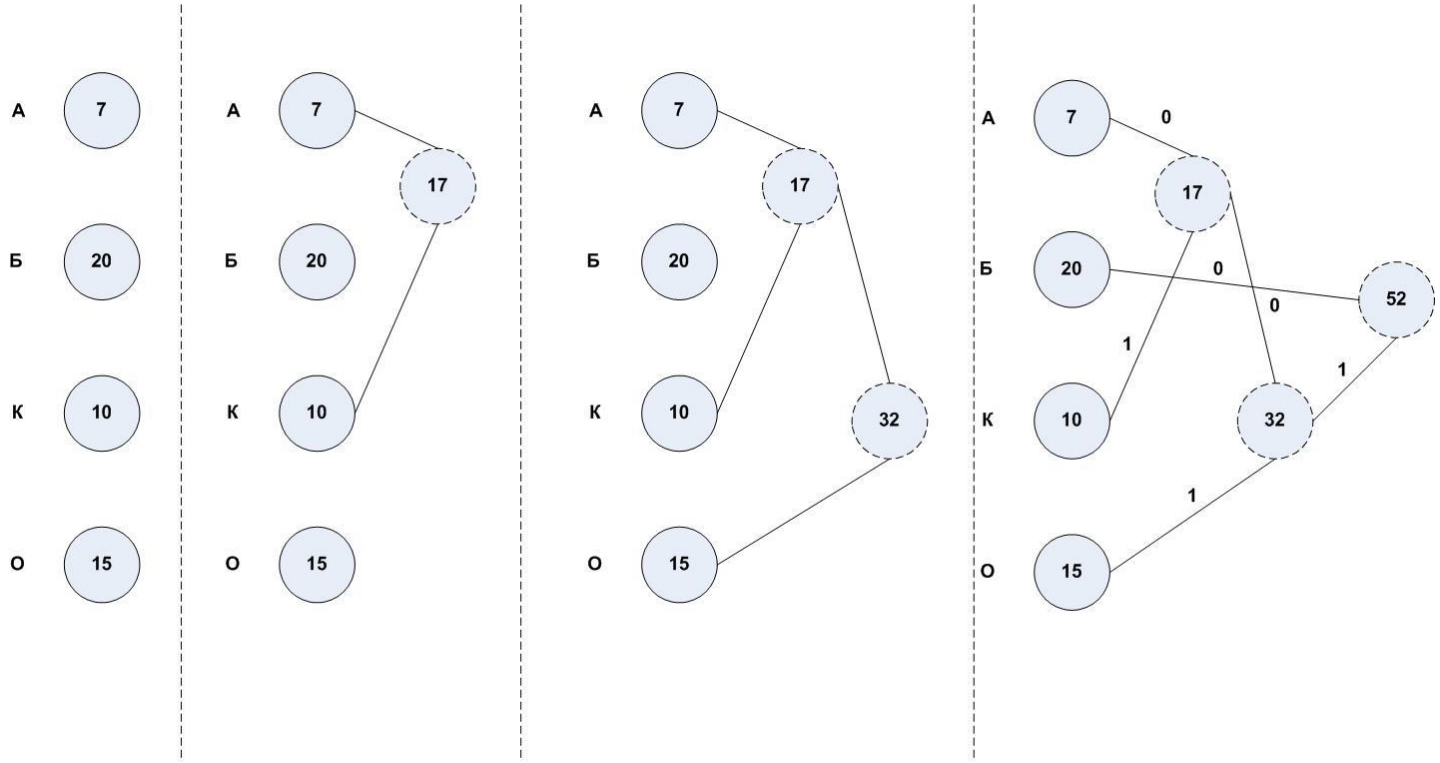
Практично всі популярні програми архівації без втрат (ARJ, RAR, ZIP і т. і.) використовують поєднання цих двох методів (напр., алгоритм LZH).

Алгоритм Хаффмана

Алгоритм засновано на тому, що деякі символи із стандартного 256-символьного набору в довільному тексті можуть зустрічатися частіше за середній період повторювання, а інші, відповідно, – рідше. Отже, якщо для запису поширених символів використовувати короткі послідовності біт, завдовжки менше 8, а для запису рідких символів – довгі, то сумарний об'єм файлу зменшиться.

Хаффманом було запропоновано простий алгоритм визначення того, який символ необхідно кодувати яким кодом для отримання файлу з довжиною, дуже близькою до його ентропії (тобто інформаційної насиченості). Наприклад, є перелік всіх символів, що зустрічаються в початковому тексті, причому відома кількість появи кожного символу в ньому. Перелічемо їх вертикально в стовпчик у вигляді осередків майбутнього графу за правою межою аркушу (рис. 5а). Виберемо два символи з якнайменшою кількістю повторень в тексті (якщо три або більша кількість символів мають однакові значення, обираємо будь-які два з них). Проведемо від них лінії вліво до нової вершини графа і позначимо в неї значення, що дорівнює сумі частот повторення кожного з об'єднуваних

символів (рис. 5б). Відтепер не братимемо до уваги при пошуку якнайменших частот повторення двох об'єднаних вузлів (для цього зітремо числа в цих двох вершинах), але розглядатимемо нову вершину як повноцінний осередок з частотою появи, що дорівнює сумі частот появи двох вершин, які було поєднано. Таку операцію поєднання вершин слід повторювати до тих пір, доки не залишиться одна вершина (рис. 2в, г), де буде записана довжина кодованого файлу. Тепер розставимо на двох ребрах графа, що надходять з кожної вершини, біти 0 і 1, таким чином, що ребро, яке пряму до вершини з більшою частотою має код 1, інше ребро – відповідно 0. Тепер для визначення коду кожної конкретної літери необхідно пройти шлях від вершини дерева до неї, виписуючи нулі і одиниці за маршрутом проходження. Для рис. 5 символ "А" одержує код "000", символ "Б" – код "01", символ "К" – код "001", а символ "О" – код "1".



а б в г

Рис.5. – Приклад отримання коду Хаффмана

У теорії кодування інформації доведено, що код Хаффмана є префіксним, тобто код жодного символу не є початком коду будь-якого іншого символу. З цього слід, що код Хаффмана є однозначно відновлюваним одержувачем, навіть якщо не зазначається довжина коду кожного переданого символу. Одержувачу пересилають тільки дерево Хаффмана в компактному вигляді, а потім вхідна послідовність кодів символів декодується їм самостійно без будь-якої додаткової інформації. Наприклад, при прийомі "0100010100001" їм спочатку відділяється перший символ "Б" : "01-00010100001", потім знову починаючи з вершини дерева – "А" "01-000-10100001", потім аналогічно декодується весь запис "01-000-1-01-000-01" "БАОБАБ".

Алгоритм Лемпеля-Зіва

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

Також використовується подібний до цього метод стиснення RLE (англ. Run Length Encoding), який полягає в запису замість послідовності однакових символів одного символу і їх кількості. Т.ч., метод RLE є підкласом алгоритму Лемпеля-Зіва. Наприклад, послідовність "АААААА" за допомогою алгоритму RLE буде закодована як "(А,7)", в той же час її можна достатньо добре стиснути і за допомогою алгоритму LZ77 : "А(-1,6)". При цьому ступінь стиснення такої послідовності алгоритмом LZ77 буде меншим (приблизно на 30-40%), але метод Лемпеля-Зіва є більш універсальним, і може краще обробляти послідовності, які взагалі нестискаються методом RLE.

–

Хешування паролів

Хешування паролів – дозволяє користувачам запам'ятовувати не 128 байт, тобто 256 шістнадцяткових цифр ключа, а деякий осмислений вираз, слово або послідовність символів, що називається паролем. При розробці будь-якого криптоалгоритму слід враховувати те, що часто кінцевим користувачем системи є людина, а не автоматична система. Це ставить питання про те чи, зручно, і чи взагалі реально людині запам'ятати 128-бітний ключ (32 шістнадцяткові цифри). Межа запам’ятовування сягає межі 8-12 подібних символів, а, отже, якщо користувач буде оперувати саме ключем, то він буде примушуватись до запису ключа на якому-небудь листку паперу або електронному носії, наприклад, в текстовому файлі. Це, звісно, різко знижує захищеність системи.

Для вирішення цієї проблеми були розроблені методи, що перетворюють осмислений рядок довільної довжини – пароль, у вказаний ключ наперед заданої довжини. В переважній більшості випадків для цієї операції використовуються так звані хеш-функції (від англ. hashing – дрібна нарізка і перемішування). Хеш-функцією називається таке математичне або алгоритмічне перетворення заданого блоку даних, яке володіє наступними властивостями:

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

Ці властивості дозволяють подавати на вхід хеш-функції паролі, тобто текстові рядки довільної довжини на будь-якій національній мові і, обмеживши область значень функції діапазоном 0…2N-1, де N – довжина ключа в бітах, отримати на виході достатньо рівномірно розподілені за областю значення блоки інформації – ключі.

Такі вимоги, подібні 3 і 4 пунктам вимог до хеш-функції, здійснюють блокові шифри. Це указує на один з можливих шляхів реалізації стійких хеш-функцій – проведення блокових криптоперетворень над матеріалом рядка-пароля. Цей метод і

використовується в різних варіаціях практично у всіх сучасних криптосистемах. Матеріал рядка-пароля багато разів послідовно використовується як ключ для шифрування деякого наперед відомого блоку даних – на виході виходить зашифрований блок інформації, однозначно залежний тільки від пароля і при цьому він має достатньо добрі статистичні характеристики. Такий блок або декілька таких блоків і використовуються як ключ для подальших криптоперетворень.

Характер використання блокового шифру для хешування визначається відношенням розміру блоку криптоалгоритма і розрядності необхідного хеш-результату, що застосовується.

Якщо вказані вище величини співпадають, то використовується схема одноланцюгового блокового шифрування. Первинне значення хеш-результату H0 встановлюється рівним 0, весь рядок-пароль поділяється на блоки байт, рівні за довжиною ключу блокового шифру, що використовується для хешування, потім виконується перетворення за рекурентною формулою:

$$Hj=Hj\text{-}1 \text{ XOR } EnCrypt(Hj\text{-}1,PSWj),$$

де EnCrypt(X,Key) – блоковий шифр, що використовується (рис. 6).

Останнє значення Hk використовується як результат хешування.

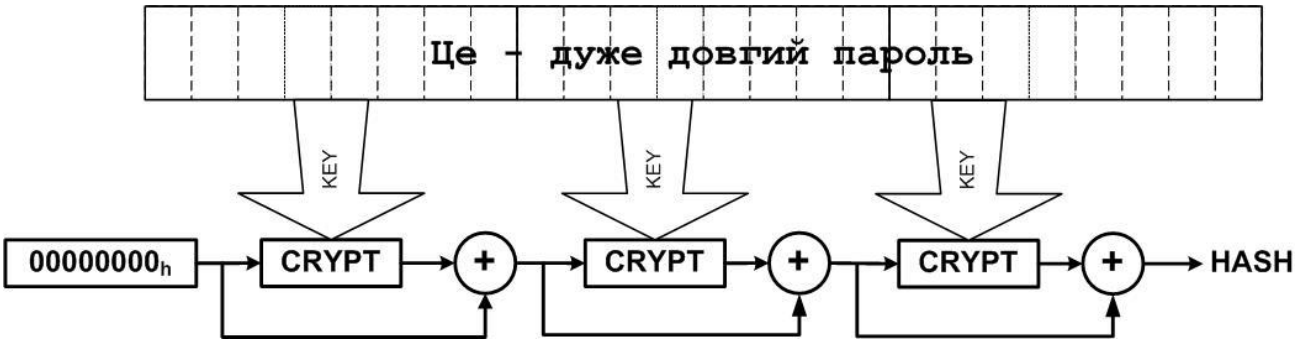


Рис.6. – Схема хешування на основі одноланцюгового блокового шифрування

У тому випадку, коли довжина ключа в два рази перевищує довжину блоку, а подібна залежність досить часто зустрічається в блокових шифрах, використовується схема, що нагадує мережу Фейштеля. Характерним недоліком і наведеної вище формули, і хеш-функції, заснованої на мережі Фейштеля, є велика ресурсомісткість відносно паролю. Для здійснення лише одного перетворення, наприклад, блоковим шифром з ключем завдовжки 128 біт використовується 16 байт рядка-пароля, а власне довжина пароля зрідка перевищує 32 символи. Отже, при обчисленні хеш-функції над паролем будуть проведено максимум 2 "повноцінних" криптоперетворення.

Рішенням цієї проблеми є :

1) використання заздалегідь помноженого рядку-паролю, наприклад, записавши його багато разів послідовно до досягнення довжини у 256 символів;

2) модифікувати схему застосовування криптоалгоритму так, щоб матеріал рядка-пароля "повільніше" витрачався при обчисленні ключа.

За другому методом Девіс і Майер запропонували на основі блокового шифру алгоритм, що використовує матеріал рядка-пароля багатократно і невеликими порціями. Йому притаманні елементи обох наведених вище схем, але крипостійкість цього

алгоритму підтверджена численними реалізаціями в різних криптосистемах. Алгоритм одержав назву "Tandem DM" (рис. 7):

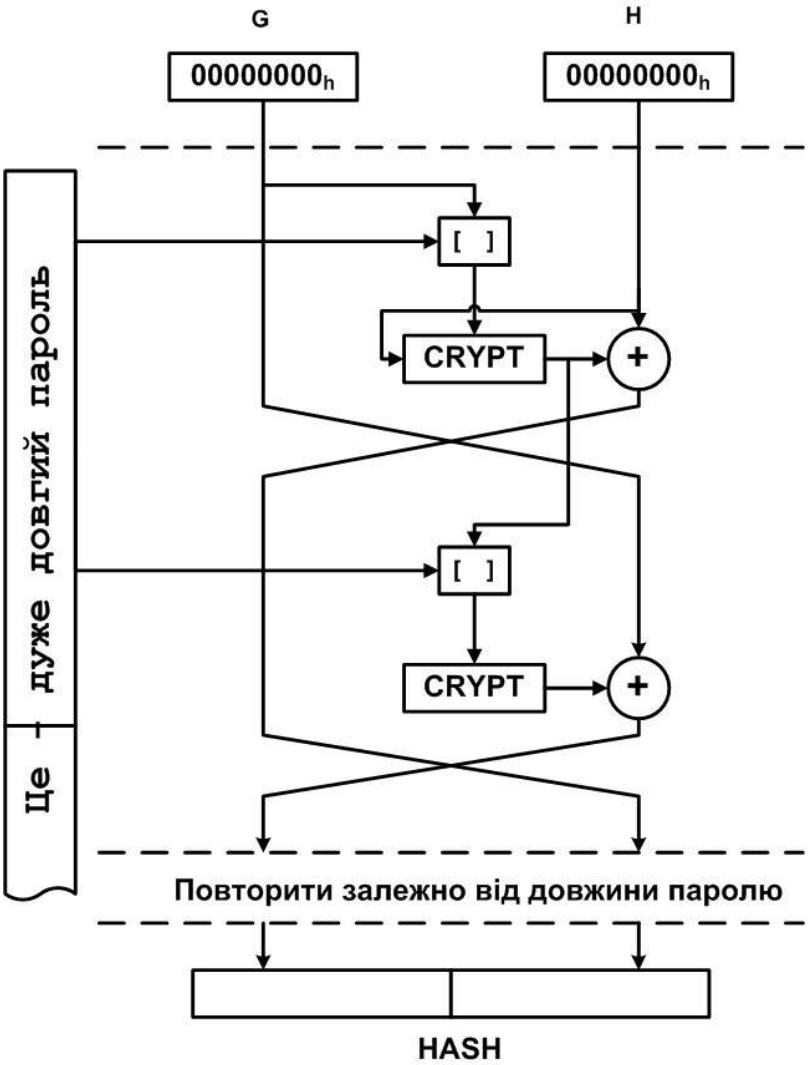


Рис. 7 – Схема алгоритму хешування Tandem DM

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

Квадратними дужками (X16=[A8,B8]) тут позначено просте поєднання (склеювання) двох блоків інформації рівної величини в один – подвоєної розрядності. А у якості процедури EnCrypt(X,Key) може бути обрано будь-який стійкий блоковий шифр. Як слід з формул, такий алгоритм орієнтовано на те, що довжина ключа двократно перевищує розмір блоку криптоалгоритму. Характерною особливістю схеми є те, що рядок пароля зчитується блоками по половині довжини ключа, і кожний блок використовується в створенні хеш-результату двічі. Таким чином, при довжині пароля в 20 символів і

необхідність створення 128 бітового ключа внутрішній цикл хеш-функції повториться 3 рази.

Транспортне кодування

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

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

Найпростіший метод полягає в записі кожного байту двома шістнадцятковими цифрами-символами. Так байт 252 буде записаний двома символами 'FC'; байт з кодом 26, якому відповідає спеціальний символ CTRL-Z, буде записаний двома припустимими символами '1A'. Така схема є дуже надмірною, так як у одному байті передається тільки 4 біти інформації.

Практично в будь-якій системі комунікації без проблем можливо передавати приблизно 68 символів (латинський алфавіт рядковий і прописний, цифри і розділові знаки). З цього слід, що цілком реально створити систему з передачею 6 біт в одному байті (26<68), тобто кодувати 3 байти довільного змісту чотирма байтами з виключно дозволених (так званих друкарських) символів. Подібна система була розроблена і стандартизована на рівні протоколів мережі Інтернет – це система Base64 (стандарт RFC1251), яку наведено на рис. 8.

Процес кодування перетворює 4 вхідні символи у вигляді 24-бітової групи, обробляючи їх зліва направо. Ці групи потім розглядаються як 4 сполучені шостибітові групи, кожна з яких транслюється в одиночну цифру алфавіту Base64. При кодуванні Base64 вхідний потік байтів повинен бути впорядкований старшими бітами спочатку.

Кожна шостибітова група використовується як індекс для масиву 64-х друкарських символів. Символ, на який вказує значення індексу, поміщається у вихідний рядок. Ці символи вибрані так, щоб бути універсально уявними і виключають символи, що мають спеціальне значення (".", CR, LF).

Вихідний потік (закодовані байти) повинен мати довжину рядків не більше 76 символів. Всі ознаки перекладу рядка і інші символи, відсутні на рис. 8, повинні бути проігноровані декодером base64. Серед даних в Base64 символи, не перераховані в табл. 1, закінчення рядку і т. і., повинні говорити про помилку передачі даних, і, відповідно, програма-декодер повинна сповістити користувача про неї.

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

Рис. 8 – Алфавіт Base64

Якщо у хвості потоку кодованих даних залишилося менше, ніж 24 біти, справа додаються нульові біти до утворення цілої кількості шостибітових груп. У випадках коли до кінця 24-бітової групи може залишатися тільки від 0 до 3-х не дістаючих шостибітових груп, замість кожної з яких ставиться символ-заповнювач "=". Оскільки весь вхідний потік є цілим числом восьмибітових груп (тобто, просто байтних значень), то можливі лише наступні випадки:

1. вхідний потік закінчується рівно 24-бітовою групою (довжина файлу кратна 3); у такому разі вихідний потік закінчуватиметься чотирма символами Base64 без будь-яких додаткових символів;
2. "хвіст" вхідного потоку має довжину 8 біт; тоді в кінці вихідного коду містяться два символи Base64, з додаванням двох символів "=";
3. "хвіст" вхідного потоку має довжину 16 біт; тоді в кінці вихідного коду містяться три символи Base64 і один символ "=".

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

Загальна схема симетричної криптосистеми

Узагальнена схема симетричної криптосистеми з урахуванням всіх розглянутих функцій зображена на рис. 9.

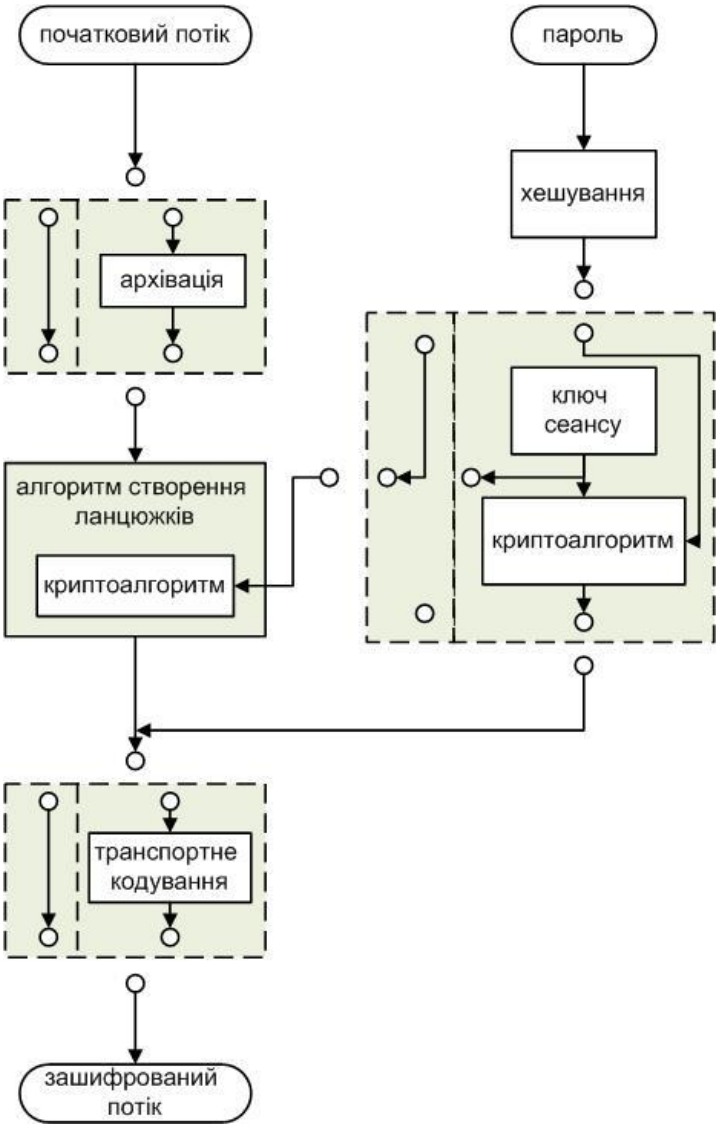


Рис.9. - Узагальнена схема симетричної криптосистеми