# **Модуль 1**

## **Знайомство з ІТ-безпекою**

### **Вступ до курсу**

Перш ніж перейти до наступного курсу, я хочу вас щиро привітати. Ви пройшли непростий шлях до завершального курсу цієї програми. Так, це було нелегко, але погляньте, якого прогресу ви досягли. Ви вивчили всі основи ІТ. Зараз ми на останньому курсі, який доповнить ваші базові знання. Ми поговоримо про безпеку. Без неї всі процеси, які ви вивчили до цього часу, можуть зазнати невдачі, а цього не хоче жоден ІТ-фахівець.

Перш ніж ми перейдемо до справи, ще раз представлюся. Ми зустрічалися ще на першому курсі, коли говорили про історію Інтернету й Інтернет речей. Мене звати Джіан Спікузза, і я керівник програм у відділі безпеки Android. Я допомагаю захищати понад 2 млрд пристроїв Android, випускаючи нові функції безпеки та керуючи ними для кожної версії Android (які мають назви десертів).

Скільки я себе пам’ятаю, я завжди любив техніку. Я працюю у сфері ІТ з 16 років, і на дозвіллі я любив читати книжки про нові технології й збирати сервери зі старих комп’ютерних деталей у підвалі батьківського будинку. Я ніколи не складав тести дуже добре, і доказом тому були мої оцінки. Але це не зупинило мене на шляху до успішної кар’єри. Я працював єдиним ІТ-спеціалістом на три некомерційні організації, поки здобував освіту. Це був ще той стрес – відповідати за все: від налаштування й адміністрування баз даних до навчання нових працівників роботі з електронною поштою та внутрішніми інструментами.

Озираючись назад, я розумію, що цей досвід був безцінним. І, звісно, важливою частиною моєї роботи у сфері ІТ була саме безпека. Тепер я працюю безпосередньо з виробниками обладнання, розробниками додатків і командами інженерів у Google, забезпечуючи найвищий рівень захисту для наших користувачів. Для багатьох із них мобільний телефон – це єдиний спосіб зв’язку з Інтернетом. І я відчуваю велике задоволення, знаючи, що моя робота суттєво впливає на людей в усьому світі, які так покладаються на свої пристрої. Щоб досягти успіху в кібербезпеці, іноді потрібно подумки поставити себе на місце нападника й завжди бути на крок попереду. Отже, ви готові робити добро, думаючи про погане? Тоді поїхали!

Цей модуль буде повністю присвячений питанням безпеки: як безпеку намагаються порушити та як ми захищаємося від цих атак. Наприкінці цього модуля ви зможете визначати й розпізнавати ризики, вразливості й загрози безпеці. Ви також зможете визначити найпоширеніші атаки на системи безпеки. Нарешті, ви зрозумієте, як безпека обертається навколо принципу КЦД, і що таке принцип КЦД.

Коли йдеться про безпеку, що найперше спадає вам на думку? Мабуть, це фізична безпека, щось на кшталт захисту майна від потенційних злодіїв, зачинення вхідних дверей уночі та зберігання цінних речей у безпечному місці. Але в сучасному цифровому світі гроші зберігаються не лише в гаманці. Кошти також зберігаються на банківських рахунках онлайн, до яких можна отримати доступ, знаючи пароль. Дехто з нас узагалі не носить із собою кредитних карток, а якщо й носить, то не в гаманці. Вони зберігаються на наших улюблених вебсайтах, щоб зручніше робити покупки.

Але ми турбуємося не лише через гроші. Більша частина нашого особистого світу зберігається на мобільних телефонах. SMS-повідомлення, фотографії, особисті дані, дані для входу в додатки й багато іншого – усе це зберігається всередині пристроїв, які ми носимо в кишенях. На жаль, ми живемо у світі, у якому деякі люди й організації намагаються викрасти дані в компаній, урядів і навіть простих людей, як ми з вами. Однак фільми дають спотворене уявлення про цих цифрових злодіїв. Вони не такі гламурні чи професійні, як вам може здаватися. Цифрові злодії не мають команди хакерів, не носять темні худі й не друкують несамовито на своїх терміналах цілими днями, сподіваючись зламати системи багатомільярдних компаній. Я не кажу, що такого не трапляється, – нам усім відомі такі випадки. Але зазвичай середньостатистичний інтернет-зловмисник має такий самий вигляд, як ми з вами, – це звичайна людина, яка випадково дізналася про діру у вашій системі безпеки, а потім нею скористалася.

Наприклад, злодій міг просто дізнатися, що ваш пароль – це кличка вашої собаки. Коли єдине, що захищає ваш банківський рахунок, – це слово Fido, у вас проблеми. Але так само, як фізичні пристрої сигналізації стримують потенційних грабіжників, існує також чимало способів запобігти порушенню нашої цифрової безпеки. У цьому курсі ви глибше зрозумієте комп’ютерну безпеку. Ви дізнаєтеся, як запобігти найчастішим комп’ютерним атакам. Ви зрозумієте різні протоколи й механізми безпеки, які використовуються на наших комп’ютерах, в Інтернеті та в наших мережах. Ви також більше дізнаєтеся про криптографію, автентифікацію й механізми доступу, що є важливими навичками для фахівця з ІТ-підтримки. Наприкінці курсу ми дамо вам необхідні інструменти для оцінки безпеки в організації та визначення оптимальних запобіжних заходів для її захисту.

Сьогодні майже кожна компанія або галузь значною мірою покладається на технології для ведення повсякденного бізнесу. Чи можете ви уявити собі компанію (велику чи малу), яка б працювала без електронної пошти, функціональних комп’ютерів чи Інтернету? Візьмімо, наприклад, маленьку компанію. Їй потрібні певні технології для доступу до кредитних карток. Нещодавні атаки, як-от крипточерв’як WannaCry і масштабні атаки з використанням бот-мережі Mirai, унаочнюють обсяг і масштаб того, як безпека впливає на всіх нас. До цього питання слід ставитися серйозно. Через нашу широку залежність від технологій цифрова безпека важливіша, ніж будь-коли раніше, і її вплив лише зростатиме в усіх галузях і аспектах нашого життя. Тож зробімо так, щоб ви були озброєні належними інструментами для захисту себе й своїх майбутніх клієнтів.

## **Шкідливе програмне забеспечення**

### **Тріада CIA**

Протягом усього цього курсу ми часто вживатимемо одну ключову абревіатуру – "КЦД" (англ. – CIA). Ні, я говорю не про Центральне розвідувальне управління США (теж CIA), хоча воно багато робить для національної безпеки. У нашому випадку "КЦД" означає "конфіденційність, цілісність і доступність". Ці три ключові принципи є основою того, що зазвичай називають "тріадою КЦД". Це керівна модель для розробки правил інформаційної безпеки. Ці три принципи допоможуть вам розробити правила безпеки на робочому місці та у ваших особистих середовищах.

Почнімо з конфіденційності. Конфіденційність означає зберігання в таємниці. В IT це означає зберігання наявних даних надійно прихованими від сторонніх очей. Один із методів конфіденційності, який ви напевно використовуєте щодня, – це захист паролем. Лише ви і ще, можливо, ваш партнер маєте знати пароль для доступу до банківського рахунку онлайн. Для забезпечення конфіденційності потрібно обмежити доступ до даних. Знати, як отримати доступ, мають лише ті, кому це абсолютно необхідно.

Літера "Ц" в абревіатурі "КЦД" означає цілісність. Цілісність – це збереження даних точними й неспотвореними. Дані, які ми надсилаємо чи отримуємо, мають залишатися незмінними протягом усього процесу. Уявіть, що ви завантажили файл з Інтернету. На вебсайті, з якого ви його завантажили, указано, що розмір файлу – 3 МБ. Але після завантаження виявилось, що справжній розмір файлу – 30 МБ. Це сигнал про небезпеку. Щось сталося під час завантаження, щось потенційно небезпечне. Можливо, на вашому жорсткому диску тепер існує небажаний файл. Як ви дізнаєтеся на наступних уроках, таке трапляється дуже часто.

І останнє, але не менш важливе. Літера "Д" у "КЦД" означає "доступність". Доступність означає, що наявна інформація легко доступна тим користувачам, яким вона необхідна. Це може означати багато речей, зокрема готовність до втрати даних або збою системи. Хакерські атаки мають на меті вкрасти у вас чимало речей: час, матеріальні цінності, вашу гідність. Іноді у вас крадуть час, який ви мусите витрачати на відновлення роботи сервісів. Деякі зловмисники утримують вашу систему в заручниках, доки ви не заплатите викуп. Звучить страшно, і насправді так і є. Але саме тому ви тут – щоб дізнатися, як не допускати подібних атак.

Протягом цього курсу ви дізнаєтесь, як кожний аспект безпеки пов’язаний із цими трьома ключовими принципами: конфіденційністю, цілісністю та доступністю.

### **Основні умови безпеки**

Перш ніж докладно розглянути, як зупиняти цифрових злодіїв,  прояснімо деякі терміни. Ми будемо використовувати ці терміни протягом усього курсу, тож їх слід добре зрозуміти.

Перший термін – "ризик". Це – можливість зазнати збитків у разі атаки на систему. Скажімо, ви купуєте новий телефон. Один із заходів безпеки для захисту вашого пристрою – налаштувати блокування екрана за допомогою пароля або ключа, щоб запобігти доступу інших осіб до ваших даних. Блокування екрана – це функція безпеки, що допомагає запобігти небажаному доступу шляхом створення дії, яку потрібно виконати, щоб увійти. Якщо ви вирішите не блокувати екран телефона, ви піддаєте себе ризику того, що хтось може легко отримати до нього доступ і викрасти ваші дані. Навіть просте додавання пароля чи блокування екрана може допомогти захистити ваші особисті дані або дані компанії від потрапляння до сторонніх осіб.

Наступний термін – "вразливість". Це недолік у системі, який можна використати, щоб її зламати. Вразливостями можуть бути прогалини, про які ви можете знати чи ні. Ви можете їхати в тривалу відпустку й зачинити всі двері та вікна у своєму будинку перед виходом. Але ви забудете зачинити вікно ванної кімнати. Тепер це вікно ванної кімнати – вразливість, яку грабіжники можуть використати, щоб проникнути у ваш будинок. Інший приклад: скажімо, ви створюєте додаток і активували обліковий запис для тестування під час розробки, але забули вимкнути його перед запуском додатка. Тепер у вашому додатку є вразливість, яку потенційно може виявити зловмисник. Є особливий тип вразливості, що називається "вразливість нульового дня". Це вразливість, яка не відома розробнику або постачальнику програмного забезпечення, але відома зловмиснику. Назва означає кількість часу, яку мав постачальник програмного забезпечення, щоб зреагувати й усунути вразливість, а саме "нуль днів".

Ще один ключовий термін – "експлойт". Це програма, за допомогою якої використовується вада безпеки чи вразливість. Зловмисники створюють експлойти для вразливостей, які знаходять у ПЗ, щоб заподіяти шкоду системі. Скажімо, зловмисниця виявляє вразливість нульового дня. Вона вирішує скористатися раніше не відомою вадою та пише код експлойту нульового дня. Цей код буде спрямований на використання цієї конкретної невідомої вади, щоб отримати доступ до систем і заподіяти їм шкоду. Не надто приємна новина.

Наступний важливий термін – "загроза". Це ймовірність небезпеки, яка може використати вразливість. Загрози – це просто можливі зловмисники, свого роду грабіжники. Не всі грабіжники намагатимуться проникнути у вашу оселю, щоб викрасти найцінніші речі, але вони можуть це зробити, а тому вважаються загрозами.

Наступний термін – "хакер". У світі безпеки це той, хто намагається проникнути в систему чи використати її. Більшість із нас асоціює термін "хакер" зі зловмисниками. Але насправді є два поширені типи хакерів. Є так звані "чорні хакери", які намагаються проникнути в системи, щоб зробити щось шкідливе. Але є й "білі хакери", які намагаються знайти слабкі місця в системі, але при цьому попереджають власників цих систем, щоб вони могли виправити ситуацію, перш ніж хтось інший заподіє шкоду. Існують й інші типи хакерів, але ці два – основні й найважливіші для нас зараз.

Останній термін, який потрібно знати, – "атака". Це – фактична спроба заподіяти шкоду системі. Дуже важливо усвідомлювати можливі загрози й вразливості системи, щоб мати змогу краще підготуватися до атак. Сумна реальність полягає в тому, що атаки на вашу систему будуть завжди. Але перш ніж почати пошук підземного сховища, щоб провести там решту своїх днів, подумайте про те, що існують способи виявляти та пом’якшувати атаки, і ми допоможемо вам навчитися це робити. У цьому модулі ми поговоримо про деякі найпоширеніші типи атак, які траплятимуться вам вдома й на роботі.

На цьому курсі ви дізнаєтеся, як убезпечити свої системи від цих атак. Виявляється, існують сотні способів атакувати вашу систему. Але також існують і сотні способів запобігти їм. Ми розглядатимемо не всі, а лише найважливіші з них. Тож відкиньте ідею з бункером і приготуйтеся до нових знань, бо ми говоритимемо про реальні речі – реальну безпеку.

### **Шкідливе програмне забезпечення**

Зловмисні програми – це різновид шкідливого ПЗ, що може використовуватися для отримання конфіденційної інформації, видалення чи змінення файлів. Зазвичай вони можуть використовуватися для будь-яких небажаних цілей. Найпоширеніші зловмисні програми, з якими ви будете стикатися, – це віруси, черв’яки, рекламне ПЗ, шпигунське ПЗ, трояни, руткіти, бекдори, ботнети… О так, я знаю, це довгий список. Але ми детально розглянемо кожен із них і навіть дізнаємося про деякі реальні випадки. Та наразі поговорімо про найпоширеніші форми шкідливих програм.

Віруси – це найвідоміший вид шкідливих програм. Вони працюють так само, як віруси у вашому організмі. Якщо ви захворіли, вірус прикріплюється до здорової клітини в організмі, а тоді відтворює себе й поширюється на інші здорові клітини організму. І раптом ви починаєте чхати, хрипіти й почуваєтеся жахливо. Комп’ютерний вірус прикріплюється до певного виконуваного коду, наприклад до програми. У роботі програми бере участь багато файлів, кожен із яких тепер вразливий до зараження вірусом. Отже, вірус реплікує себе в цих файлах, робить свою зловмисну справу та повторює це знову й знову, доки максимально не пошириться. Страшно, чи не так? Тримайтеся, ми лише розпочинаємо.

Черв’яки схожі на віруси, але їм не потрібно кудись прикріплюватися, щоб поширюватися. Черв’яки можуть жити самостійно й поширюватися через такі канали, як мережа. Відомим прикладом комп’ютерного черв’яка був ILOVEYOU (або "Love Bug"), який поширився на мільйони комп’ютерів з ОС Windows. Цей черв’як поширювався електронною поштою. Від когось надходив електронний лист із темою "I Love You" і вкладенням, яке насправді було черв’яком, замаскованим під текстовий файл листа кохання. Текстовий файл насправді був виконуваним файлом, який після його відкриття вчиняв багато атак, зокрема копіював себе в кілька файлів і каталогів, запускав шкідливі програми, заміняв файли, а після цього приховував себе. Черв’як поширювався, викрадаючи електронні адреси, що зберігалися на комп’ютері та в клієнтах чатів жертви. Потім він надсилав цей електронний лист усім контактам в адресній книзі. Love Bug поширився по всьому світу й спричинив збитки на мільярди доларів. Таке от кохання. Це лише одна з багатьох причин, чому в жодному разі не слід відкривати невідомі вкладення електронної пошти.

Рекламні програми – одна з найпомітніших форм зловмисних програм, з якими ви зіткнетеся. Більшість із нас бачить їх щодня. Рекламні програми – це програми, які показують рекламу й збирають дані. Іноді ми завантажуємо рекламні програми на законних підставах. Це стається, коли ви приймаєте умови обслуговування, які дозволяють використовувати безкоштовне ПЗ в обмін на показ реклами. Але в решті випадків вони можуть встановлюватися без вашої згоди та можуть робити шкідливіші речі, ніж просто показувати рекламу.

У грецькій міфології є відомий міф про вторгнення в місто Троя. Греки, які довго намагалися потрапити всередину міських стін, нарешті вирішили сховатися у велетенській дерев’яній статуї коня, яку начебто хотіли принести в дарунок. Троянці пропустили подарунок усередину, а тоді під покровом ночі греки вибралися зі статуї й напали на місто. У сфері комп’ютерної безпеки є зловмисні програми, що функціонують як троянський кінь і мають відповідну назву. Троян – це зловмисна програма, що маскується під щось одне, але робить зовсім інше. Точно як історичний троянський кінь, якого пустили в місто жителі Трої. Комп’ютерний троян має бути прийнятий користувачем, тобто користувач має запустити програму. Ніхто б не став із власної волі встановлювати зловмисну програму на комп’ютері, тому трояни спокушають вас установити їх, маскуючись під інші програми.

Шпигунські програми – це різновид зловмисних програм, які шпигують за вами. Вони можуть стежити за екраном вашого комп’ютера, натисканням клавіш, вебкамерами, а потім передавати чи транслювати всю цю інформацію іншій особі. Це не дуже добре. Клавіатурні шпигуни – поширений тип шпигунських програм, що записують кожне натискання вами клавіші. Така програма може фіксувати всі повідомлення, які ви набираєте, вашу конфіденційну інформацію, паролі й навіть більше.

Вимагачі – тип атаки, що утримує ваші дані або систему в заручниках, доки ви не заплатите певний викуп. Пам’ятаєте про принцип доступності, про який ми говорили в першому відео? Ця атака схожа на спосіб зменшити доступність нашої безпеки? Бінго! Саме так і є. Одним із нещодавніх випадків була атака вимагача WannaCry у травні 2017 року. Ця зловмисна програма, скориставшись вразливістю в старих системах Windows, заразила сотні тисяч комп’ютерів по всьому світу. Найбільш примітним стало те, що ця атака вимкнула системи Національних служб охорони здоров’я Англії, що викликало кризу в цій сфері. Атака вимагача WannaCry зруйнувала системи по всьому світу. Ці типи атак стають дедалі поширенішими, і нам треба бути готовими боротися з ними. Тому продовжуймо роботу.

### **Зловмисне програмне забезпечення. Продовження**

Продовжимо говорити про зловмисні програми.  Ми вже розглянули деякі з основних типів зловмисних програм, що зустрічаються в системі. Це, зокрема, зловмисні програми, віруси, черв’яки, рекламні та шпигунські програми й вимагачі. А якщо наші зловмисники могли б викрадати не лише, наприклад, дані, але й ресурси наших комп’ютерів, як-от процесора? Не хочу вас засмучувати, але й таке теж трапляється.

Існують шкідливі програми, які можуть використовувати чужий комп’ютер для виконання завдання, яке централізовано контролює зловмисник. Такі зламані комп’ютери називають "ботами". Мережу з одного або кількох пристроїв-ботів називають "ботнетом". Ботнети призначені для використання потужності комп’ютерів, під’єднаних до Інтернету, для виконання певної розподіленої функції. Візьмімо, наприклад, майнінг біткоїнів. Для майнінгу біткоїнів потрібен комп’ютер, ресурси якого використовуються для виконання певних обчислень. Наприкінці ви можете отримати винагороду у вигляді певної суми біткоїнів. Популярною атакою є створення ботнетів, щоб, зокрема, майнити біткоїни. Так, замість того, щоб виконувати обчислення на одному комп’ютері, зловмисники можуть мати їх тисячу, щоб виконувати обчислення й отримувати ще більше біткоїнів.

Бекдор – це спосіб потрапити в систему, якщо інші способи не дозволені. Це потаємний вхід для зловмисників. Бекдори найчастіше встановлюються після того, як зловмисник отримує доступ до вашої системи й хоче його зберегти. Навіть якщо ви виявили, що вашу систему зламано, ви можете не усвідомлювати, що в ній існує бекдор. Якщо так, вам потрібно заблокувати його, перш ніж він заподіє ще більшої шкоди.

Ще однією формою зломисних програм, яка може спричиняти чимало проблем, є руткіт. З назви "руткіт" випливає, що це набір для кореневого елемента, тобто набір програм чи інструментів, які може використовувати адміністратор. Він дозволяє змінювати рівень адміністратора операційної системи. Руткіт може бути важко виявити, оскільки він може приховуватися від системи за допомогою самої системи. Хитрий маленький паразит. Руткіт може запускати багато шкідливих процесів, але водночас ці процеси не видно в диспетчері завдань, адже руткіт може приховувати свою присутність.

Логічна бомба – це різновид шкідливих програм, що навмисно встановлюється. Після настання певної події або часу вона запускає зловмисну програму. Широковідомий випадок логічної бомби мав місце у 2006 році. Тоді один нещасливий системний адміністратор банку запустив логічну бомбу й вивів із ладу сервіси компанії, намагаючись знизити ціни на її акції. Колишнього співробітника спіймали й звинуватили в шахрайстві, а потім засудили до восьми років ув’язнення. Не найлогічніша логічна бомба.

<https://www.independent.co.uk/news/business/news/disgruntled-worker-tried-to-cripple-ubs-in-protest-over-32-000-bonus-481515.html>

## **Мережеві атаки**

### **Мережеві атаки**

Мережевою атакою, концепція якої досить проста, але яка може завдати великої шкоди, є атака DNS Cache Poisoning. Ви, мабуть, пам’ятаєте з курсу про основи комп’ютерних мереж, що DNS працює, отримуючи інформацію про IP-адреси й імена, щоб полегшити вам пошук сайту. Атака DNS Cache Poisoning обманним шляхом змушує DNS-сервер прийняти підроблений запис DNS, який указує на зламаний DNS-сервер. Після цього він повідомляє вам підроблені DNS-адреси, коли ви намагаєтеся отримати доступ до дійсних сайтів. Мало того, DNS Cache Poisoning може поширитися й на інші мережі. Якщо інші DNS-сервери отримують свою інформацію DNS від зламаного сервера, вони надаватимуть такі несправжні записи DNS іншим вузлам.

Кілька років тому в Бразилії сталася масштабна атака DNS Cache Poisoning. З’ясувалося, що зловмисникам удалось "отруїти" DNS-кеш кількох місцевих інтернет-провайдерів, вставляючи підроблені записи DNS для різних популярних сайтів, як-от Google, Gmail або Hotmail. Коли хтось намагався відвідати один із цих сайтів, він отримував підроблений запис DNS і спрямовувався на сервер, який контролював зловмисник, де розміщувався невеликий Java-аплет. Після цього користувача обманом змушували встановити аплет, що насправді був зловмисним банківським трояном для викрадення банківських облікових даних. Це приклад реальної шкоди, яку можуть заподіяти атаки DNS Cache Poisoning. Докладніше про це можна прочитати в наступній додатковій статті.

Атака "людина посередині" – це атака, у якій зловмисник розташовується між двома вузлами, які вважають, що спілкуються безпосередньо один з одним. Вочевидь, цю назву потрібно було б змінити, бо хакерами бувають не лише люди. Ця атака відстежує інформацію, що надходить на ці вузли та з них, і потенційно може змінювати її під час передавання.

Поширений вид атаки "людина посередині" – перехоплення сеансу або файлів cookie. Скажімо, ви увійшли в обліковий запис на сайт й забули вийти. Ви вже автентифікували себе на сайті та створили маркер сеансу, який надає вам доступ до цього сайту. Якщо хтось у цей час здійснював перехоплення сеансу, він може викрасти цей маркер і видати себе за вас на сайті, чого ви, звісно, не хотіли б. Це ще один привід подумати про принципи безпеки "КЦД". Ви завжди хочете бути впевнені в тому, що дані, які ви надсилаєте чи отримуєте, є цілісними й неушкодженими.

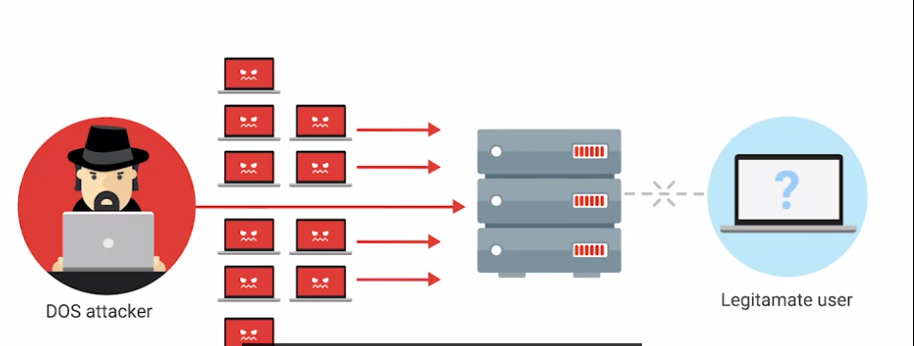
Ще один можливий спосіб атаки "людина посередині" – це атака з фальшивою точкою доступу. Фальшивою є точка доступу, установлена в мережі без відома її адміністратора. Іноді в корпоративних середовищах хтось може під’єднувати маршрутизатор до корпоративної мережі, щоб створити просту бездротову мережу. Звучить досить невинно, так? Насправді – ні! Це може бути досить небезпечно й може надати несанкціонований доступ до авторизованої захищеної мережі. Тепер зловмиснику, щоб отримати доступ до мережі, не потрібно підключатися безпосередньо до мережевого порту, а достатньо, просто стоячи за межами будівлі, під’єднатися до цієї бездротової мережі.

Останній спосіб атаки "людина посередині" називається "злий двійник". Вона схожа на фальшиву точку доступу, але має невелику, утім важливу відмінність. Атака "злий двійник" стає можливою, коли ви під’єднуєтеся до мережі, яка ідентична вашій. Ця ідентична мережа є "злим двійником" нашої мережі, і її контролює зловмисник. Щойно ми під’єднаємося до неї, він зможе відстежувати наш трафік. Цікаво: Фред Візлі коли-небудь робив таке з Джорджем? Мабуть, ні – вони ж чарівники. Вони б вирішували свої проблеми за допомогою магії. Мабуть, це круто.

<https://threatpost.com/major-dns-cache-poisoning-attack-hits-brazilian-isps-110711/75859/>

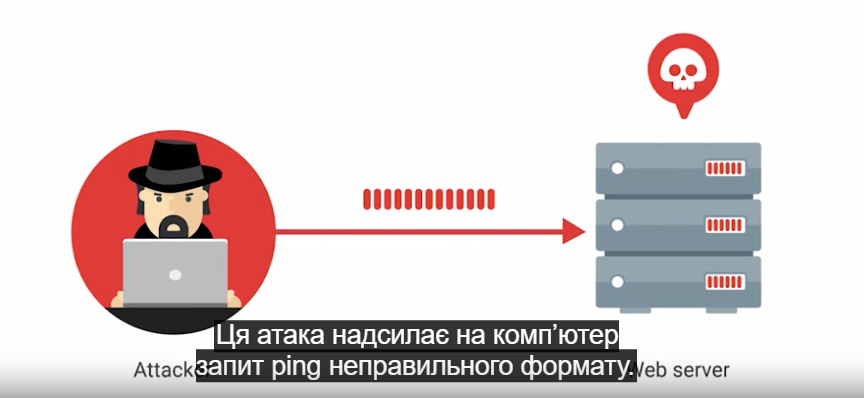
### **Відмова в обслуговуванні**

Атака "відмова в обслуговуванні" (DoS) – це атака, що намагається перешкодити доступу до сервісу законних користувачів шляхом перевантаження мережі або сервера. Подумайте, як ви зазвичай потрапляєте на сайт. Більшість основних сайтів здатні обслуговувати мільйони користувачів. Але для цього прикладу, уявіть, що ваш сайт може обслуговувати лише 10 користувачів. Якби хтось провів атаку "відмова в обслуговуванні", то зміг би просто забрати всі 10 місць і законним користувачам було б відмовлено в доступі до сервісу, бо для них не лишилося б місця.



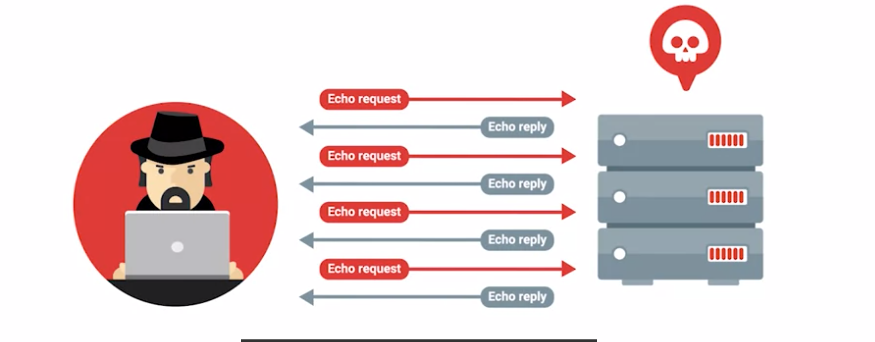
Тепер застосуйте це до такого сайту, як Google чи Facebook. DoS-атаки намагаються зайняти такі ресурси сервісу й запобігти доступу до нього реальних користувачів. Невесела картина.

"Ping of death" (або POD) – досить простий приклад DoS-атаки. Ця атака надсилає на комп’ютер запит ping неправильного формату.



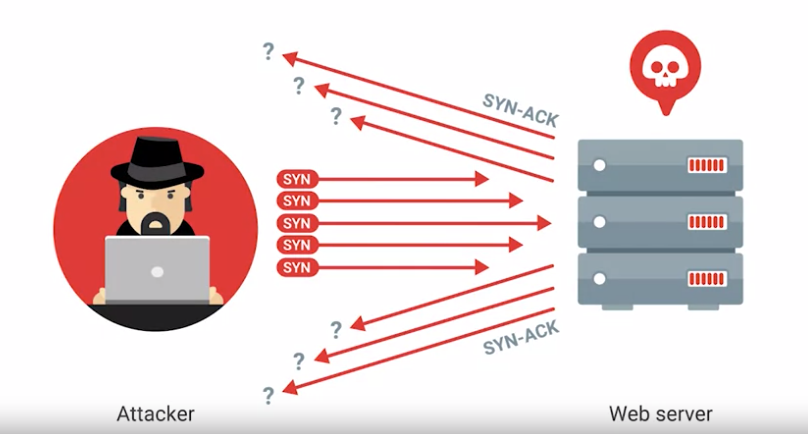
Розмір цього запиту більший, ніж може обробити інтернет-протокол. Як наслідок, виникає переповнення буфера. Це може призвести до збою системи й потенційного виконання зловмисного коду.

Іншим прикладом є атака ping-флуд, що надсилає в систему безліч пакетів ping. Зокрема, вона надсилає ехо-запити ICMP, адже запит ping очікує рівної кількості ехо-відповідей ICMP.



Якщо комп’ютер не може впоратися з цим, може виникнути його перевантаження і збій. Це недобре, ping-флуд, недобре!

Подібно до ping-флуду працює й SYN-флуд. Згадайте: щоб установити TCP-з’єднання, клієнт надсилає пакет SYN на сервер, до якого хоче під’єднатися. Далі сервер надсилає зворотне повідомлення SYN-ACK, після чого клієнт надсилає повідомлення ACK. Під час атаки SYN-флуд сервер піддається бомбардуванню SYN-пакетами. Сервер надсилає назад пакети SYN-ACK, але зловмисник не надсилає повідомлення ACK.



Це означає, що з’єднання залишається відкритим і забирає ресурси сервера. Інші користувачі не зможуть під’єднатися до сервера, що є великою проблемою. Оскільки TCP-з’єднання напіввідкрите, ми називаємо атаки SYN-флуд напіввідкритими атаками. Не дуже зрозуміло, правда? Так і є.

DoS-атаки, які ми вже розглядали, використовують лише один комп’ютер для проведення атаки. Але що, якби зловмисники могли використовувати кілька комп’ютерів? У набагато страшнішому сценарії вони могли б викликати збої в сервісах у більших обсягах і ще швидше. Що ще страшніше, зловмисники справді можуть це зробити. DoS-атака з використанням кількох систем називається розподіленою атакою "відмова в обслуговуванні" (DDoS). Для проведення DDoS-атаки потрібна велика кількість систем, і в цьому зазвичай допомагають ботнети. У цьому сценарії зловмисники можуть отримати доступ до великої кількості комп’ютерів для проведення атаки. У жовтні 2016 року сталася DDoS-атака, мішенню якої став DNS-провайдер Dyn. Підроблені запити на пошук за DNS разом із SYN-флудом, що поширювались через ботнети, перевантажили системи провайдера. Компанія Dyn обслуговувала DNS для великих сайтів, як-от Reddit, GitHub, Twitter тощо. Тож збій її систем одразу вплинув на клієнтів, зробивши ці сервіси недоступними. Не створюйте людям проблем у спілкуванні в Reddit чи Twitter. З власного досвіду знаю: це не дуже добре.

<https://en.wikipedia.org/wiki/DDoS_attacks_on_Dyn>

## **Інші атаки**

### **Атаки на стороні клієнта**

Ми багато говорили про атаки, безпосередньо націлені на жертв, але це не єдиний тип атак, які відбуваються в Інтернеті. Можливо, колись ви займатиметеся розробкою ПЗ чи програмуванням, і вам потрібно буде знати про такі види атак, щоб забезпечувати захист своїх продуктів. Поширеним експлойтом безпеки, який може мати місце під час розробки ПЗ і швидко поширюватися мережею, є вставлення зловмисником шкідливого коду. Атаки такого типу ми називаємо "ін’єкційними". Як працюють ін’єкційні атаки? Чудове запитання.

Для простоти ми не вдаватимемось у подробиці впровадження коду, а уявимо собі автомобіль. Щоб автомобіль їхав, ви заливаєте в нього бензин. Тепер уявімо, що хтось хоче заподіяти шкоду цьому автомобілю. Він може залити у ваш бензобак полунично-банановий молочний коктейль. Це звучить смачно, але може вивести автомобіль із ладу. Тож як із цим боротися? Гіпотетично ви можете запобігти цьому, додавши в автомобіль певний механізм, який прийматиме лише бензин, але не інші рідини. Ін’єкційні атаки на сайтах працюють точно так само, але без апетитних полунично-бананових молочних коктейлів і без надмірно складних рішень.

Ін’єкційним атакам можна запобігати завдяки надійним принципам розробки ПЗ, наприклад перевірці вхідних даних і очищенню даних. Хтось ще голодний? Зробимо перерву на молочний коктейль? Ні? Гаразд, рухаймося далі.

Міжсайтовий скриптинг (або XSS-атаки) – це різновид ін’єкційних атак, де зловмисник може вставляти шкідливий код, націлений на користувача сервісу. XSS-атаки – це поширений метод перехоплення сеансів. Для цього можна просто вбудувати зловмисний сценарій у сайті, щоб користувач несвідомо запустив його у своєму браузері. Цей сценарій може виконати шкідливі дії, як-от викрасти файли cookie жертви чи отримати доступ до даних для входу на сайт. Ммм, печиво.

Ще один різновид ін’єкційної атаки – це "SQL-ін’єкція". На відміну від XSS-атаки, що націлена на користувача, SQL-ін’єкція націлена на весь сайт, якщо він використовує базу даних SQL. Зловмисники потенційно можуть запускати команди SQL, які дозволять їм видаляти дані сайту, копіювати їх і виконувати інші зловмисні команди. А тепер, коли ми закінчили, можна й перекусити.

### **Атаки з метою зламу пароля**

Що би хто не казав, але паролі – найнадійніший стандартний засіб безпеки для запобігання несанкціонованому доступу до облікових записів. На жаль, іноді наші паролі не настільки безпечні або надійні, як це потрібно.

Поширеною атакою для отримання доступу до облікового запису є атака на пароль. Атаки на пароль використовують такі програми, як зламувачі паролів, які намагаються вгадати ваш пароль. І вони працюють дуже ефективно, тож не використовуйте скрізь один пароль FIDO. Він не захистив ваш банківський рахунок, не спрацює і тут. Гаразд, рухаймося далі.

Поширеною атакою на пароль є атака методом "грубої сили", яка намагається підібрати пароль, пробуючи різні комбінації символів, доки нарешті не отримає доступ. Оскільки ця атака вимагає випробовування багатьох комбінацій паролів, зазвичай вона триває довго.

Бачили колись CAPTCHA під час входу на сайт? CAPTCHA використовується, щоб відрізнити людину від комп’ютера. Цей метод запитує вас, чи ви людина, чи ви робот або чи ви танцюрист. Під час атаки паролем, якщо на сайті немає CAPTCHA, автоматизована система може намагатися увійти у ваш обліковий запис, доки не підбере правильну комбінацію символів пароля. Але CAPTCHA перешкоджає таким атакам.

Наступний тип атаки на пароль – "перебір за словником". Перебір за словником не випробовує комбінації "грубої сили", як-от abc1 або ABC1. Натомість він випробовує слова, які часто використовуються в паролях, як-от password, monkey, football. Найкращий спосіб запобігти атаці на пароль – використовувати надійні паролі. Не вживайте справжні слова, що містяться в словнику, і обов’язково використовуйте комбінацію великих і малих літер та символів. Без таких заходів безпеки, як CAPTCHA чи інших засобів захисту облікового запису, типовий зламувач паролів приблизно за хвилину зламає такий пароль, як sandwich. Але йому знадобиться значно більше часу, щоб зламати пароль, який ви бачите на екрані, а саме: s@nDwh1ch. Як бачите, це майже те саме, але цей пароль набагато важче зламати.

### **Оманливі атаки**

Приготуйтеся, бо зараз ми поговоримо про одну з найменш технічно складних, але найбільш дошкульних атак, а саме про "соціальну інженерію". Соціальна інженерія – це метод атаки, який спирається насамперед на взаємодію з людьми, а не комп’ютерами. Ви можете скільки завгодно посилювати засоби захисту. Ви можете витрачати мільйони доларів на найсучаснішу інфраструктуру безпеки. Але якщо системна адміністраторка Сьюзен має повний доступ до вашої системи і в неї обманом виманять її облікові дані, ви ніяк не зможете це зупинити.

Як ми знаємо з найкращих науково-фантастичних фільмів, люди – завжди найслабша ланка в житті, і у вашій системі безпеки також. Соціальна інженерія – це своєрідна аферистська гра, де зловмисники обманними методами намагаються отримати доступ до особистої інформації. Потім вони намагаються змусити користувача виконати певну дію і, по суті, обманом змушують жертву це зробити.

Популярним видом атак соціальної інженерії є фішингова атака. Фішинг зазвичай відбувається, коли жертві надсилається зловмисний електронний лист, замаскований під щось законне. Одна з поширених фішингових атак – електронний лист, що повідомляє про злам вашого банківського рахунку й містить посилання для скидання пароля. За цим посиланням ви переходите начебто на сайт свого банку, але насправді він підробний. Таким чином, вас обманом змушують ввести ваш дійсний пароль і облікові дані, щоб скинути дійсний пароль.

Ще одним різновидом фішингу є цільовий фішинг. Обидві ці фішингові схеми мають однакові кінцеві цілі, але цільовий фішинг націлений на конкретну особу або групу. Фейкові електронні листи можуть містити деякі особисті дані,  як-от ваше ім’я, імена ваших друзів чи родичів. Так вони здаються надійнішими.

Ще одна популярна атака соціальної інженерії – спуфінг електронної пошти. Спуфінг – це коли відправник маскується під щось інше. Як виглядає спуфінг? Це відбувається, коли ви отримуєте електронний лист із підробленою адресою відправника. Ви можете надіслати електронний лист, який начебто походить із будь-якого місця – реального чи ні. Уявіть, що ви відкриваєте електронний лист начебто від вашого друга Браяна. На початку листа буде справжня адреса Браяна, а сам лист закликатиме вас переглянути це смішне посилання. Ви знаєте Браяна. Він чудовий хлопець і завжди надсилає дуже смішні листи, тож ви натискаєте посилання. Раптом ви розумієте, що встановили зловмисну програму. І вже, мабуть, не так добре думаєте про Браяна.

Не всі атаки соціальної інженерії відбуваються цифровим шляхом. Одна з таких атак відбувається через фактичний фізичний контакт. Це так званий "бейтинг", що використовується,  щоб змусити жертву щось зробити. Наприклад, зловмисник може просто залишити десь USB-накопичувач, сподіваючись, що хтось під’єднає його до комп’ютера й подивиться, що на ньому. Але цей хтось встановить зловмисну програму, навіть не знаючи про це.

Ще одна популярна атака, яка можлива без інтернет-з’єднання, – "тейлгейтинг", тобто отримання доступу до забороненої зони або будівлі, проходячи за справжнім працівником. У більшості корпоративних середовищ доступ до будівель обмежується за допомогою карток-ключів тощо. Але зловмисник, використовуючи тактику соціальної інженерії, може змусити працівника подумати, що він тут із законною метою, наприклад для обслуговування будівлі або доставки посилок. Потрапивши всередину, зловмисник отримує доступ до корпоративних активів.

Ми поговорили про досить страшні речі, чи не так? Ви, мабуть, і не знали, що існує стільки способів зламати систему безпеки. Сподіваюся, ви краще зрозуміли поширені атаки, а також їхні ознаки, на які слід звертати увагу. Тепер, коли ви дізналися про основні типи загроз безпеці, розгляньмо детально найкращі способи захисту від них, а також як технічно реалізовуються захищені системи. Але спочатку ми перевіримо ваші знання за допомогою тесту, що охоплює різні типи атак, про які ми говорили в цьому модулі.

# **Модуль 2**

## **Симетричне шифрування**

### **Криптографія**

У дитинстві у вас із братами й сестрами була своя секретна мова, яку не розуміли батьки? Насправді не мало значення, про що ви говорили.  Головне, щоб батьки про це не здогадалися. Це було весело, чи не так? У дитинстві це здавалося вам цікавою грою. Але протягом усього свого існування люди створювали способи засекретити свої повідомлення від інших.

На цьому уроці ми розглянемо, як це відбувається шляхом симетричного й асиметричного шифрування та хешування. Ми також розглянемо, як описати найпоширеніші алгоритми в криптографії та навчимося вибирати оптимальний криптографічний метод для конкретного сценарію. Але перш ніж ми зануримося в нетрі криптографії та дізнаємось про різні її типи, що існують у додатках, зупинімося на деяких основних термінах і загальних принципах, які допоможуть вам зрозуміти, про що йтиметься далі.

Тема криптографії, тобто приховування повідомлень від потенційних ворогів, існує вже тисячі років. Вона надзвичайно розвинулася з появою сучасних технологій, комп’ютерів і засобів телекомунікації. Шифрування – це акт прийому повідомлення у вигляді відкритого тексту та застосування до нього спеціального шифру, щоб на виході отримати спотворене й нерозбірливе повідомлення у вигляді шифротексту. Зворотний процес, коли ми беремо незрозуміле повідомлення й перетворюємо його на читабельний текст, називається дешифруванням.

Наприклад, розгляньмо простий шифр, де ми підставляємо "е" замість "o" і "o" замість "y". Ми візьмемо текст "Hello World" і застосуємо до нього наш базовий шифр. Як гадаєте, який вигляд матиме цільовий шифротекст? Сподіваюсь, у вас вийшло "Holly Wyrld". Цей шифротекст досить легко розшифрувати, оскільки це дуже простий приклад.

Існують набагато складніші й надійніші шифри чи алгоритми, які ми розглянемо пізніше в цьому розділі. По суті, шифр складається з двох компонентів: алгоритму шифрування та ключа. Алгоритм шифрування – це базова логіка або процедура, за допомогою якої відкритий текст перетворюється на шифротекст. Ці алгоритми – це зазвичай дуже складні математичні операції. Але є й деякі дуже прості алгоритми, які ми можемо докладно розглянути й для розуміння яких не потрібно бути доктором математичних наук.

Інший основний компонент шифр – це ключ, який вводить у ваш шифр щось унікальне. Без ключа будь-хто, хто використовує той самий алгоритм, міг би декодувати ваше повідомлення, і насправді жодної таємниці не вийшло б. Отже, підсумуймо: спочатку ви обираєте алгоритм шифрування для кодування свого повідомлення, а потім вибираєте ключ. Тепер у вас є шифр, який ви можете застосувати до повідомлення з відкритим текстом, щоб отримати шифротекст, готовий до поширення й надійно захищений від сторонніх очей. Це не змушує вас почуватися як міжнародний детектив?

Зачекайте. Оскільки основною метою криптографії є захист ваших таємниць від прочитання невповноваженими особами, було б логічно, якби принаймні деякі компоненти шифру також зберігалися в таємниці, чи не так? Ви можете навести аргумент, що зберігання алгоритму в таємниці захищає ваші повідомлення від шпигунів, і технічно ви матимете рацію. Це загальне поняття називається "безпека через неясність".

По суті, це означає: якщо ніхто не знає, який алгоритм ми використовуємо або який загальний принцип безпеки, то ми в безпеці від зловмисників. Це як сховати ключ від будинку під килимком на вході. Допоки грабіжник не знає, що запасний ключ сховано під килимком, ви в безпеці. Але щойно ця інформація розкриється, уся безпека одразу зникне разом із вашими цінностями.

Тож, вочевидь, не варто надто покладатися на безпеку через неясність для захисту комунікацій або систем (або будинку, як у нашому прикладі). Це загальне поняття в криптографії називається "принцип Керкгоффза". Згідно з цим принципом, криптосистема, або набір алгоритмів для генерування ключів і операцій шифрування й дешифрування, які утворюють криптографічний сервіс, має залишатися надійним, навіть якщо відома вся інформація про систему, крім ключа. Це означає, що навіть якщо ваш ворог знає точний алгоритм шифрування, за допомогою якого ви захищаєте свої дані, він усе одно не зможе відновити відкритий текст на основі перехопленого шифротексту.

Ви також можете почути іншу назву цього принципу – "максима Шеннона" або "ворог знає систему". Означає вона те саме. Система має залишатися в безпеці, навіть якщо супротивник точно знає, які системи шифрування ви застосовуєте, доки ключі залишаються захищеними. Ми вже визначили термін "шифрування", але комплексна дисципліна, яка охоплює методи кодування й приховування повідомлень від третіх осіб, називається "криптографія". Вивченням цих методів займається наука криптологія. Протилежність цього, тобто пошук прихованих повідомлень або намагання розшифрувати зашифроване повідомлення, називається "криптоаналіз".

Ці дві галузі спільно еволюціонували протягом історії, розроблялися нові шифри й криптосистеми на заміну тим, які зламувались або виявлялися вразливими. Один із найбільш ранніх зафіксованих описів криптоаналізу дав у IX столітті один арабський математик. Він описував метод частотного аналізу для зламу кодованих повідомлень. Частотний аналіз – це метод визначення частоти появи літер у шифротексті. За цим типом аналізу лежить знання про те, що в письмових мовах певні літери вживаються частіше, ніж інші, і деякі літери частіше поєднуються між собою, ніж інші. Наприклад, в англійській мові найчастіше використовуються літери e, t, a та o. Найчастіше вживані пари з цими літерами: th, er, on і an. Деякі шифри, особливо класичні перестановочний і підстановочний шифри, зберігають відносну частоту літер у відкритому тексті. А тому вони потенційно вразливі для аналізу цього типу.

Під час Першої та Другої світових війн криптографія та криптоаналіз відігравали дуже важливу роль. Відбувся перехід від лінгвістичного й частотного аналізу до аналізу, більше заснованого на математичних принципах. Це було пов’язано з розробкою складніших і продуманіших шифрів. Ключовий переломний момент у галузі криптоаналізу стався під час Другої світової війни, коли союзники США почали застосовувати складну математику, щоб зламувати схеми шифрування для обмеження доступу. Тоді ж уперше почали використовуватися технології автоматизації криптоаналізу в Блечлі-Парку, Англія.

Перший програмований цифровий комп’ютер під назвою "Колос" було розроблено саме для допомоги в цій діяльності. Застосування перших комп’ютерів для зламу криптографії відчинило двері для величезного стрибка вперед і розробки ще складніших і продуманіших криптосистем.

Стеганографія – це споріднений метод, який, утім, виразно відрізняється від криптографії. Це метод приховування інформації від спостерігачів без її кодування. Уявіть, що ви пишете повідомлення невидимим чорнилом. Повідомлення написане відкритим текстом і не потребує декодування, але текст прихований від сторонніх очей. Чорнило невидиме, а щоб його побачити, потрібен механізм, відомий одержувачу.

До сучасних стеганографічних прийомів належать вбудовування повідомлень і навіть файлів в інші файли, як-от зображення або відео. Випадковий спостерігач бачитиме лише фото симпатичного цуценяти. Але якщо відкрити це зображення в стенографічній програмі, вона видобуде повідомлення, приховане у файлі зображення. Що не так секретно, то це те, як цікаво вивчати всі ці шпигунські штучки, чи не так? Далі ми поговоримо про конкретні криптографічні методи та системи.

Кажуть, що поява сучасних методів обчислення призвела до загибелі галузі криптографічного аналізу. Проте ця практика все ще широко застосовується – змінилася лише методологія, оскільки технології трансформували загальну картину. Оскільки квантові обчислення продовжують розвиватися, є підозри, що сучасне шифрування може опинитися під загрозою. Це пов’язано з тим, що більшість сучасних алгоритмів шифрування ґрунтуються на [факторизації](https://en.wikipedia.org/wiki/Integer_factorization) великих простих чисел, що є складною з обчислювальної точки зору. Це можна значно прискорити за допомогою квантового обчислення. Саме тому квантові обчислення дозволили б суттєво пришвидшити факторизацію та атаки методом грубої сили на ключі шифрування, що ставить під сумнів майбутнє сучасної криптографії в епоху квантових обчислень.

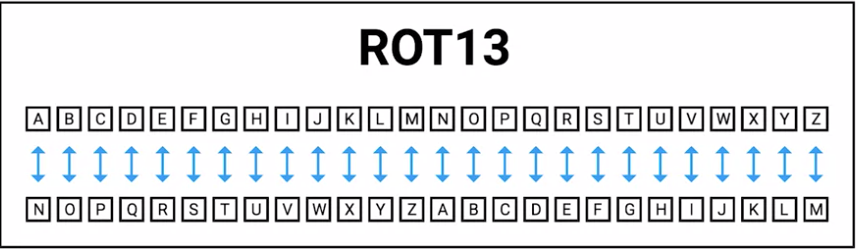
### **Симетрична криптографія**

Досі ми лише в загальних рисах говорили про криптографічні системи та звертали увагу насамперед на поняття шифрування, але не дешифрування. Логічно, що, надсилаючи комусь захищене повідомлення, ви хочете, щоб одержувач міг декодувати й прочитати його, а за потреби й відповісти власним кодованим повідомленням. Отже, розгляньмо детально першу широку категорію алгоритмів шифрування – як вона працює та які її переваги й недоліки.

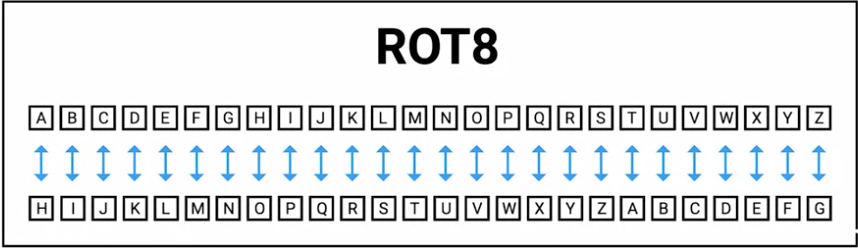
Раніше ми вже говорили про принцип Керкгоффза. Пам’ятаєте, який компонент шифру має вирішальне значення для збереження таємниці? Правильно! Ключ має зберігатися в таємниці, щоб підслуховувач не зможе розшифрувати зашифровані повідомлення. У цьому сценарії ми припускаємо, що використовується так званий "алгоритм симетричного ключа". Такі алгоритми шифрування називаються симетричними, оскільки використовують той самий ключ для шифрування й розшифрування повідомлень.

Розгляньмо простий приклад алгоритму шифрування із симетричним ключем, щоб розібратися в загальному процесі шифрування й розшифрування повідомлень. Підстановочний шифр – це механізм шифрування, який замінює частини відкритого тексту на шифротекст. Пам’ятаєте наш давніший приклад "Hello World"? Це приклад підстановочного шифру, адже ми замінюємо деякі символи на інші. У цьому випадку ключем буде відповідність між символами відкритого тексту й шифротексту. Не знаючи, які саме літери заміняються, ви не зможете легко декодувати шифротекст і відновити відкритий текст. Якщо у вас є ключ або таблиця підстановки, тоді ви можете легко виконати зворотний процес і розшифрувати закодоване повідомлення, просто виконавши зворотну операцію.

Відомим прикладом підстановочного шифру є шифр Цезаря, тобто підстановочний алфавіт. У цьому випадку ви замінюєте символи алфавіту на інші, зазвичай шляхом зміщення або обертання алфавіту – набору цифр чи символів. Ключем є число зміщення. Інший популярний приклад називається ROT13.



Тут алфавіт зміщений на 13 символів, але фактично ROT13 – це шифр Цезаря, що використовує ключ 13. Повернімося до нашого прикладу "Hello World" і спробуймо закодувати його за допомогою шифру ROT13. Наш шифротекст матиме такий вигляд: URYYB JBEYQ. Щоб повернутися до відкритого тексту, ми просто виконуємо зворотну операцію, шукаючи символи в цільовій частині таблиці підстановки. Ви можете помітити певну особливість у таблиці підстановки ROT13 або в тому факті, що ми зміщуємо алфавіт на 13 символів. 13 – це рівно половина літер алфавіту. Отже, шифр ROT13 є зворотним сам до себе. Це означає, що ви можете відновити відкритий текст із шифротексту, застосувавши операцію ROT13 до шифротексту. Якщо ми виберемо інший ключ (скажімо, 8), як ми зможемо зробити те саме? Давайте перевіримо.

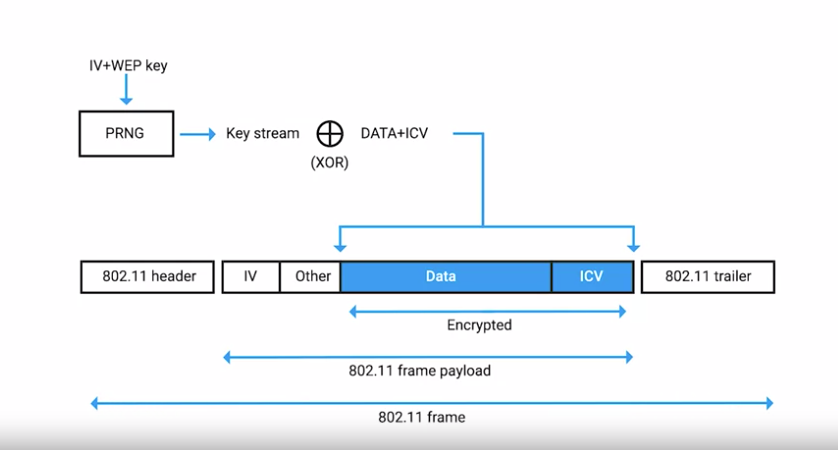


Ось таблиця підстановки зі зміщенням на 8, що дає нам шифротекст OLSSV DVYSK. Якщо ми ще раз застосуємо шифр, то отримаємо такий текст: VSZZC KCFZR. У нас не виходить виконати зворотний процес шифрування, чи не так?

Є ще дві категорії, до яких можна віднести шифри із симетричним ключем. Вони є або блоковими, або потоковими шифрами. Це стосується того, як шифри обробляють відкритий текст, який треба зашифрувати. Потоковий шифр, як випливає з назви, бере потік вхідних даних і шифрує потік по одному символу або одній цифрі, даючи на виході по одному зашифрованому символу або цифрі. Отже, існує чіткий взаємозв’язок між вхідними й зашифрованими даними. Інша категорія симетричних шифрів – блокові шифри. Шифр приймає вхідні дані, поміщає їх у блок даних фіксованого розміру, а потім кодує весь блок як одне ціле.

Якщо дані, що шифруються, недостатньо великі, щоб заповнити блок,  додаються додаткові місця, щоб відкритий текст заповнював блоки рівномірно. Загалом, потокові шифри швидші й менш складні для реалізації, але вони можуть бути менш надійними, ніж блокові шифри, якщо ключі генеруються й обробляються не як слід.

Якщо той самий ключ використовується для шифрування даних двічі або більше, такий шифр можна зламати й відновити відкритий текст. Щоб уникнути повторного використання ключа, застосовується так званий ініціалізаційний вектор (ІВ). Це біт випадкових даних, інтегрований у ключ шифрування, і для шифрування даних використовується утворений таким чином об’єднаний ключ. Ідея полягає в тому, що, якщо у вас є один спільний головний ключ, а потім генерується одноразовий ключ шифрування, цей ключ шифрування використовується лише один раз шляхом генерування нового ключа на основі головного ключа й ІВ. Щоб декодувати зашифроване повідомлення, ІВ має бути надіслано у відкритому тексті разом із зашифрованим повідомленням. Наочний приклад цього можна побачити у фреймі 802.11 зашифрованого бездротового пакета WEP. ІВ у формі відкритого тексту вставляється безпосередньо перед зашифрованими корисними даними.



У наступному відео ми детальніше розглянемо симетричне шифрування, проілюструвавши деякі найпопулярніші алгоритми, і дізнаємося про плюси й мінуси використання симетричного шифрування.

### **Алгоритми симетричного шифрування**

В останньому розділі ми в загальних рисах розглянули, що таке алгоритми симетричного шифрування, і навели базовий приклад шифру Цезаря як різновиду підстановочного шифру. За його допомогою ми б не змогли захистити нічого цінного, правда? Мають бути складніші й надійніші симетричні алгоритми, чи не так? Звісно, вони є.

Одним із найбільш ранніх стандартів шифрування є DES, що розшифровується як "стандарт шифрування даних". Стандарт DES в 1970-х роках розробила компанія IBM на основі деяких вхідних даних від Агентства національної безпеки США. DES було схвалено як офіційний федеральний стандарт обробки інформації (FIPS) у США. Це означає, що DES було схвалено як федеральний стандарт шифрування та захисту державних даних. DES – це симетричний блоковий шифр, що використовує 64-бітові ключі та працює з 64-бітовими блоками. Хоча розмір ключа технічно становить 64 біти, 8 бітів використовуються лише для перевірки парності – це проста форма перевірки на наявність помилок. Це означає, що реальна довжина ключа DES становить лише 56 бітів.

Коротка примітка про розміри ключів шифрування, оскільки ми ще це не розглядали. В алгоритмах симетричного шифрування один ключ використовується для шифрування й розшифрування. Усе інше таке саме. Ключ – це унікальний елемент, що захищає ваші дані. Симетричний ключ має зберігатися в таємниці, щоб забезпечувати конфіденційність даних, які захищаються. Розмір ключа, визначений у бітах, – це загальна кількість бітів або даних, які утворюють ключ шифрування. Таким чином, ви можете уявляти розмір ключа як верхню межу кількості всіх можливих ключів для певного алгоритму шифрування. Довжина ключа є надважливою в криптографії, оскільки вона, по суті, визначає максимальну потенційну надійність системи.

Уявіть собі ідеальний алгоритм симетричного шифрування, що не має недоліків або слабких місць у самому алгоритмі. У цьому сценарії єдиний можливий спосіб для супротивника зламати шифрування – атакувати ключ, а не алгоритм. Один зі способів атаки – просто вгадати ключ і перевірити, чи повідомлення декодується правильно. Це називається атакою "грубої сили". Збільшення довжини ключа захищає від атак цього типу.

Візьмімо як приклад ключ DES. Довжина 64 біти мінус 8 бітів парності дає нам ключ довжиною 56 бітів. Це означає, що існує максимум 2 в 56-му ступені, або 72 квадрильйони можливих ключів. Здається, що це просто безліч ключів, і в 1970-х роках так і було. Але з розвитком технологій комп’ютери стали швидшими й продуктивнішими, а тому 64-бітові ключі швидко виявилися замалими. Атаки, що колись здавалися лише теоретичними з огляду на розмір ключа, стали реальністю в 1998 році, коли Electronic Frontier Foundation (EFF) розшифрувала повідомлення, зашифроване за допомогою DES, лише за 56 годин.

Через властиву DES ненадійність з огляду на малий розмір ключа, розроблялися й пропонувалися інші алгоритми на заміну. У 1980-1990-х роках з’явилася низка нових алгоритмів. Багато з них зберегли 64-бітовий розмір блоків, але використовували більший розмір ключа, що дозволило легше замінити DES. У 1997 році Національний інститут стандартів і технологій (NIST) вирішив замінити DES на новий алгоритм, і в 2001 році було схвалено AES – покращений стандарт шифрування, за результатами міжнародного конкурсу. AES також є першим і єдиним загальнодоступним шифром, схваленим для використання з абсолютно секретною інформацією Агентством національної безпеки США. AES також є симетричним блоковим шифром, як і DES, який він замінив. Але в AES використовуються 128-бітові блоки – удвічі більші, ніж у DES, і він підтримує довжину ключа 128 бітів, 192 біти або 256 бітів.

Через великий розмір ключа атаки "грубої сили" на AES наразі є лише теоретичними, адже потрібна обчислювальна потужність (або потрібний час із використанням сучасних технологій) перевищує всі сьогоднішні можливості. Хочу підкреслити, що ці алгоритми самі по собі є загальними моделями шифрів. Ці моделі ще потрібно реалізувати програмними чи апаратними засобами, щоб можна було застосовувати функції шифрування на практиці.

Розглядаючи різні алгоритми шифрування, важливо звертати увагу на швидкість і простоту реалізації. В ідеалі, алгоритм не має бути надто складно реалізувати, оскільки ускладнена реалізація може призводити до помилок і потенційного зниження безпеки через помилки, зроблені під час реалізації. Швидкість важлива, оскільки іноді дані шифруються шляхом кількаразового застосування до них шифру. Такі типи криптографічних операцій дуже часто виконують пристрої, тож що швидше їх можна виконати

з мінімальним впливом на систему, то краще. Саме тому деякі платформи реалізують ці криптографічні алгоритми апаратними засобами, щоб прискорити процеси й зняти частину навантаження з процесора. Наприклад, сучасні процесори від Intel або AMD мають вбудовані в них інструкції AES. Це забезпечує набагато вищу обчислювальну швидкість і продуктивність під час виконання криптографічних операцій.

Поговорімо трохи про алгоритм, який свого часу був дуже популярним і широко використовувався, але потім виявився ненадійним, і від нього відмовились. RC4 (або Rivest Cipher 4) – це симетричний потоковий шифр, що отримав широке застосування завдяки своїй простоті та швидкості. RC4 підтримує ключі розміром від 40 до 2048 бітів. Отже, слабкі сторони RC4 не пов’язані з атаками "грубої сили". Але сам шифр містить притаманні йому слабкі місця та вразливості, які можливі не лише теоретично. Є багато практичних прикладів зламу RC4. Нещодавній приклад зламу RC4 пов’язаний з атакою RC4 NOMORE. Ця атака змогла відновити файл cookie для автентифікації із зашифрованого за протоколом TLS з’єднання лише за 52 години. Оскільки це атака на сам шифр RC4, до неї вразливий будь-який протокол, що використовує цей шифр. Незважаючи на це, шифр RC4 використовувався в багатьох популярних протоколах шифрування, як-от у WEP для бездротового шифрування або у WPA – наступнику WEP. Він також підтримувався в SSL і TLS до 2015 року, коли підтримку RC4 було припинено в усіх версіях TLS через властиві цьому шифру слабкі місця. З цієї причини більшість основних браузерів повністю відмовилися від підтримки RC4, разом з усіма версіями SSL, і використовують натомість TLS.

Оптимальна безпечна конфігурація – це TLS 1.2 з AES GCM – спеціальний режим роботи для блокового шифру AES, який, по суті, перетворює його на потоковий шифр. GCM (Galois/Counter Mode) бере початкове значення рандомізації, збільшує його на крок і шифрує це значення, створюючи послідовно пронумеровані блоки шифротекстів. Потім ці шифротексти включаються у відкритий текст, який потрібно зашифрувати. Режим GCM дуже популярний завдяки своїй безпеці, заснованій на шифруванні AES у поєднанні з його ефективністю і тим фактом, що він може працювати паралельно з високою продуктивністю. Докладніше про атаку RC4 NOMORE ви прочитаєте в додатковій статті.

Тепер, коли ми розглянули симетричне шифрування й деякі приклади алгоритмів симетричного шифрування, подумаймо, які переваги й недоліки має використання симетричного шифрування. Через симетричний характер процесу шифрування й розшифрування його відносно легко реалізувати й використовувати. Є лише один спільний секрет, який треба утримувати й зберігати в безпеці. Подумайте про свій домашній пароль Wi-Fi. Є один спільний секрет – ваш пароль Wi-Fi, що дозволяє всім пристроям підключатися. Можете собі уявити, що для кожного вашого пристрою буде окремий пароль Wi-Fi? Це був би жах, і за цим було б важко слідкувати. Симетричні алгоритми також дуже швидкі й ефективні під час шифрування та розшифрування великих партій даних.

А які ж тоді мінуси використання симетричного шифрування? Хоча мати один спільний секрет для шифрування й розшифрування спершу здається зручним, насправді це можете створювати певні ускладнення. Що станеться, якщо ваш секрет буде розкрито? Уявіть, що ваш пароль Wi-Fi було вкрадено й тепер вам треба його змінити. Ви маєте оновити пароль Wi-Fi на всіх своїх пристроях, а також на пристроях своїх друзів і членів сім’ї. Що ви маєте зробити, якщо друг або родич приїжджає до вас у гості й хоче під’єднатися до Wi-Fi? Ви маєте дати йому пароль Wi-Fi, тобто спільний секрет, який захищає вашу мережу Wi-Fi. Зазвичай це не проблема, оскільки ви знаєте цю людину й довіряєте їй. І зазвичай це лише одна або дві людини одночасно. А якби у вас в оселі була вечірка з 50 незнайомцями? До речі, навіщо ви влаштовуєте вдома вечірку з 50 незнайомцями? Але менше з тим. Як ви повідомите пароль Wi-Fi лише тим, кому довіряєте, щоб його не дізналися сторонні? Усе це швидко може стати дуже незручним. На наступному уроці ми розглянемо інші методи захисту даних і інформації, крім алгоритмів із симетричним ключем.

<https://www.rc4nomore.com/>

### **Роб: що робить інженер з систем безпеки**

Мене звати Роб Ентоні і я працюю інженером із безпеки в команді інфраструктури контролю доступу. Є сервери, до яких ви хочете обмежити доступ, тому що вони є конфіденційними. Фінансові системи, до яких повинна мати доступ лише фінансова команда. Отже, саме цим ми займаємось. Ми контролюємо політику, яка надає людям доступ до певних речей у Google.

Інженер із безпеки несе відповідальність за захист Google. Важлива річ, яку люди хочуть робити, – відстежувати свої сервери. У них є налаштований сервер, і вони хочуть бути впевненими, що він дійсно працює. Потрібно, щоб хтось за цим стежив. Вони можуть захотіти отримати віддалений доступ до нього для керування. Цей комп’ютер може знаходитися будь-де в країні, в якомусь центрі обробки даних. Уявіть, що я розмовляю з вами, а поруч сидить зловмисник. Якщо ми говоримо розбірливо, то зловмисник може нас слухати. Якщо ми говоримо через зашифрований код, він нас не зрозуміє.

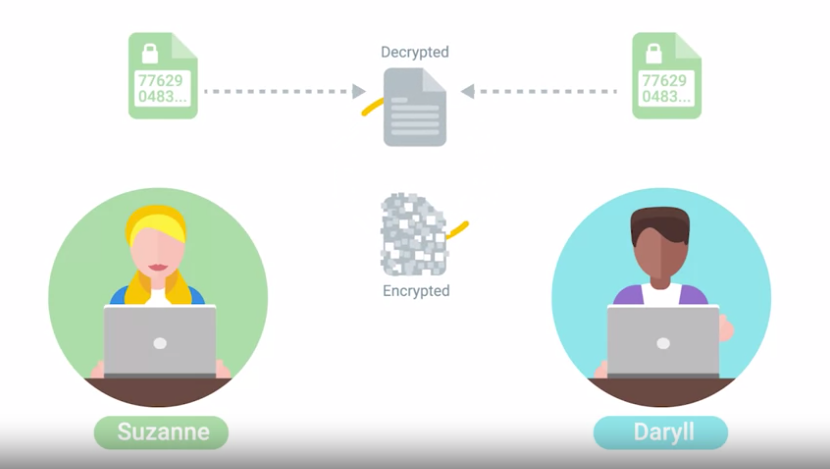
Отже, якщо хтось має розробку й каже: "Ми хочемо, щоб усе було просто, тож хочемо використовувати Telnet". Ми скажемо: "О, ні, ні, ні. Я знаю, що налаштувати щось значно безпечніше потребує більше зусиль. Тому це і є причиною, чому ви повинні це зробити".

## **Відкритий ключ або асиметричне шифрування**

### **Асиметрична криптографія**

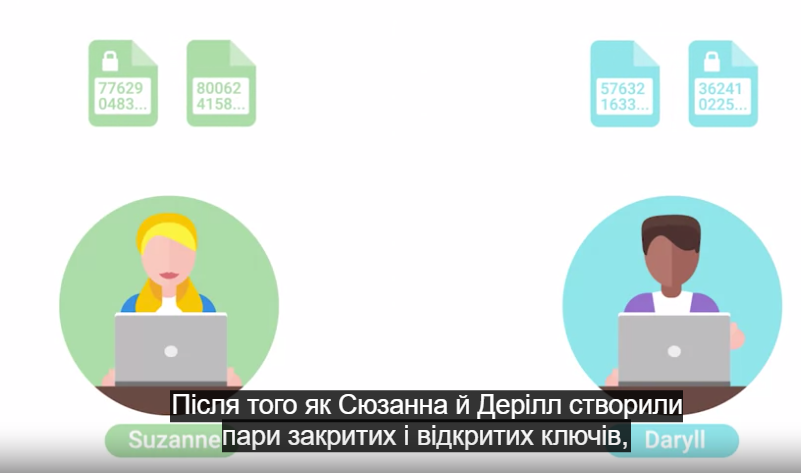
На попередньому уроці ми розглянули одну з двох основних категорій, з яких складається шифрування, – шифрування із симетричним ключем. На цьому уроці ми розглянемо другий клас шифрування – з асиметричним або відкритим ключем. Пам’ятаєте, чому симетричні шифри мають таку назву? Тому, що один ключ використовується для шифрування та розшифрування. Це відрізняється від асиметричних систем шифрування, оскільки, як видно з назви, для шифрування та розшифрування використовуються різні ключі. Тож, як саме це працює?

Уявімо двох людей, які хотіли б спілкуватися безпечно. Називатимемо їх Сюзанна й Дерілл. Оскільки вони використовують асиметричне шифрування в цьому прикладі, перше, що вони мають зробити, це згенерувати закритий ключ.

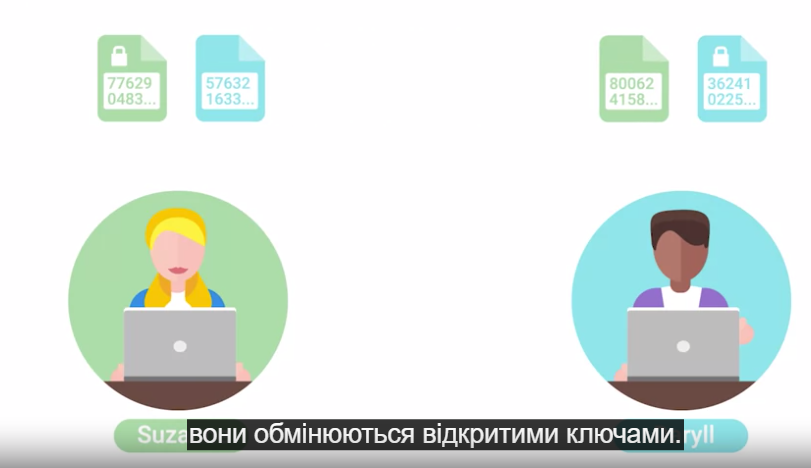


Потім, з використанням цього закритого ключа отримується відкритий ключ. Надійність асиметричної системи шифрування полягає в обчислювальній складності з’ясування відповідності закритого ключа належному відкритому ключу.

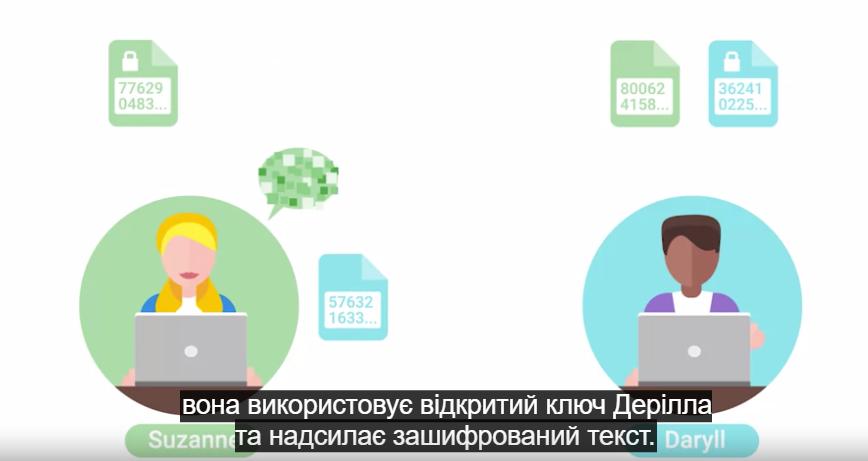
Після того як Сюзанна й Дерілл створили пари закритих і відкритих ключів,



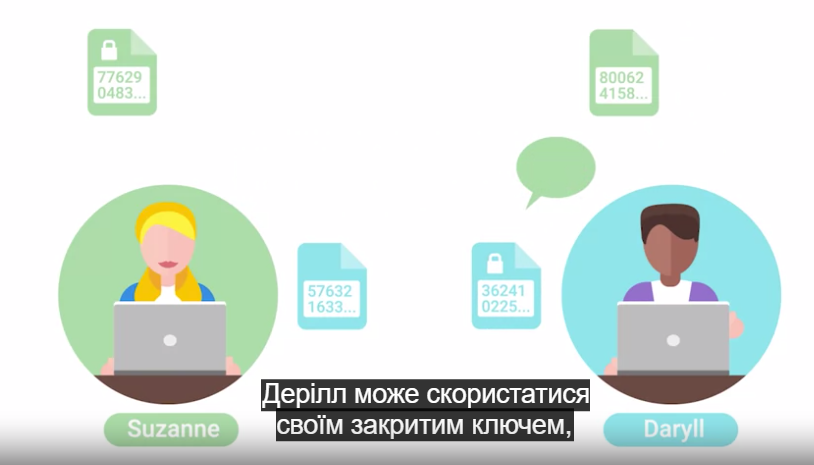
вони обмінюються відкритими ключами.



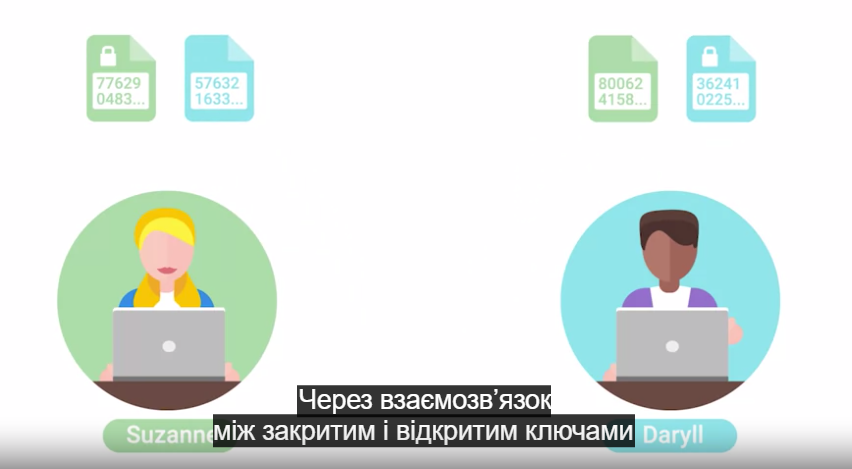
Ви могли здогадатися з назв, що відкритий ключ є публічним і його можна надати будь-кому, тоді як закритий ключ має зберігатися в таємниці. Коли Сюзанна й Дерілл обмінялися відкритими ключами, вони готові почати обмін захищеними повідомленнями. Коли Сюзанна хоче надіслати Деріллу зашифроване повідомлення, вона використовує відкритий ключ Дерілла та надсилає зашифрований текст.



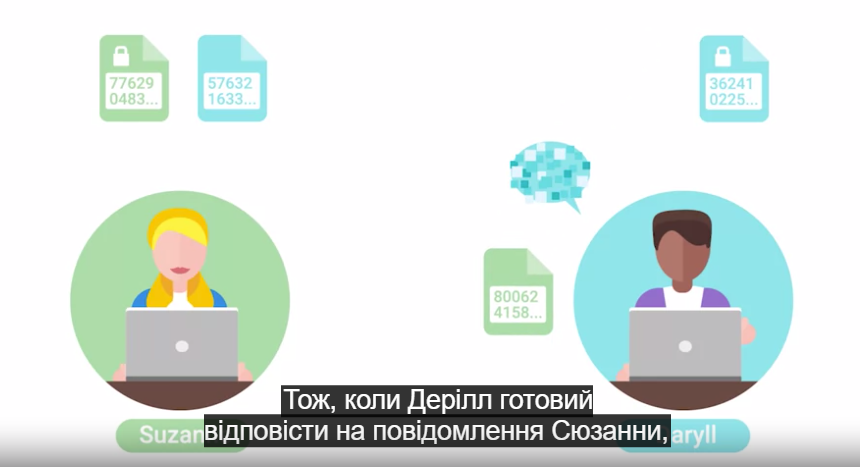
Дерілл може скористатися своїм закритим ключем,



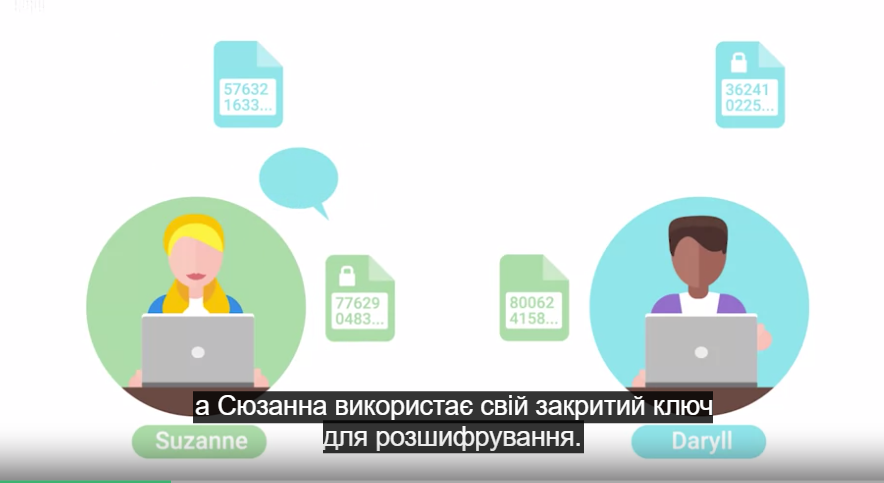
щоб розшифрувати й прочитати повідомлення. Через взаємозв’язок між закритим і відкритим ключами лише закритий ключ Дерілла може розшифровувати повідомлення, зашифровані відкритим ключем Дерілла.



Те саме стосується ключових пар Сюзанни. Тож, коли Дерілл готовий відповісти на повідомлення Сюзанни, він використає відкритий ключ Сюзанни для шифрування повідомлення,

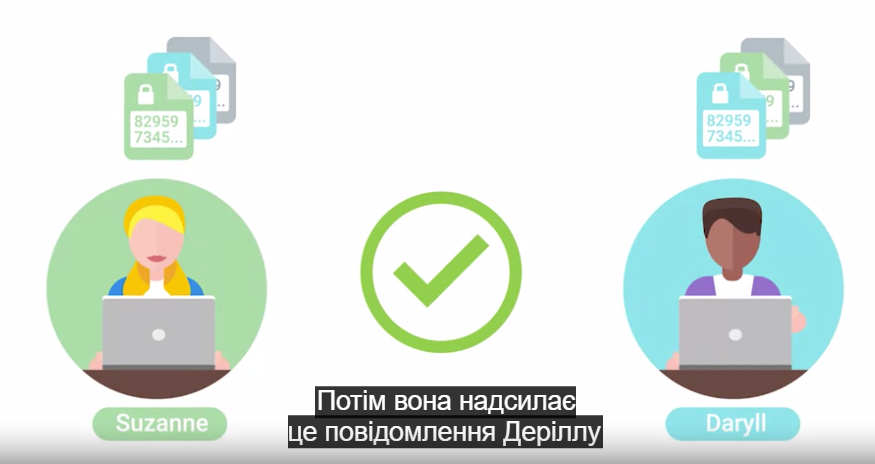


а Сюзанна використає свій закритий ключ для розшифрування.



Тепер ви розумієте, чому ця криптографія називається асиметричною або з відкритим ключем? Ми щойно описали операції шифрування та розшифрування з використанням асиметричної криптосистеми, але є ще одна дуже корисна функція, яку може виконувати ця система – підписи відкритих ключів. Повернімося до наших друзів Сюзанни та Дерілла.

Скажімо, Сюзанна хоче надіслати повідомлення Деррілу й бути впевненою, що Дерілл знає, що це повідомлення від неї, а не когось іншого, і що повідомлення не було змінено чи підроблено. Вона може зробити це, написавши повідомлення та об’єднавши його зі своїм закритим ключем для створення цифрового підпису.



Потім вона надсилає це повідомлення Деріллу разом із пов’язаним цифровим підписом. Ми припускаємо, що Сюзанна й Дерілл уже раніше обмінялися відкритими ключами за цим сценарієм. Тепер Дерілл може перевірити джерело та автентичність повідомлення, поєднавши його з цифровим підписом і відкритим ключем Сюзанни. Якщо повідомлення справді підписано за допомогою закритого ключа Сюзанни, а не чужого, і повідомлення не зазнало змін, тоді цифровий підпис має це засвідчити. Якщо повідомлення було змінено, навіть одним символом пробілу, засвідчення не вдасться, і Деріллу не варто довіряти цьому повідомленню. Це важлива складова асиметричної криптосистеми. Без засвідчення повідомлення будь-хто може скористатися відкритим ключем Дерілла й надіслати йому зашифроване повідомлення, стверджуючи, що воно від Сюзанни.

Три концепції, які нам надає асиметрична криптосистема: конфіденційність, автентичність і незаперечність.

Конфіденційність забезпечується механізмом шифрування-розшифрування, оскільки зашифровані дані зберігаються в таємниці від сторонніх осіб.

Автентичність забезпечується механізмом цифрового підпису, тому можна засвідчити, що повідомлення не було підроблено.

Незаперечність означає, що автор повідомлення не може оскаржити походження повідомлення. Іншими словами, це дає нам гарантію, що повідомлення надійшло від особи, яка стверджує, що є автором.

Чи бачите ви перевагу використання асиметричного алгоритму шифрування проти симетричного? Асиметричне шифрування дає можливість захищеного зв’язку ненадійним каналом, а із симетричним шифруванням потрібен певний спосіб безпечно передати іншій стороні спільний секретний ключ. Якщо це так, здається, що асиметричне шифрування краще, правда ж? Певною мірою. Хоча асиметричне шифрування дійсно добре працює в ненадійних середовищах, його комп’ютерне обчислення дорожче та складніше. З іншого боку, алгоритми симетричного шифрування швидші та ефективніші для шифрування великих обсягів даних. Насправді, багато захищених комунікаційних схем використовують відносні переваги обох типів шифрування, використовуючи їх для різних цілей. Алгоритм асиметричного шифрування обрано як механізм обміну ключами або шифром. Це означає, що симетричний ключ шифрування або спільний секретний ключ безпечно передається іншій стороні, використовуючи асиметричне шифрування для захисту спільного ключа під час його передавання. Після отримання спільного ключа дані можуть бути відправлені швидко, ефективно й надійно, використовуючи алгоритм симетричного шифрування. Розумно, чи не так?

Остання тема, яку слід згадати, і яка певним чином пов’язана з асиметричним шифруванням, – це MAC або коди автентифікації повідомлень. Не плутати з керуванням доступом до медіа або MAC-адресами. MAC – це біт інформації, що дозволяє засвідчити отримане повідомлення, гарантуючи, що повідомлення надійшло від передбачуваного відправника, а не від замаскованої сторонньої особи. Це також гарантує, що повідомлення не було змінено певним чином з метою забезпечення цілісності даних. Це надзвичайно схоже на цифрові підписи, що використовують криптографію з відкритим ключем, чи не так? Попри схожість, він дещо відрізняється, оскільки секретний ключ, що використовується для створення MAC та його перевірки, однаковий. У цьому сенсі він схожий на симетричну систему шифрування, і секретний ключ має бути заздалегідь погоджений усіма сторонами зв’язку, або наданий певним безпечним способом.

Це описує один популярний і безпечний тип MAC під назвою HMAC, або код автентифікації повідомлення з ключем-хешем. HMAC використовує криптографічну хеш-функцію разом із секретним ключем для генерування MAC.

Можуть використовуватися будь-які криптографічні хеш-функції, як-от Shahwan або MD5, а надійність чи безпека MAC залежить від надійності криптографічної хеш-функції.

MAC надсилається разом із повідомленням, яке перевіряється. MAC перевіряється одержувачем за допомогою тієї ж самої операції з отриманим повідомленням, а потім шляхом порівняння обчисленого MAC з тим, який отримано разом із повідомленням. Якщо обидва MAC однакові, тоді повідомлення автентифікується.

Є також MAC на основі алгоритмів симетричного шифрування, чи то блокові, чи потокові, як-от DES та AES, які називаються CMAC, або коди автентифікації повідомлень на основі шифру. Процес подібний до HMAC, але замість використання хеш-функції для створення дайджесту, симетричний шифр зі спільними ключами використовується для шифрування повідомлення, а отриманий результат використовується як MAC.

Конкретним і популярним прикладом CMAC із незначними відмінностями є CBC-MAC, або код автентифікації повідомлення ланцюгуванням шифроблоків. CBC-MAC – механізм створення MAC з використанням блокових шифрів. Це працює шляхом шифрування повідомлення за допомогою блокового шифру, що працює в режимі CBC. Режим CBC – це режим роботи для блокових шифрів, який включає попередньо зашифрований блоковий шифротекст у звичайний текст наступного блоку. Таким чином, він створює ланцюжок зашифрованих блоків, який вимагає повного, немодифікованого ланцюжка для розшифрування. Цей ланцюг взаємопов’язаних зашифрованих блоків означає, що будь-яка модифікація звичайного тексту спричинить інший результат у кінці ланцюжка, що гарантує цілісність повідомлення.

У наступному розділі ми розглянемо деякі поширені приклади алгоритмів і систем асиметричного шифрування. До зустрічі!

### **Алгоритми асиметричного шифрування**

Отже, однією з перших практичних систем асиметричної криптографії є RSA,  назва якої складається з ініціалів трьох винахідників: Рон Рівест, Аді Шамір і Леонард Адлеман. Ця криптосистема була запатентована в 1983 році і випущена у відкритий доступ від RSA Security у 2000 році. Система RSA визначає механізми генерування та розповсюдження ключів разом з операціями шифрування та розшифрування за допомогою цих ключів. Ми не вдаватимемося в деталі математики, оскільки це досить високорівневий матеріал, що не входить до цього курсу. Але важливо знати, що процес генерування ключів залежить від вибору двох унікальних, випадкових і зазвичай дуже великих простих чисел.

DSA, або алгоритм цифрового підпису, є ще одним прикладом асиметричної системи шифрування, хоча його використовують для підписання та перевірки даних. Він був запатентований у 1991 році та входить до складу федерального стандарту обробки інформації уряду США. Подібно до RSA, ця специфікація охоплює процес генерування ключів разом із підписанням і перевіркою даних за допомогою пар ключів.

Важливо зазначити, що безпека цієї системи залежить від вибору випадкової вихідної величини, яка задіяна в процесі підписання. У разі розкриття цього значення або можливості його визначення через те, що просте число не є дійсно випадковим, зловмисник може відновити закритий ключ. Це насправді сталося у 2010 році з ігровою консоллю PlayStation 3 від Sony. Виявляється, вони не гарантували зміну цього випадкового значення для кожного підпису. Це призвело до того, що хакерська група Fail0verflow змогла відновити закритий ключ, який Sony використовувала для підписання ПЗ для своєї платформи. Це дало змогу модераторам створювати й підписувати спеціальне ПЗ, яке могло виконуватися на дуже закритій консольній платформі. Це призвело до того, що ігрове піратство стало проблемою для Sony, сприяючи незаконному копіюванню та розповсюдженню ігор, що спричинило значні втрати продажів. У наступному матеріалі для читання я додав посилання, за якими ви зможете прочитати про це більше.

Ми вже говорили про те, як асиметричні системи зазвичай використовуються як механізми обміну ключами для встановлення спільного секретного ключа для використання із симетричним шифром. Інший популярний алгоритм обміну ключами – DH або Diffie-Hellman, названий на честь винахідників. Розглянемо, як він працює.

Припустімо, що двоє людей бажають спілкуватися незахищеним каналом. Назвемо їх Сюзанна й Дерілл. Я досить прихильно ставлюся до них. По-перше, Сюзанна й Дерілл домовляються про початкове число, яке буде випадковим і дуже великим цілим числом.



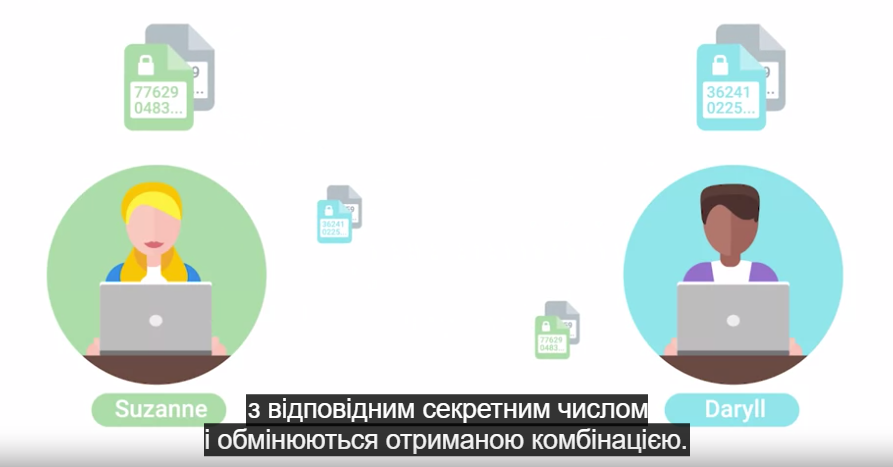
Це число повинно бути різним для кожного сеансу й не має бути таємним. Далі кожен обирає інше випадкове велике число, яке зберігається в таємниці.



Потім вони поєднують своє спільне число з відповідним секретним числом і обмінюються отриманою комбінацією.



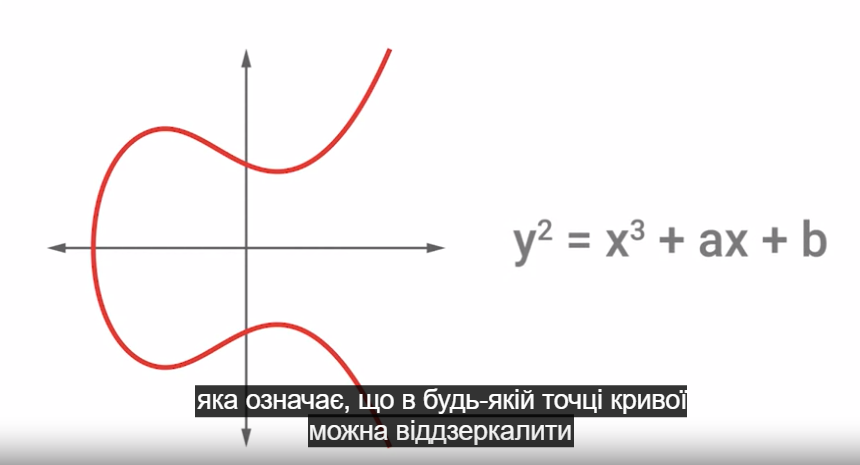
Далі кожен об’єднує своє таємне число з комбінованим значенням, отриманим у попереднього кроці.





Результатом є нове значення, однакове з обох сторін без розкриття достатньої інформації для потенційних підслуховувачів, щоб з’ясувати спільний секретний ключ. Цей алгоритм був розроблений виключно для обміну ключами попри те, що його намагалися адаптувати для цілей шифрування. Він навіть використовувався як складова системи PKI – система інфраструктури відкритих ключів. Ми заглибимося в PKI-системи згодом у цьому курсі.

Еліптична криптографія, або ECC, – система шифрування з відкритим ключем, що використовує алгебраїчну структуру еліптичних кривих над скінченними полями для генерування захищених ключів. Що це взагалі означає? Традиційні системи відкритих ключів використовують факторинг великих простих чисел, а ECC використовує еліптичні криві. Еліптична крива складається з набору координат, які вписуються в рівняння такого типу: Y у квадраті = X у кубі + ax + b. Еліптичні криві мають кілька цікавих і унікальних властивостей. Одна з них – горизонтальна симетрія,



яка означає, що в будь-якій точці кривої можна віддзеркалити вздовж осі x і все одно утворити таку саму криву. Крім того, будь-яка невертикальна лінія перетинатиме криву максимум у трьох місцях.



Саме ця властивість дозволяє використовувати еліптичні криві в шифруванні. Перевага систем шифрування на основі еліптичних кривих полягає в тому, що вони здатні досягти безпеки, подібної до традиційних систем відкритих ключів із меншим розміром ключів. Наприклад, 256-бітний ключ еліптичної кривої можна прирівняти до 3072-бітового ключа RSA. Це дійсно корисно, оскільки зменшується обсяг даних, який необхідно зберігати й передавати під час роботи з ключами.

Як Diffie-Hellman, так і DSA мають варіанти з еліптичною кривою з відповідними назвами ECDH та ECDSA. US NEST рекомендує використовувати шифрування EC, а NSA дозволяє використовувати його для захисту надсекретних даних із використанням 384-бітних ключів EC. Але NSA висловило стурбованість з приводу шифрування EC через його потенційну вразливість до атак квантових обчислень, оскільки технологія квантових обчислень продовжує активно розвиватися.

Я збираюся пригостити Сюзанн і Дерріла напоєм за їхню важку працю. А тим часом ми підготували для вас завдання, в якому ви перевірите свої навички шифрування й розшифрування. Зосередьтеся на розкодуванні всіх подробиць, а ми побачимося на наступному уроці.

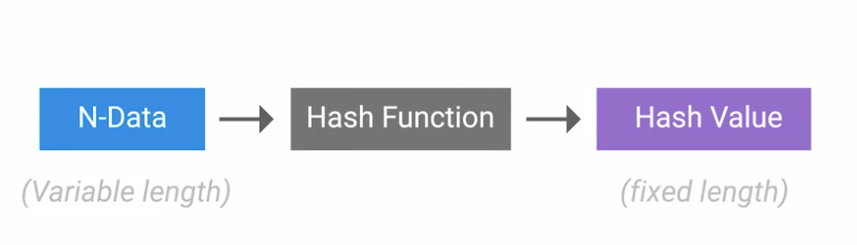
Атака на асиметричне шифрування сталася в 2010 році проти [Sony та її ігрової консолі Playstation 3](https://nakedsecurity.sophos.com/2012/10/25/sony-ps3-hacked-for-good-master-keys-revealed/). Виявилося, що компанія не забезпечувала зміну випадкового початкового значення для кожного підпису. Це призвело до того, що хакерська група під назвою fail0verflow змогла відновити закритий ключ, який Sony використовувала для підписання програмного забезпечення для платформи. Це дозволило розробникам модів писати та підписувати спеціальне програмне забезпечення, яке можна було запускати на дуже закритій консольній платформі. Це [призвело до того, що ігрове піратство](https://www.theguardian.com/technology/gamesblog/2011/jan/07/playstation-3-hack-ps3) стало проблемою для Sony, оскільки воно сприяло незаконному копіюванню та розповсюдженню ігор, і призвело до значних втрат у продажах.

## **Хешування**

### **Хешування**

Досі ми розглядали дві форми шифрування, симетричне й асиметричне. На цьому уроці ми охопимо особливий тип функції, що широко використовується в обчисленнях і особливо у сфері безпеки, – хешування. Ні, це не про сніданок, хоча вони смачні.

Хешування, або хеш-функція, – це тип функції або операція, що аналізує довільно введені дані та генерує результат фіксованого розміру, який називається хешем або дайджестом.



Утворений розмір зазвичай вказується в бітах даних і часто включається в ім’я функції хешування. Це означає, що ви задаєте будь-який обсяг даних функції хешування й отриманий результат завжди матиме однаковий розмір. Але результат має бути унікальним для введених даних, щоб різні введені дані ніколи не утворювали однаковий результат.

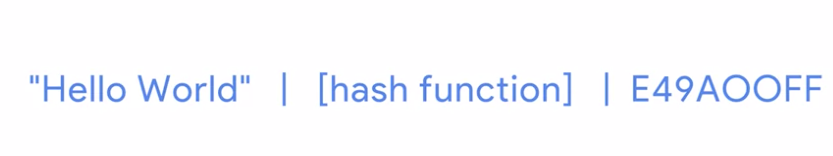
Хеш-функції широко застосовуються в комп’ютерних обчисленнях, здебільшого для ідентифікації унікальності даних. Можливо, ви чули термін "хеш-таблиця" в контексті програмної інженерії. Це тип структури даних, що використовує хеші для прискорення пошуку даних. Хешування також може використовуватися для виявлення дублікатів наборів даних у базах даних чи архівах для пришвидшення пошуку таблиць або для видалення дублікатів даних, щоб заощадити місце.

Залежно від застосування, є різні корисні властивості й функції хешування для різних застосувань. Нас першочергово цікавлять криптографічні хеш-функції, які використовуються для різних застосувань, як-от автентифікації, цілісності повідомлень, цифрових відбитків, виявлення пошкодження даних і цифрових підписів.

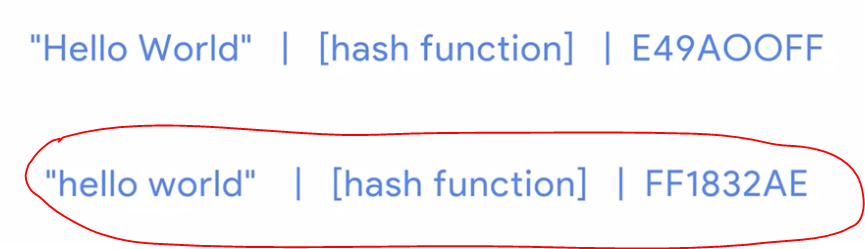
Криптографічне хешування виразно відрізняється від шифрування, тому що криптографічні хеш-функції мають бути односторонніми. Вони схожі тим, що можна ввести звичайний текст у хеш-функцію та отримати результат, який неможливо розібрати. Але за допомогою хеш-результату не можна відновити звичайний текст.

Ідеальна криптографічна хеш-функція повинна бути детермінованою, тобто однакове вхідне значення має завжди повертати однакове хеш-значення. Функція повинна бути швидкою для обчислення та ефективною. Для неї має бути неможливою зворотна функція й відновлення звичайного тексту з хеш-дайджесту. Незначна зміна вхідних даних має призводити до зміни результату без кореляції між зміною вхідних даних й отриманим результатом зміни на виході. Нарешті, функція не повинна дозволяти хеш-колізії, коли різні вхідні дані утворюють однаковий результат.

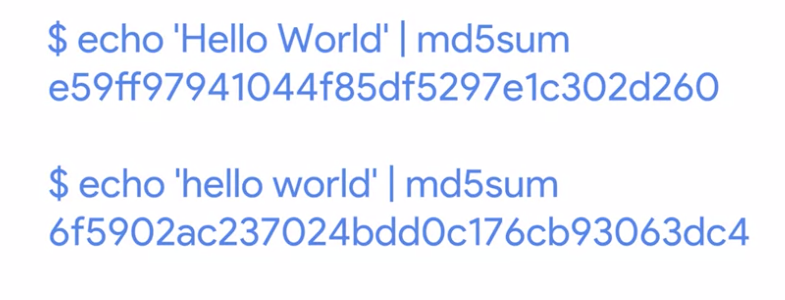
Криптографічні хеш-функції дуже схожі з блоковими шифрами симетричних ключів і вони оперують блоками даних. Насправді, багато популярних хеш-функцій фактично базуються на модифікованих блокових шифрах. Візьмемо базовий приклад, щоб швидко продемонструвати, як працює хеш-функція. Ми скористаємося уявною хеш-функцією для демонстрації. Скажімо, у нас є вхідний рядок "Hello World", який ми задаємо функції хешування, яка генерує отриманий хеш E49AOOFF.



Щоразу, коли ми задаємо цей рядок нашій функції, отримуємо той самий хеш-дайджест. Тепер дуже несуттєво змінимо вхідні дані на "hello world" (усі маленькі літери). Хоча ця зміна здається нам незначною, отриманий хеш дуже відрізняється – FF1832AE.



Ось той самий приклад, але з використанням реальної хеш-функції, у цьому випадку md5sum.



Сподіваюся, що тепер вам зрозуміла концепція функцій хешування. У наступному розділі ми розглянемо деякі приклади алгоритмів хешування та слабкі місця або атаки на функції хешування.

### **Алгоритми хешування**

У цьому розділі ми розглянемо деякі з найпопулярніших функцій хешування, як сучасні, так і в історичному контексті. MD5 – популярна й широко використовувана хеш-функція, розроблена на початку 1990-х років як криптографічна функція хешування. Вона працює на 512-бітових блоках і генерує 128-бітові хеш-дайджести. MD5 було випущено ще в 1992 році, а конструктивний недолік було виявлено в 1996 році, і криптографи рекомендували використовувати хеш SHA-1 як безпечнішу альтернативу. Але цей недолік не вважався критичним, тому ця хеш-функція й надалі поширювалася та адаптувалася.

У 2004 році було виявлено, що MD5 сприйнятливий до хеш-колізій, даючи змогу зловмиснику створювати шкідливий файл, який може генерувати такий самий дайджест MD5, що й інший легітимний файл. Зловмисники – це найгірше, чи не так?

Незабаром після виявлення цієї вади дослідники з безпеки змогли генерувати два різні файли, які мають однакові хеш-дайджести MD5. У 2008 році дослідники з безпеки зробили ще один крок уперед і продемонстрували можливість створення підробленого сертифіката SSL, що підтверджується через хеш-колізію в MD5. Через ці дуже серйозні вразливості в хеш-функції було рекомендовано припинити використання MD5 для криптографічних застосувань до 2010 року.

У 2012 році ця хеш-колізія була використана у злісних цілях у шкідливому програмному забезпеченні, яке використовувало підроблений цифровий сертифікат Microsoft для підпису зловмисних програм, що призвело до появи шкідливого ПЗ, яке виглядало як легітимне програмне забезпечення від Microsoft. Докладніше про шкідливе ПЗ можна дізнатися в матеріалі для читання.

Коли в MD5 були виявлені недоліки, рекомендувалося натомість використовувати SHA-1. SHA-1 є частиною захищеного набору функцій алгоритму хешування, розробленого NSA та оприлюдненого в 1995 році. Він працює з 512-бітовими блоками й генерує 160-бітовий хеш-дайджест. SHA-1 – ще одна широко використовувана криптографічна хеш-функція, яка використовується в популярних протоколах, як-от TLS/SSL, PGP SSH та IPsec. SHA-1 також використовується в системах контролю версій, як-от Git, який використовує хеш для ідентифікації ревізій та гарантування цілісності даних шляхом виявлення пошкодження чи підробки.

SHA-1 і SHA-2 були необхідні для використання в деяких урядових справах США для захисту конфіденційної інформації. Хоча Національний інститут стандартів і технологій США рекомендував припинити використання SHA-1 та покладатися на SHA-2 у 2010 р. Багато інших організацій також рекомендували замінити SHA-1 на SHA-2 або SHA-3. Також основні постачальники браузерів оголосили про наміри припинити підтримку сертифікатів SSL, які використовують SHA-1, у 2017 році.

SHA-1 також має свою частку слабких місць і вразливостей, у яких аналітики з безпеки намагаються продемонструвати реальні хеш-колізії. Упродовж 2000-х років було викладено низку теоретичних атак та продемонстровано певні часткові колізії, але повні колізії з використанням цих методів вимагають значних обчислювальних потужностей. За оцінками, одна така атака потребувала 2,77 мільйона доларів хмарних обчислювальних ресурсів процесора. Ого!

У 2015 році був розроблений інший метод атаки, який не продемонстрував повної колізії, але це був перший випадок, коли було продемонстровано одну з таких атак, що мала значний вплив на майбутню безпеку SHA-1. Те, що раніше було можливо лише теоретично, тепер ставало можливим завдяки ефективнішим методам атак та збільшенню продуктивності обчислень, особливо в просторі обчислень, прискорених графічними процесорами, у хмарних ресурсах.

Повна колізія з цим методом атаки вважалася можливою завдяки хмарним комп’ютерним обчисленням ЦП та ГП та оцінювалася приблизно від 75 до 120 000 доларів, що значно дешевше, ніж попередні атаки. Докладніше про ці атаки та колізії можна прочитати в наступному матеріалі.

На початку 2017 року було оприлюднено першу повну колізію SHA-1. З використанням значних ресурсів центрального й графічного процесорів було створено два унікальні PDF-файли, які утворювали однаковий хеш SHA-1. Розрахункова обчислювальна потужність, необхідна для цього, дорівнювала 6500 років роботи одного ЦП, або 110 років безперервного обчислення одним графічним процесором. Це дуже багато років.

Існує також концепція MIC, або перевірка цілісності повідомлення. Не слід плутати з MAC, або перевіркою автентифікації повідомлення, оскільки те, як вони працюють і від чого захищають, – різні речі. MIC – це, власне, хеш-дайджест повідомлення в запиті. Ви можете розглядати його як контрольну суму для повідомлення, яка гарантує, що контент повідомлення не було змінено під час передавання. Але він чітко відрізняється від MAC, про який ми говорили раніше. Він не використовує секретні ключі, тому повідомлення не автентифікується. Ніщо не заважає зловмиснику змінити повідомлення, повторно обчисливши контрольну суму та змінивши MIC, приєднаний до повідомлення. Можна вважати MIC захистом від випадкового пошкодження чи втрати, але не захистом від підробки або зловмисних дій.

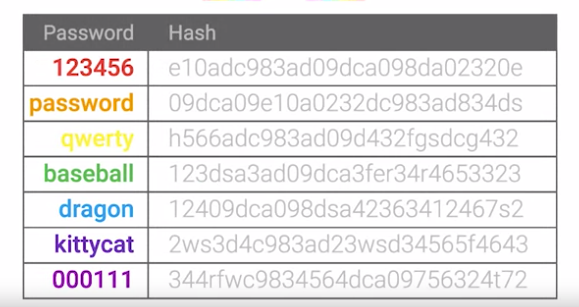
### **Алгоритми хешування (продовження)**

Ми вже згадували про атаки на хеші. Розгляньмо цю тему докладніше, зокрема, як захиститися від цих атак. Важливим застосуванням криптографічних хеш-функцій є автентифікація. Подумайте про вхід в обліковий запис електронної пошти. Ви вводите адресу електронної пошти й пароль. На вашу думку, що відбувається на задньому плані системи електронної пошти для вашої автентифікації? Вона має підтвердити, що введений пароль правильний для вашого облікового запису. Можна просто порівняти введений пароль із паролем у файлі для цього облікового запису. Чи не так? Якщо вони однакові, тоді користувач автентифікується. Здається простим рішенням, але чи виглядає це безпечним для вас? У сценарії автентифікації доведеться зберігати паролі користувачів у звичайному тексті. Це жахлива ідея. Ніколи не варто зберігати вразливу інформацію, як-от паролі, у звичайному тексті. Натомість, слід робити те, що робить майже кожна система автентифікації, зберігати хеш пароля замість власне пароля.

Під час входу в обліковий запис пошти введений пароль обробляється функцією хешування, а потім отриманий хеш-дайджест порівнюється з хешем у файлі. Якщо хеші збігаються, тоді ми знаємо, що пароль правильний, і ви автентифікуєтесь. Пароль не слід зберігати у звичайному тексті, оскільки якщо ваші системи уражені, паролі інших облікових записів стануть подарунком для зловмисника. Якщо зловмисник отримає доступ до вашої системи й зможе просто скопіювати базу даних облікових записів і паролів, очевидно, це буде погана ситуація. Зберігаючи лише хеші паролів, найгірше, що зловмисник зможе отримати, – це хеші паролів, які не дуже корисні самі по собі. А що як зловмисник хоче з’ясувати, які паролі відповідають викраденим хешам? Він виконав би атаку прямим добором проти бази даних хешу паролів. У такому разі зловмисник намагається отримати всі можливі вхідні значення, доки отриманий хеш не збігатиметься з тим, для якого він намагається відновити звичайний текст. Як тільки є збіг, ми знаємо, що вхідні дані, які згенерували той збіг, що збігається з хешем, є відповідним паролем.

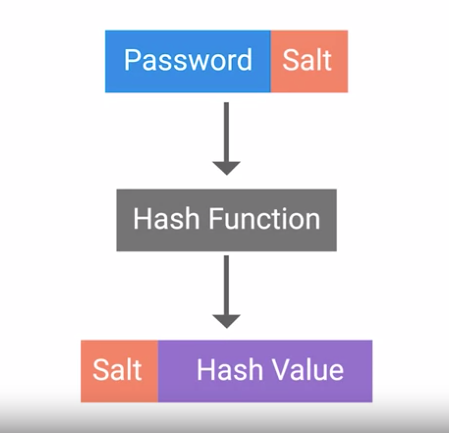
Як ви можете уявити, атака прямим добором може вимагати значних обчислювальних ресурсів, залежно від використовуваної хеш-функції. Важливою характеристикою атаки прямим добором, яку слід згадати, є технічна неможливість цілком захиститися від них. Успішна атака прямим добором навіть проти найзахищенішої системи, яку можна уявити, є справою часу й ресурсів зловмисника. Якщо зловмисник має необмежений час і ресурси, будь-яку систему можна зламати прямим добором. Найліпше, що можна зробити для захисту від цих атак, це підняти планку. Зробити так, щоб це вимагало стільки часу й ресурсів, щоб практично унеможливити її виконання за припустимий термін або з наявними технологіями.

Інший поширений метод підняття обчислювального рівня та захисту від атак прямим добором полягає в багаторазовій обробці пароля хеш-функцією, іноді через тисячі взаємодій. Для кожної спроби вгадати пароль вимагатиметься значно більше обчислень. Це підводить нас до теми райдужних таблиць.



Не обманюйтеся різнобарвним ім’ям. Ці таблиці використовують зловмисники, щоб допомогти пришвидшити процес відновлення паролів із викрадених хешів паролів. Райдужна таблиця – це лише попередньо розрахована таблиця всіх можливих значень паролів та відповідних їм хешів. Ідея атак на райдужні таблиці – це використання дискового простору замість обчислювальної потужності, попередньо обчислюючи хеші й зберігаючи їх у таблиці. Зловмисник може визначити відповідний пароль для певного хешу, просто шукаючи цей хеш у своїй райдужній таблиці. Це не схоже на атаку прямим добором, де хеш обчислюється для кожної спроби. Можна завантажити райдужні таблиці з Інтернету для популярних списків паролів і хеш-функцій. Таким чином зменшується потреба в обчислювальних ресурсах, вимагаючи великий обсяг сховища для зберігання всіх паролів і хеш-даних.

Вам може бути цікаво, як можна захиститися від цих заздалегідь обчислених райдужних таблиць. Ось тут до гри долучаються солі. Ні, я не маю на увазі столову сіль. Сіль пароля – це додаткові випадкові дані, що додаються до функції хешування для створення унікального хешу для комбінації пароля та солі. Ось як це працює.



Вибрана випадковим чином велика сіль приєднується до кінця пароля. Потім комбінація солі та пароля обробляється функцією хешування для генерування хешу, який потім зберігається поряд із сіллю. Тепер для зловмисника це означає, що йому доведеться обчислити райдужну таблицю для кожного можливого значення солі.

Якщо використовується велика сіль, вимоги до обчислення та зберігання, щоб згенерувати корисні райдужні таблиці, стають майже недосяжними. Ранні системи Unix використовували 12-бітну сіль, що складає загалом 4096 можливих солей. Отже, зловмиснику доведеться генерувати хеші для кожного пароля у своїй базі даних понад 4096 разів. Сучасні системи, як-от Linux, BSD і Solaris використовують 128-бітну сіль. Це означає, що існує два у 128 степені можливих значень потужності солі, що становить понад 340 ундецильйонів. Це 340 з 36 нулями.

Очевидно, що 128 бітова сіль піднімає планку досить високо, щоб атака на райдужну таблицю була неможлива в будь-яких реалістичних часових рамках. Просто ще один сценарій, коли додавання солі до чогось поліпшує це. На цьому урок про функції хешування закінчується. Згодом ми поговоримо про реальне застосування криптографії та пояснимо, як вона використовується в різних програмах і протоколах. Але спершу проєкт, який допоможе вам попрактикуватися з хешуванням. #зробімо\_це

## **Використання криптографії**

### **Інфраструктура відкритих ключів**

На цьому уроці ми розглянемо PKI – інфраструктуру відкритих ключів. Спойлер: це критично важлива складова захисту сучасних комунікацій в Інтернеті. Раніше ми говорили про криптографію з відкритим ключем та її використання для безпечного передавання даних ненадійним каналом і перевірки особу відправника з використанням цифрових підписів.

PKI – це система, яка визначає створення, зберігання та розповсюдження цифрових сертифікатів. Цифровий сертифікат – це файл підтвердження, що суб’єкт володіє певним відкритим ключем. Сертифікат містить інформацію про відкритий ключ, суб’єкт, якому він належить, і цифровий підпис від іншої сторони, яка перевірила цю інформацію. Якщо підпис дійсний, і ми довіряємо суб’єкту, який підписав сертифікат, тоді ми можемо довіряти відкритому ключу для використання під час безпечного зв’язку із суб’єктом, який ним володіє.

Суб’єкт, який відповідає за зберігання, видачу та підписання сертифікатів, називається Центром сертифікації. Це невід’ємна складова системи PKI. Також існує Центр реєстрації, який відповідає за перевірку будь-яких суб’єктів, що запитують підписання сертифікатів та їх зберігання в Центрі сертифікації. Ця роль зазвичай групується разом із Центром сертифікації. Для надійного зберігання та індексування ключів необхідне центральне сховище, а система управління сертифікатами певною мірою спрощує керування доступом до зберігання та видання сертифікатів.

Існує декілька типів сертифікатів, які мають різне застосування чи використання. Одним із найбільш знайомих вам є сертифікат сервера SSL або TLS. Це сертифікат, який вебсервер представляє клієнту як частину початкового безпечного встановлення SSL/TLS-з’єднання. Ми розглянемо SSL/TLS докладніше на одному з наступних уроків.

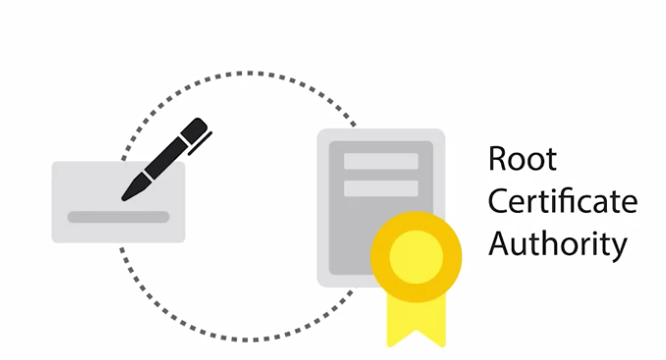
Клієнт, який зазвичай є браузером, має перевірити, чи суб’єкт сертифіката відповідає назві вузла сервера, з яким він намагається з’єднатися. Він також має впевнитись, що сертифікат підписано центром сертифікації, якому він довіряє. Може бути так, що сертифікат є дійсним для декількох вузлів. У деяких випадках може видаватися Wildcard-сертифікат,  в якому ім’я вузла замінено зірочкою, що позначає дійсність для всіх імен вузлів у межах домену.

Крім того, сервер може використовувати так званий самопідписаний сертифікат. Ви, мабуть, здогадалися з назви. Цей сертифікат був підписаний тим самим суб’єктом, який видав сертифікат. Загалом, це те саме, що підписати ваш відкритий ключ із використанням вашого закритого ключа. Якщо ви ще не довіряли цьому ключу, цей сертифікат не пройде перевірку.

Інший тип сертифіката – SSL, або сертифікат клієнта TLS. Це необов’язковий компонент з’єднань SSL/TLS і він зустрічається рідше, ніж сертифікати сервера. Як випливає з назви, цей сертифікат прив’язується до клієнта і використовується для його автентифікації на сервері, даючи змогу керувати доступом до служби SSL/TLS. Він відрізняються від серверних сертифікатів, у яких сертифікати клієнта не видаються публічним центром сертифікації.

Зазвичай оператор послуги має власні внутрішні центри сертифікації, які видають сертифікати клієнта й керують ними для їхньої служби. Також існують сертифікати підпису коду, які використовуються для підписання виконуваних програм. Це дає змогу користувачам цих підписаних програм перевіряти підписи та бути певними, що програму не було підроблено. Це також дозволяє перевірити, що ця програма від автора ПЗ, а не зловмисний близнюк.

Ми згадували про довіру до центру сертифікації, але не пояснили цього. Тож витратимо трохи часу, щоб розглянути, як усе це працює. PKI дуже залежить від довіри у відносинах між суб’єктами та створенні мережі або ланцюга довіри.



Цей ланцюжок довіри має звідкись починатися, і починається він із центру кореневої сертифікації. Ці кореневі сертифікати є самопідписаними, оскільки з них починається ланцюжок довіри. Отже, не існує вищого центру, який може підписати від їх імені. Цей кореневий центр сертифікації тепер може використовувати самопідписаний сертифікат і пов’язаний закритий ключ, щоб почати підписання інших відкритих ключів та видавати сертифікати. Він створює своєрідну деревоподібну структуру з кореневим закритим ключем у верхній частині структури. Якщо кореневий ЦС підписує сертифікат і встановлює в полі з назвою "ЦС" значення "true", це позначає сертифікат як посередник або підлеглий центру сертифікації. Це означає, що суб’єкт, якому видано цей сертифікат, тепер може підписувати інші сертифікати. Цей ЦС тепер має такий самий рівень довіри, що й кореневий. Посередницький ЦС може підписувати сертифікати інших посередницьких ЦС.

Ви можете побачити, як розширення довіри від одного кореневого ЦС до посередників може почати створення ланцюжка. Сертифікат, який не має видавця, як ЦС, називається кінцевим суб’єктом або Leaf-сертифікатом. Подібно до листка на дереві, це кінець структури дерева і його можна вважати протилежністю коренів.

Вам може бути цікаво, як ці кореневі центри сертифікації у підсумку отримали першочергову довіру. Що ж, це дуже хороше запитання. Щоб завантажити цей ланцюжок довіри, ви повинні довіряти сертифікату кореневого центру сертифікації, інакше весь ланцюжок буде недовіреним. Для цього треба розповсюдити сертифікати кореневого центру сертифікації альтернативними каналами. Кожен основний постачальник ОС постачає велику кількість довірених сертифікатів кореневого центру сертифікації зі своєю ОС. Вони зазвичай мають власні програми для полегшення розповсюдження сертифікатів кореневого центру сертифікації. Потім більшість браузерів використовуватимуть сховище кореневих сертифікатів, надане ОС. Розгляньмо докладніше сертифікати, виходячи за межі їх функції.

Стандарт X.509 визначає формат цифрових сертифікатів. Він також визначає список відкликання сертифікатів або CRL, що є засобом розповсюдження списку сертифікатів, які більше не діють. Стандарт X.509 уперше випущений у 1988 році, а поточною сучасною версією стандарту є версія 3. Полями, визначеними у сертифікаті X.509, є версія, якій відповідає сертифікат стандарту X.509.

Серійний номер – унікальний ідентифікатор сертифіката, призначений центру сертифікації, який дає йому змогу керувати окремими сертифікатами та ідентифікувати їх.

Алгоритм підпису сертифіката – це поле вказує, який алгоритм відкритого ключа використовується для відкритого ключа та який алгоритм хешування використовується для підписання сертифіката.

Видавець – це поле містить інформацію про видавця, який підписав сертифікат. Термін дії – тут є два підполя, "Не раніше" й "Не пізніше", які визначають період терміну дії сертифіката.

Тема – це поле містить відомості про суб’єкт, якому було видано сертифікат.

Інформація про відкритий ключ суб’єкта – ці два підполя визначають алгоритм відкритого ключа разом із власне відкритим ключем.

Алгоритм підпису сертифіката – те саме, що й поле "Інформація про відкритий ключ суб’єкта". Ці два поля повинні збігатися.

Значення підпису сертифіката – дані цифрового підпису.

Також є цифрові відбитки сертифіката, які насправді не є полями самого сертифіката, а обчислені клієнтами під час підтвердження чи перевірки сертифікатів. Це просто хеш-дайджести всього сертифіката.

Ви можете прочитати про повний стандарт X.509 у наступному матеріалі. Альтернативою централізованої моделі PKI встановлення довіри та прив’язування ідентичностей є так звана "Мережа довіри". Мережа довіри – це місце, де окремі особи підписують відкриті ключі інших осіб замість центрів сертифікації.



Перш ніж підписати ключ, потрібно перевірити особу за допомогою узгодженого механізму. Зазвичай, перевіряючи документи, що посвідчують особу, як-от водійське посвідчення, паспорт тощо. Після підтвердження особи підписання її відкритого ключа загалом є порукою за цю людину. Ви засвідчуєте, що довіряєте тому, що відкритий ключ належить цій особі. Цей процес має бути взаємним, тобто обидві сторони мають підписати ключі одна одної. Зазвичай люди, зацікавлені у створенні мережі довіри, організовують так звані сторони підписання ключів, де учасники проходять однакову перевірку та підписання. Врешті, відкритий ключ кожного учасника має бути підписаний кожним іншим учасником, який створює мережу довіри.



У майбутньому, коли один із цих учасників початкової сторони підписання встановлює довіру з новим учасником, мережа довіри розширюється, щоб включити нового учасника та інших осіб, яким вони також довіряють. Це дозволяє створювати зв’язок між окремими мережами довіри через осіб та дає змогу мережі довіри зростати.

Стандарт Х.509

<https://www.ietf.org/rfc/rfc5280.txt>

### **Криптографія в дії**

У цьому розділі ми заглибимося в деякі реальні застосування концепцій шифрування, які ми вже розглядали. В останньому розділі ми згадували SSL/TLS, коли говорили про цифрові сертифікати. Тепер, коли ми розуміємо, як працюють цифрові сертифікати та яку вирішальну роль відіграють ЦС, розгляньмо, як вони застосовуються для захисту вебтрафіку через HTTPS.

Ймовірно, ви вже чули про HTTPS, але чи знаєте ви напевне, що це таке та чим відрізняється від HTTP? Дуже просто, HTTPS є захищеною версією HTTP, протокол передачі гіпертексту. Отже, як саме HTTPS захищає нас в Інтернеті? HTTPS також може називатися HTTP через SSL або TLS, оскільки це, власне, капсула для HTTP-трафіку через зашифрований, захищений канал із використанням SSL або TLS.

Можливо, ви почуєте, що SSL і TLS використовується взаємозамінно, але SSL 3.0, остання редакція SSL, застаріла у 2015 році, а TLS 1.2 є поточною рекомендованою редакцією з версією 1.3, що й досі працює. Тепер важливо зазначити, що TLS фактично не залежить від HTTPS, і насправді є загальним протоколом для дозволу безпечного зв’язку автентифікації в мережі. TLS також використовується для захисту інших комунікацій, окрім вебперегляду, як-от VoIP виклики, такі як Skype або Hangouts, електронна пошта, обмін миттєвими повідомленнями та навіть безпека мережі Wi-Fi.

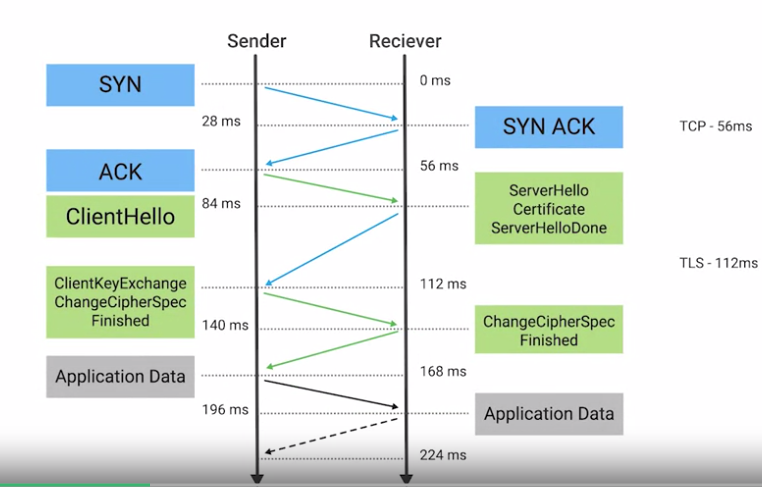
TLS надає нам три речі.

1 – захищена лінія зв’язку, тобто дані, що передаються, захищені від потенційних підслуховувачів.

2 – можливість автентифікувати обидві сторони зв’язку, хоча, здебільшого, лише сервер перевіряється клієнтом.

3 – цілісність комунікації, тобто виконання перевірок, які гарантують незмінність і цілісність переданих повідомлень.

TLS, власне, надає програмі безпечний канал для спілкування зі службою, але має бути механізм для початкового встановлення цього каналу. Саме це мається на увазі, коли йдеться про рукостискання TLS. Я, швидше, комунікабельна людина, але рухаймося далі.



Процес рукостискання починається з клієнта, який встановлює з’єднання зі службою, яка має увімкнений TLS та в протоколі називається ClientHello. Сюди входить інформація про клієнта, як-от версія TLS, яку підтримує клієнт, список підтримуваних наборів шифрів і, можливо, деякі додаткові параметри TLS. Потім сервер відповідає з повідомленням ServerHello, в якому він вибирає найвищу версію протоколу, спільного з клієнтом, та обирає зі списку набір шифрів для використання. Він також передає цифровий сертифікат і кінцеве повідомлення ServerHelloDone.

Після цього клієнт перевірить надісланий сервером сертифікат, щоб переконатися, що він довірений і призначений належному імені вузла. Припускаючи, що сертифікат проходить перевірку, клієнт потім надсилає повідомлення ClientKeyExchange. У цей час клієнт обирає механізм обміну ключами, щоб безпечно встановити спільний секретний ключ із сервером, що буде використовуватися із симетричним алгоритмом шифрування для шифрування всієї подальшої комунікації. Клієнт також надсилає повідомлення ChangeCipherSpec, яке вказує на те, що тепер він переходить на захищений зв’язок, маючи всю інформацію, необхідну для початку зв’язку захищеним каналом.

Слідом за цим йде зашифроване кінцеве повідомлення, яке також служить для підтвердження успішного завершення рукостискання. Сервер дає відповідь ChangeCipherSpec і кінцеве зашифроване повідомлення одразу після отримання спільного ключа. Після завершення дані програми можуть почати передаватися захищеним каналом. Дамо п’ять за це.

Ключ сеансу – це спільний ключ симетричного шифрування, який використовує сеанси TLS для шифрування даних, що передаються в обох напрямках. Оскільки цей ключ походить від відкритого-закритого ключа, якщо закритий ключ уражено, є потенціал для зловмисника декодувати всі попередні передані повідомлення, які були закодовані з ключами, отриманими з цього закритого ключа.

Щоб захиститися від цього, існує поняття прямої секретності. Це властивість криптографічної системи, за якої навіть у випадку ураження закритого ключа, ключі сеансу залишаються в безпеці.

SSH, або захищена оболонка, є безпечним мережевим протоколом, який використовує шифрування для надання доступу до мережевої служби через незахищені мережі. Найчастіше ви зустрічатимете SSH для віддаленого входу в системах на базі командного рядка, але протокол є надзвичайно гнучким і має передумови, що дозволяють довільним мережам і трафіку проходити через ці порти тунелем зашифрованого каналу.

Спочатку він був розроблений як захищена заміна протоколу Telnet й інших протоколів оболонок незахищеного віддаленого входу, як-от rlogin, r-exec. Дуже важливо, щоб віддалений вхід і протоколи оболонки використовували шифрування. Інакше ці служби передаватимуть імена користувачів і паролі, введені з клавіатури, і дані термінала як звичайний текст. Це відкриває можливість для підслуховувача перехопити облікові дані та натискання клавіш, що не дуже добре.

SSH використовує криптографію з відкритим ключем для автентифікації віддаленого комп’ютера, до якого підключається клієнт, і має передумови, що дозволяють автентифікацію користувача через сертифікати клієнта, за бажанням.

Протокол SSH дуже гнучкий і модульний, він підтримує різноманітні механізми обміну ключами, як-от Diffie-Hellman, разом із різноманітними алгоритмами симетричного шифрування. Він також підтримує різні методи автентифікації, включно зі спеціальними, які можна написати. Під час використання автентифікації з відкритим ключем пара ключів генерується користувачем, який хоче автентифікуватися. Потім він має розповсюдити ті відкриті ключі для всіх систем, з якими він хоче автентифікуватися, використовуючи пари ключів.

Під час автентифікації SSH гарантуватиме відповідність представленого відкритого та закритого ключа, який завжди має бути у володінні користувача.

PGP розшифровується як Pretty Good Privacy. Як вам така творча назва? PGP – це програма для шифрування, що дозволяє автентифікувати дані разом із приватністю від третіх сторін, покладаючись на асиметричне шифрування для досягнення цієї мети. Найчастіше вона використовується для зашифрованого зв’язку електронної пошти, але також доступна у вигляді рішення для повного шифрування диска, або для шифрування довільних файлів, документів або папок.

PGP був розроблений Філом Циммерманом у 1991 році й був у вільному доступі для використання будь-ким. Разом із програмою поширювався навіть програмний код. Циммерман був антиядерним активістом, і політичний активізм спонукав його розробити програму для шифрування PGP для полегшення безпечного зв’язку з іншими активістами. PGP злетів одразу після випуску й став популярним у всьому світі, що в підсумку призвело до проблем Циммермана з Федеральним урядом США. У той же час федеральна експортна комісія США класифікувала технологію шифрування, яка використовувала ключі розміром понад 40 біт, як боєприпаси. Це означало, що PGP підлягав схожим обмеженням, як-от ракети, бомби, вогнепальна зброя чи навіть ядерна зброя. PGP був розроблений для використання ключів, розміром не менше ніж 128 біт,

тому він натрапив на ці експортні обмеження, а Циммерман зіткнувся з федеральним розслідуванням щодо широкого розповсюдження свого криптографічного ПЗ.

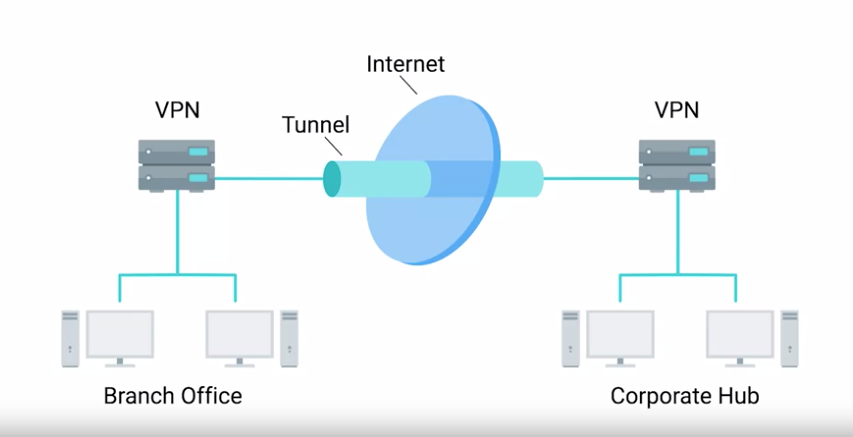
Циммерман застосував творчий підхід для подолання цих обмежень, опублікувавши програмний код у друкованій книзі в твердій обкладинці, яка стала доступною широкому загалу. Ідея полягала в тому, що зміст книги має бути захищений першою поправкою Конституції США. Досить розумно? Зрештою, у 1996 році розслідування було закрито без пред’явлення будь-яких звинувачень, а Циммерману навіть не довелося відвідувати суд.

Ви можете прочитати більше про те, чому він розробив PGP, у наступному матеріалі. PGP вважається дуже захищеним, з відсутністю відомих механізмів зламу шифрування за допомогою криптографічних або обчислювальних засобів. Він порівнюється із шифруванням військового класу, і є численні випадки нездатності поліції та уряду відновити дані, які захищені PGP. У цих випадках правоохоронні органи зазвичай вживали законних заходів для змушення передачі паролів або ключів. Спочатку PGP використовував алгоритм RSA, але його врешті будо замінено на DSA, щоб уникнути проблем із ліцензуванням.

<https://www.philzimmermann.com/EN/essays/WhyIWrotePGP.html>

### **Захист мережевого трафіку**

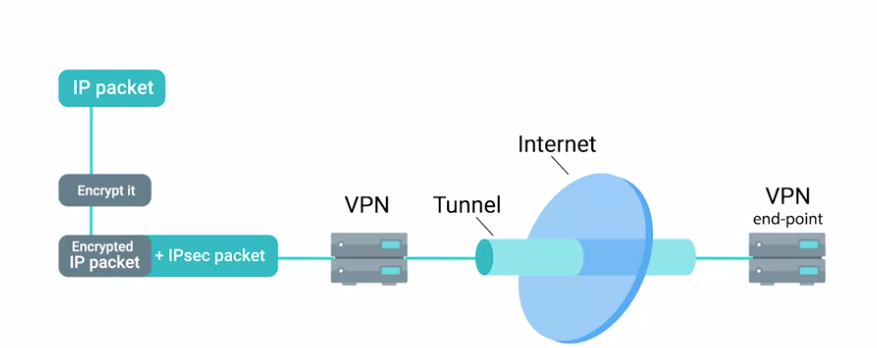
Поговорімо про захист мережевого трафіку. Як ми бачили, шифрування використовується для захисту даних як з огляду конфіденційності, так і для забезпечення цілісності даних. Ця технологія надає захист даних під час їх передавання. Але що, як наша програма не використовує шифрування? Або якщо ми хочемо надати віддалений доступ до внутрішніх ресурсів, занадто чутливих для їх розкриття безпосередньо через Інтернет?  Ми використовуємо VPN, або віртуальну приватну мережу.



VPN – це механізм, який дає змогу віддалено підключити вузол чи мережу до внутрішньої приватної мережі, передаючи дані загальнодоступним каналом, як-от через Інтернет. Це можна вважати своєрідним зашифрованим тунелем, в якому рухається мережевий трафік віддаленої системи, прозоро направляючи наші пакети через тунель віддаленої приватної мережі. VPN також може бути точковою, де два шлюзи з’єднані через VPN.

Загалом, це зв’язок двох приватних мереж через зашифрований тунель. Є безліч VPN-рішень, які використовують різні підходи та протоколи з різними перевагами й компромісами. Розгляньмо деякі з найпопулярніших.

IPsec, або Безпека інтернет-протоколу, – це VPN-протокол, який був розроблений у поєднанні з IPv6. Спочатку від нього вимагалося бути сумісним зі стандартами реалізацій IPv6, але врешті цю вимогу було знято. Він необов’язковий для використання з IPv6. IPsec працює шляхом шифрування IP-пакета й розміщення зашифрованого пакета всередині пакета IPsec.



Потім цей зашифрований пакет спрямовується до кінцевої точки VPN, де він дістається та розшифровується, після чого відправляється до пункту призначення. IPsec підтримує два режими роботи: транспортний і режим тунелю.

Коли використовується транспортний режим, шифрується тільки корисне навантаження IP-пакета, залишаючи IP-заголовки недоторканими.

Увага: заголовки автентифікації також використовуються. Значення заголовків хешуються та перевіряються разом із транспортом та рівнями додатків. Це запобігає використанню будь-чого, що могло б модифікувати ці значення, як-от NAT або PAT.

У режимі тунелю весь IP-пакет, заголовок, корисне навантаження та все шифрується й розміщується всередині нового IP-пакета з новими заголовками. Хоча зазвичай для підтримки VPN використовується не саме VPN-рішення, а L2TP (протокол тунелювання рівня 2). Поширена реалізація L2TP у поєднанні з IPsec, коли необхідна конфіденційність даних, оскільки L2TP не забезпечує шифрування. Це простий протокол тунелювання, який дозволяє помістити в капсулу різні протоколи або трафік через мережу, яка може не підтримувати тип трафіку, що надсилається.

L2TP також може просто відокремлювати й керувати трафіком. Наприклад, інтернет-провайдери використовуватимуть L2TP для надання мережевого доступу кінцевій точці клієнта. Протокол, що створився при поєднанні L2TP і IPsec, отримав назву L2TP IPsec і був офіційно стандартизованим в IETF RFC 3193. Встановлення з’єднання L2TP IPsec працює спершу узгодженням асоціації безпеки IPsec, в якому погоджуються дані захищеного з’єднання, включно з обміном ключами, якщо вони використовуються. Воно також може ділитися секретними ключами, відкритими ключами й рядом інших механізмів. У наступному матеріалі я додав посилання для докладнішої інформації.

Далі встановлюється захищений зв’язок із використанням протоколу Encapsulating Security Payload. Це частина набору протоколів IPsec, які розміщують у капсулу IP-пакети, забезпечуючи конфіденційність, цілісність та автентифікацію пакетів.

Щойно встановлено захищене капсулювання, можна проводити узгодження та створення L2TP-тунелю. Пакети L2TP тепер знаходяться в капсулі IPsec, захищаючи інформацію про внутрішню приватну мережу.

Важлива ознака в цьому налаштуванні – різниця між тунелем і захищеним каналом. Тунель забезпечується L2TP, що дозволяє проходження немодифікованих пакетів з однієї мережі в іншу. З іншого боку, захищений канал надається IPsec, який забезпечує конфіденційність, цілісність і автентифікацію даних, що передаються.

На противагу окремим сеансам або з’єднанням, у деяких реалізаціях VPN також використовується SSL TLS для захисту мережевого трафіку. Таким прикладом є OpenVPN, який використовує бібліотеку OpenSSL для обробки обміну ключами та шифрування даних, разом із каналами керування. Це також дає змогу OpenVPN використовувати всі шифри, реалізовані бібліотекою OpenSSL.

Підтримуються методи автентифікації за завчасно переданими секретними ключами на основі сертифікатів та імені користувача з паролем. Автентифікація на основі сертифіката буде найбезпечнішою, але вона вимагає більше накладних витрат на підтримку та керування, оскільки кожен клієнт повинен мати сертифікат. Автентифікація за іменем користувача і паролем може використовуватися разом із автентифікацією за сертифікатом, надаючи додатковий рівень безпеки.

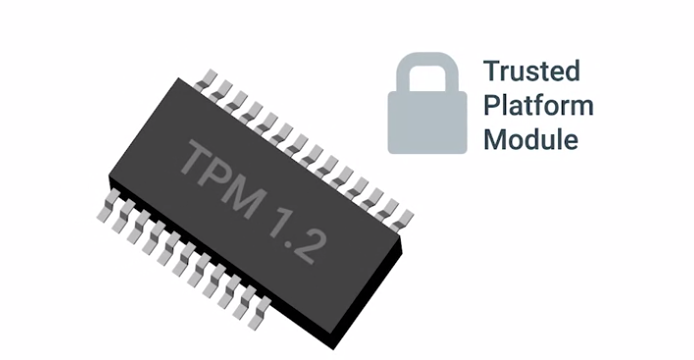
Слід зазначити, що OpenVPN не впроваджує безпосередню автентифікацію за іменем користувача та паролем. Він використовує модулі для підключення до систем автентифікації, які ми розглянемо в наступному розділі. OpenVPN може працювати через TCP або UDP, зазвичай через порт 1194. Він підтримує параметри подання мережевої конфігурації від сервера до клієнта, а також два інтерфейси для мережевого зв’язку. Він може покладатися на тунель Layer 3 IP або на TAP Layer 2 Ethernet. TAP Ethernet є більш гнучким, завдяки чому обробляє ширший спектр трафіку. З точки зору безпеки OpenVPN підтримує до 256 біт шифрування через бібліотеку OpenSSL. Він також працює в просторі користувача, обмежуючи серйозність потенційних вразливостей, які можуть бути наявними. Є багато акронімів для ознайомлення, тож знайдіть хвилину, щоб переглянути їх детальніше, а ми побачимося в наступному відео.

<https://datatracker.ietf.org/doc/html/rfc3193>

<https://openvpn.net/community/>

### **Криптографічне апаратне забезпечення**

З поверненням. Почнемо. Ще одним цікавим застосуванням концепцій криптографії є Trusted Platform Module, або TPM.



Це апаратний пристрій, який зазвичай інтегрується в апаратне забезпечення комп’ютера та є спеціальним криптопроцесором. TPM пропонує безпечне генерування ключів, генерування випадкових чисел, віддалену атестацію, прив’язування даних і пломбування. TPM має унікальний секретний ключ RSA, записаний під час виробництва, що дозволяє TPM виконувати такі речі, як апаратна автентифікація. Вона може виявити несанкціоновані зміни в системі.

Віддалена атестація – це ідея системи, що автентифікує своє програмне забезпечення та апаратну конфігурацію до віддаленої системи. Це дозволяє віддаленій системі визначати цілісність віддаленої системи. Це можна зробити з використанням TPM, згенерувавши безпечний хеш конфігурації системи за допомогою унікального RSA-ключа, вбудованого в TPM.

Іншим використанням цього ключа шифрування з апаратною підтримкою є прив’язування та пломбування даних. Воно передбачає використання секретного ключа для отримання унікального ключа, який потім використовується для шифрування даних. Загалом, це прив’язує зашифровані дані до TPM і, як розширення, до системи, в якій встановлено ТРМ, тож тільки ключі, збережені в обладнанні в TPM, зможуть розшифровувати дані.

Пломбування даних подібне до прив’язування, оскільки дані шифруються за допомогою ключа шифрування з апаратним забезпеченням. Однак, щоб розшифрувати дані, TPM повинен бути у визначеному стані.

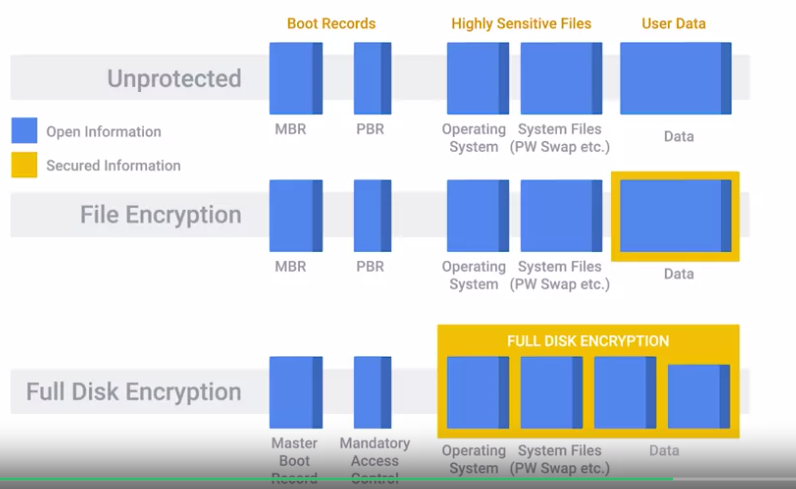
TPM є стандартом з кількома виправленнями, які можуть бути реалізовані як дискретний апаратний чіп, який інтегрований в інший чіп у системі, реалізований у мікропрограмі, або віртуалізований потім гіпервізор. Найбезпечнішою реалізацією є дискретний чіп, оскільки ці пакети мікросхем також включають стійкість до фізичного втручання, щоб запобігти фізичним атакам на чіп.

Мобільні пристрої мають щось подібне, що називається захищеним елементом. Подібно до TPM, це стійка до несанкціонованого доступу мікросхема, часто вбудована в мікропроцесор або інтегрована в материнську плату мобільного пристрою. Вона забезпечує надійне сховище для криптографічних ключів і забезпечує захищене середовище для додатків.

Еволюцією елементів безпеки є довірене середовище виконання, або ТЕЕ, яке дещо розширює цю концепцію. Воно забезпечує повноцінне ізольоване середовище виконання, яке працює разом з основною ОС. Це забезпечує ізоляцію додатків від основної ОС та інших встановлених додатків. Воно також ізолює захищені процеси один від одного під час роботи в TEE. Технологія TPM отримала критику, пов’язану з довірою до виробника. Оскільки секретний ключ записується в обладнанні під час виробництва, виробник на той час мав би доступ до цього ключа. Виробник може зберігати ключі, які потім можуть бути використані для дублювання TPM, що може порушити вимоги безпеки, яку повинен забезпечувати модуль. Був один звіт про фізичну атаку на TPM, яка дозволила досліднику з безпеки переглянути й отримати доступ до всього контенту TPM. Але ця атака вимагала використання електронного мікроскопа та мікроточного обладнання для маніпулювання схемою TPM. Хоча цей процес був неймовірно тривалим і потребував вузькоспеціалізованого обладнання, він довів, що така атака можлива, незважаючи на наявні засоби захисту від несанкціонованого доступу. Докладніше про це можна прочитати одразу після відео. TPM найчастіше використовується для гарантування цілісності платформи, запобігаючи несанкціонованим змінам у системі, як в програмному, так і в апаратному забезпеченні, і повного шифрування диска за допомогою TPM для захисту всього вмісту диска.

Повне шифрування диска, або FDE, як ви могли здогадатися з назви, – це практика шифрування всього накопичувача в системі, а не лише окремих конфіденційних файлів у системі. Це дає змогу захистити весь вміст диска від викрадення або підробки даних. Зараз існує безліч можливостей для впровадження FDE. Наприклад, комерційний продукт PGP, Bitlocker від Microsoft, який дуже добре інтегрується з TPM, Filevault 2 від Apple, і програмне забезпечення з відкритим кодом dm-crypt, яке забезпечує шифрування для систем Linux.

Конфігурація FDE передбачає один розділ або логічний диск, який містить дані, що підлягають шифруванню. Зазвичай, це кореневий розділ, на якому встановлено ОС. Але для того, щоб завантажити цей розділ, його спершу потрібно розблокувати в момент завантаження. Оскільки розділ зашифрований, BIOS не може отримати доступ до даних на ньому для завантаження. Ось чому конфігурації FDE передбачають невеликий незашифрований завантажувальний розділ, який містить такі елементи, як ядро, завантажувач і netRD.



Ці елементи завантажуються під час запуску, після чого користувач отримує запит введення пароля, щоб розблокувати диск і продовжити процес завантаження. FDE також може співпрацювати з TPM, використовуючи ключі шифрування TPM для захисту диска. Крім того, він має цілісність платформи, щоб запобігти розблокуванню диска, якщо змінено конфігурацію системи. Це захищає від атак на зразок апаратного втручання, викрадення диска або клонування.

Перш ніж ми закінчимо цей модуль про шифрування, я хотів зазирнути в основи поняття випадковості. Раніше, коли ми розглядали різні системи шифрування, постійно з’являлася спільна особливість, на яку покладаються ці системи. Чи помітили ви, що це було? Якщо ні – нічого страшного. Це вибір випадкових чисел. Це дуже важливе поняття в шифруванні, оскільки якщо процес вибору чисел не є дійсно випадковим, тоді ймовірна певна закономірність, яку може виявити зловмисник, ретельно спостерігаючи та аналізуючи зашифровані повідомлення упродовж певного часу. Те, що не є справді випадковим, називається псевдовипадковим. Саме з цієї причини операційні системи підтримують так званий пул ентропії. Це, власне кажучи, джерело випадкових даних, яке допомагає генераторам вихідних випадкових чисел.

Також є спеціальні генератори випадкових чисел і псевдовипадкових чисел, які можуть бути включені в пристрій безпеки або сервер для гарантування вибору дійсно випадкових чисел під час генерування криптографічних ключів. Сподіваюся, для вас ці теми з криптографії були цікавими та інформативними. Можу сказати це про себе, коли вперше їх вивчав. У наступному модулі ми розглянемо три аспекти безпеки: автентифікацію, авторизацію та облікові записи. Ці три аспекти неймовірні, і в наступному модулі я розповім чому. Але перш ніж ми туди потрапимо, на вас чекає останній тест з уже вивчених нами понять криптографії.

<https://arstechnica.com/gadgets/2021/08/how-to-go-from-stolen-pc-to-network-intrusion-in-30-minutes/>

### **Відхилення**

Найважчий момент у пошуку роботи — це, звичайно, відмова. Ймовірно, були запитання, на яких ви спіткнулися. Перегляньте їх, відпрацюйте, зробіть їх своїм надбанням. Коли вони знов постануть на наступній співбесіді, ви впораєтеся краще. Не здавайтесь! Продовжуйте надсилати своє резюме, оновлюйте його, щоб показати всі свої технічні навички. Завжди знайдеться місце, де готові взяти когось із початковим рівнем в ІТ. На це піде деякий час, але рано чи пізно ви знайдете своє місце.

### Створення й перевірка пар ключів, шифрування й дешифрування, а також підпис і підтвердження через OpenSSL

На цьому практичному занятті ви дізнаєтеся, як генерувати пари із секретних і відкритих ключів за допомогою утиліти OpenSSL.

OpenSSL – це професійна інструментальна утиліта для протоколу безпеки транспортного рівня (TLS) і протоколу захисту інформації (SSL). Також її використовують як криптографічну бібліотеку загального призначення. OpenSSL розповсюджується на умовах ліцензії типу Apache, тобто ви можете отримати цю утиліту й використовувати її для комерційних і некомерційних цілей (з дотриманням деяких простих ліцензійних умов).

**Генерування ключів**

Перш ніж щось шифрувати й дешифрувати, потрібно мати секретний і відкритий ключі, тож спершу ми згенеруємо їх.

**Генерування секретного ключа**

Пам’ятайте, пара ключів складається з відкритого ключа, який ви можете зробити загальнодоступним, і секретного ключа, який потрібно тримати в таємниці. Тс-с-с. :) Якщо хтось хоче надіслати вам дані, які мають бути недоступними для інших, ця людина може зашифрувати ці дані за допомогою відкритого ключа. Дані, зашифровані відкритим ключем, можна дешифрувати секретним ключем, щоб гарантовано лише ви могли переглядати початкові дані. Саме тому важливо тримати секретні ключі в таємниці. Якщо інша людина отримає копію вашого секретного ключа, вона зможе дешифрувати дані, призначені для вас, а це було б погано.

По-перше, згенеруймо 2048-бітний секретний ключ RSA і розгляньмо його. Щоб згенерувати ключ, введіть у терміналі цю команду:

openssl genrsa -out private\_key.pem 2048

Ви побачите такий результат (або щось подібне):

Generating RSA private key, 2048 bit long modulus (2 primes)

................+++++

..........................................+++++

e is 65537 (0x010001)

Через цю команду можна створити 2048-бітний ключ RSA під назвою private\_key.pem. Назву ключа вказано після параметра -out, і зазвичай вона закінчується на .pem. Кількість бітів указано в останньому аргументі. Щоб переглянути новий секретний ключ, командою cat виведіть його на екран, як і будь-який інший файл:

cat private\_key.pem

Вміст файлу секретного ключа виглядатиме як довга послідовність випадкових символів. Так і має бути, тому не намагайтеся його прочитати:

-----BEGIN RSA PRIVATE KEY-----

MIIEowIBAAKCAQEA4kNMSmssCSYbOnq/UAHGH5xx9gjZaOiST3JQQtJO11L/YeBO

8DOHc7UawNADA/XDBAnGZih1M8T1PGc6Vk5SW2Lb8FMf9zG2XhYpCACFFPJAW00q

s4s1JesdugOprHZ8Jmm/QJl4KuCjlY/XdviCvcbxROIQ2mglR8nW1QWrhECQNBfo

dRSuTwmW3qBSW/Xd5pmTpP4GHCyUfRO9YCF/tZYtVMYg4FOqdGaTHRZbs6peMV4D

lSjZHDonnsGK0UJpxQNbtJEcG7vr7Vl8ziVWY5RUDND7nZYlQlbqxvvqbPPt+px3

4pAZ58eyOqeAmYBc8mwNoXp4YrC2deFng7zrKwIDAQABAoIBAB6SR0Ga33VQ/8bU

BPtzceidg8xhf7asDfDMGkodDmgLn9QCscfEvp2Er9uzf2TOlQ37oCH3f3aCOzxx

GjHFHV2Zquv630vQHLrztZGOOG0PGmD7uTRPL9wyu26BxjA2RioOibfZxKHOfmvb

5pn9k/S+Z6UOAobwIXFktTFNNdKFgalax813FlxFfmmoOC8kE30W6mP6iecP+ojm

xf577RhwR+PdE5zNNvm2F8j5ZWP39pboX7e3eYUCsEyPmVu1MSMTXrHHg6KNhCty

Qu1JfrAaisch+6vrAzfuP7t0WiILzieQgZzFDpI9HziwwOtCw+EKQhHCOPurWcO6

ByZUBzkCgYEA9aEprwqutbXB5H3QinxqXLInAH+wy8oTAMS6nV1sisIos6dD3CLO

u2fLRegv8PEUopASnzyv5PWU/iS+VJjdBCco59hmwW+7CVpaOJXlJ1qpznPVJmyx

pWsinM9Ug23GDd/jd61yKux22773RSGCYs9N7FVww5WYcDlWHLUFPk0CgYEA69DQ

h2iFuDSPonG8GPS6hf/KVRQaJZqGAINCk/2txTWmaz9VPdWT25+rxBzIoQOYAC4P

NjPHo/gJLrO/y6X6lAKBCje/Otb9E7GZwH0pFc7MxtQVR4ik6/7To3ancXNmawHe

owWZHDBRK+Ot33nZ+tYvAq48zE7rxNxsctZ9O1cCgYASsd12UR3S/q5vMZQ5thZy

T6zgQNe36v1fRZneeEnWlch7Q/PKQWvyn4e9Hlrnv7GOXeDM9dV9W6OnZCyIS8om

ksRuQO4xMsvNfm73d5ElWaUq7W3/qq4qpOjRfoY0Kpq0W6H4bd8OnUi+mN5BCLff

xV9s6WPXvv8HK5X+QVjQ0QKBgBrMqGY7IrdEge5cLpxHc8s2vq/ckPwlC4WTZUWc

VttKtZcKo41bcGpNQyAOhV6HIgcjNOdcCxw/XAvKsclbG5cmkbOvkjQFqs1KKccO

clTgI7WU9LYkeVm4pCS3n1/tVX5jwAGW6Uei1ha+0UvMdVFkdgM/+fjeHz1IL6r9

ZU4RAoGBALi33UjlJUYVMXPZc/JyFk8yyvRpYMRhmW7mQxR8gx0i1rNolPSccRkj

3NO+e1k86yyk3RsqBdixGKYDp2JqS+Aj7eHlxvUcrCAnpk9l96q8yuhQ4mJUWqs7

/hW6bxUPjDZ9BxprGZRL4ZLgPL+6C4Q4rE8TZu/5qQYDIy+ab03t

-----END RSA PRIVATE KEY-----

**Попередження**: ваш секретний ключ виглядатиме схожим чином, але не буде ідентичним. Це вкрай важливо. Якби OpenSSL постійно генерував однакові ключі, у нас були б серйозні проблеми.

**Генерування відкритого ключа**

А тепер згенеруймо із секретного ключа відкритий, і перевірмо його також. Тепер, маючи секретний ключ, вам потрібно згенерувати відкритий ключ, який іде до пари. Ви можете передати його будь-кому, хто хоче надіслати вам зашифровані дані. Якщо дані хешовано за допомогою вашого відкритого ключа, ніхто не зможе їх дешифрувати, не маючи відповідного секретного ключа. Щоб створити відкритий ключ на основі секретного, введіть команду нижче. Ви побачите такий результат:

openssl rsa -in private\_key.pem -outform PEM -pubout -out public\_key.pem

writing RSA key

Ви можете переглянути також і відкритий ключ. Він, як і секретний, виглядатиме, як набір випадкових символів, але ці символи будуть іншими та їх буде менше.

cat public\_key.pem

-----BEGIN PUBLIC KEY-----

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4kNMSmssCSYbOnq/UAHG

H5xx9gjZaOiST3JQQtJO11L/YeBO8DOHc7UawNADA/XDBAnGZih1M8T1PGc6Vk5S

W2Lb8FMf9zG2XhYpCACFFPJAW00qs4s1JesdugOprHZ8Jmm/QJl4KuCjlY/XdviC

vcbxROIQ2mglR8nW1QWrhECQNBfodRSuTwmW3qBSW/Xd5pmTpP4GHCyUfRO9YCF/

tZYtVMYg4FOqdGaTHRZbs6peMV4DlSjZHDonnsGK0UJpxQNbtJEcG7vr7Vl8ziVW

Y5RUDND7nZYlQlbqxvvqbPPt+px34pAZ58eyOqeAmYBc8mwNoXp4YrC2deFng7zr

KwIDAQAB

-----END PUBLIC KEY-----

**Попередження**: як і секретний ключ, ваш відкритий ключ виглядатиме інакше, ніж на зображенні.

Тепер, створивши обидва ключі, ви можете почати використовувати їх, щоб шифрувати й дешифрувати дані. Тож почнімо!

**Шифрування й дешифрування**

Ви зашифруєте файл за допомогою свого відкритого ключа, так ніби хтось інший зашифрував його й надіслав вам. Тоді ви (і лише ви) зможете дешифрувати файл за допомогою секретного ключа. Подібним чином ви можете шифрувати файли за допомогою відкритих ключів інших людей, знаючи, що ці люди зможуть дешифрувати їх.

Ви створите текстовий файл із певною інформацією, яку хочете захистити. Тоді ви зашифруєте й перевірите цей файл. Щоб створити його, введіть команду нижче. Ви створите новий текстовий файл під назвою secret.txt, який міститиме лише такий текст: This is a secret message, for authorized parties only. Якщо бажаєте, можете змінити повідомлення на будь-яке інше.

echo 'This is a secret message, for authorized parties only' > secret.txt

Тоді, щоб зашифрувати файл за допомогою відкритого ключа, введіть цю команду:

openssl rsautl -encrypt -pubin -inkey public\_key.pem -in secret.txt -out secret.enc

Так ви створите файл secret.enc, який буде зашифрованою версією файлу secret.txt. Зауважте, якщо ви спробуєте переглянути вміст зашифрованого файлу, він виглядатиме спотворено. Для зашифрованих повідомлень це цілком нормальна ситуація, оскільки їх вміст не призначено для візуального перегляду.

Наприклад, отак виглядатиме вміст файлу secret.enc, якщо переглянути його в редакторі Nano за допомогою команди нижче:

nano ~/secret.enc

Результат:

^? < e ^@vmD ^B% r\*M o^R ^O 8 X { ^\(^B ^}= 1i T 9~ ^RT^\^Px ^T^l n ^G ^O ^i iN (W [ ^$

^a^d~m , d Tq L < J ^Q bdQ

=Q R[^kT ^G iq GG ^T { UZ^dV8^A ^~O#koj^N^^ K vT ^O3 ^Tn^oP^l^Pa ^u3^G^N^i0=c{ ^tR09 o@^d$

<br>

<br>

<br>

<br>

<br>

<br>

<br>

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos

^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^\_ Go To Line

Щоб вийти з редактора Nano, скористайтеся командою **Ctrl-X**.

Тепер зашифрований файл можна надіслати людині, у якої є відповідний секретний ключ. Оскільки він є у вас, ви зможете дешифрувати файл і отримати початковий вміст. Пам’ятайте: оскільки файл зашифрували за допомогою відкритого ключа, для дешифрування потрібен секретний ключ. Дешифруйте файл за допомогою цієї команди:

openssl rsautl -decrypt -inkey private\_key.pem -in secret.enc

На екрані з’явиться вміст дешифрованого файлу, який має відповідати вмісту secret.txt:

This is a secret message, for authorized parties only

**Створення дайджесту**

Тепер ви створите дайджест повідомлення, а тоді – цифровий підпис цього дайджесту. Після цього ви підтвердите підпис дайджесту. Так ви запобіжите змінам і підробкам повідомлення. Якщо змінити повідомлення, зміниться його дайджест. Він відрізнятиметься від підписаного, і під час підтвердження станеться помилка.

Щоб створити дайджест повідомлення, введіть цю команду:

openssl dgst -sha256 -sign private\_key.pem -out secret.txt.sha256 secret.txt

За допомогою вашого секретного ключа утиліта створить файл під назвою secret.txt.sha256, що містить дайджест вашого таємного текстового файлу.

Тоді будь-хто зможе за допомогою вашого відкритого ключа й цього дайджесту підтвердити, що основний файл не зазнав змін, відколи ви створили й хешували його. Щоб виконати це підтвердження, введіть цю команду:

openssl dgst -sha256 -verify public\_key.pem -signature secret.txt.sha256 secret.txt

З’явиться поданий нижче результат, який вказуватиме на те, що підтвердження відбулось успішно, тобто жодна стороння особа не внесла змін у файл:

Verified OK

Якщо з’явиться якийсь інший результат, це означатиме, що вміст файлу змінили, і він, імовірно, більше не безпечний.

### **Практика хешування**

На цьому занятті ви попрактикуєтеся в хешуванні й підтвердженні хешів за допомогою інструментів md5sum і shasum.

*Md5sum* – це програма хешування, яка підраховує й підтверджує 128-бітні хеші MD5. Як і з будь-яким алгоритмом хешування, теоретично існує незліченна кількість файлів, які матимуть певний хеш MD5. Md5sum використовується для перевірки цілісності файлів.

Схожим чином працює *shasum* – програма шифрування, що підраховує й підтверджує хеші SHA. Її також широко використовують, щоб перевіряти цілісність файлів.

На цьому практичному занятті ви побачите, що майже будь-яка зміна у файлі спричинить зміни хешу MD5 або хешів SHA.

**Завдання**

* **Обчислення**: ви створите текстовий файл і згенеруєте хеші за допомогою інструментів md5sum і shasum.
* **Перевірка**: згенерувавши зведення хешів, ви перевірите створені файли.
* **Підтвердження**: ви підтвердите хеш за допомогою інструментів md5sum і shasum.
* **Зміна**: ви зміните свій текстовий файл і порівняєте ці результати з початковим хешем, щоб побачити, як змінюється зведення та як виникають помилки в процесі підтвердження хешу.

**MD5**

Ми почнемо зі створення текстового файлу, що містить якісь дані. Ви можете вносити власні текстові дані. Через цю команду можна створити текстовий файл під назвою file.txt з одним рядком простого тексту.

Ви побачите такий результат (або схожий):

echo 'This is some text in a file, just so we have some data' > file.txt

Тепер ви згенеруєте суму MD5 для цього файлу та збережете її. Щоб згенерувати суму для новоствореного файлу, введіть цю команду md5sum:

md5sum file.txt > file.txt.md5

Так ви створите хеш MD5 і збережете його в новий файл. Ви можете переглянути хеш, вивівши його вміст на екран за допомогою цієї команди:

cat file.txt.md5

Хеш, що з’явиться на терміналі, матиме приблизно такий вигляд:

c7a8ef893898f9a6b380eb4ec1e87113 file.txt

Крім того, ви також можете перевірити правильність хешу й відсутність змін (внесених після створення суми) у початковому файлі. Для цього введіть команду нижче. Указаний результат свідчитиме про те, що хеш дійсний.

md5sum -c file.txt.md5

file.txt: OK

**Перевірка недійсного файлу**

Далі ми продемонструємо безпеку цього процесу, показавши, як зміна навіть одного символу у файлі призводить до зміни хешу. Спочатку ви створите копію текстового файлу та вставите один пробіл у кінці цього файлу. Скористайтеся будь-яким зручним для вас текстовим редактором. Зауважте, що ми додали вказівки щодо того, як внести цю зміну в Nano. Щоб створити копію файлу, введіть цю команду:

cp file.txt badfile.txt

Тоді створіть нову суму md5sum для нового файлу:

md5sum badfile.txt > badfile.txt.md5

Зверніть увагу, що хеш нового файлу **ідентичний** хешу початкового файлу file.txt, хоча в них різні назви. Це тому, що під час хешування важливі дані, а не метадані файлу.

cat badfile.txt.md5

cat file.txt.md5

Щоб відкрити текстовий файл у Nano, введіть цю команду:

nano badfile.txt

Файл відкриється в текстовому редакторі. Щоб додати пробіл у кінці файлу, клавішами стрілок (не мишею) перемістіть курсор у кінець рядка тексту, а тоді натисніть пробіл. Ваш екран має бути схожим на це зображення:

This is some text in a file, just so we have some data

<br>

<br>

<br>

<br>

<br>

<br>

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos

^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^\_ Go To Line

Щоб зберегти файл, натисніть **Ctrl+X**. Має з’явитися таке повідомлення:

This is some text in a file, just so we have some data

<br>

<br>

<br>

<br>

<br>

<br>

Save modified buffer? (Answering "No" will DISCARD changes.)

Y Yes

N No ^C Cancel

Щоб підтвердити, наберіть **Y**, що означатиме **yes (так)**, а тоді натисніть **Enter**.

Ви повернетеся на звичайний екран термінала. Зробивши таку незначну зміну у файлі, спробуйте знову підтвердити хеш. Цього разу підтвердження має бути невдалим, а це означає, що будь-яка зміна призведе до зміни хешу. Спробуйте підтвердити хеш, знову ввівши цю команду:

md5sum -c badfile.txt.md5

Ви побачите повідомлення, що підтвердження не вдалося:

badfile.txt: FAILED

md5sum: WARNING: 1 computed checksum did NOT match

Щоб побачити, наскільки відрізняється хеш зміненого файлу, згенеруйте новий хеш і перевірте його:

md5sum badfile.txt > new.badfile.txt.md5

cat new.badfile.txt.md5

Перегляньте, як він відрізняється від попереднього хешу:

dcd879fd2c162dbfe9a186a67902e7ce badfile.txt

Для довідки, ось вміст початкової суми:

c7a8ef893898f9a6b380eb4ec1e87113 file.txt

**SHA**

Виконайте ті ж кроки, але з хешами SHA1 і SHA256 за допомогою інструмента shasum. З функціональної точки зору обидва хеші працюють доволі схожим чином, і в них однакове призначення. Але хеші SHA1 і SHA256 забезпечують сильніший захист, ніж MD5, а хеш SHA256 –надійніший, ніж SHA1. Це означає, що сторонньому зловмиснику простіше зламати систему, яка використовує хеш MD5, ніж ту, де застосовується SHA1. Оскільки хеш SHA256 є найнадійнішим із трьох, наразі він дуже широковживаний.

**SHA1**

Щоб створити суму SHA1 і зберегти її у файлі, скористайтеся цією командою:

shasum file.txt > file.txt.sha1

Перегляньте суму, вивіши її на екран, як ви це вже робили:

cat file.txt.sha1

65639a89992784291d769e05338085d1739645c6 file.txt

Тепер підтвердьте хеш за допомогою команди нижче. (Як і в попередньому випадку, якби в початковому файлі були зміни, перевірка була б невдалою.)

shasum -c file.txt.sha1

и побачите такий результат, що свідчитиме про успішне підтвердження:

file.txt: OK

Щоб підтвердити виконане завдання, натисніть Check my progress (Перевірити мій прогрес).

**SHA256**

За допомогою того ж інструмента можна створити суму SHA256. Параметр -a визначає, який алгоритм потрібно використати, і за умовчанням вибирає SHA1, якщо нічого не вказано. Щоб створити суму SHA256 для файлу, скористайтеся цією командою:

shasum -a 256 file.txt > file.txt.sha256

Ви можете вивести вміст цього файлу, як і раніше:

cat file.txt.sha256

Причина посиленого захисту SHA256 у тому, що в цій системі створюється довший хеш, який важче вгадати. Ви можете переконатися, що вміст файлу тут набагато довший, ніж у файлі SHA1:

7a54af37c15a82e157c8368324e7234d22778ce845219cd16172895a608030ff file.txt

Насамкінець, щоб підтвердити суму SHA256, можна скористатися тією ж командою, що й раніше:

shasum -c file.txt.sha256