Міністерство освіти і науки України
Національний Технічний Університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

Комп'ютерний практикум 4
з дисципліни
«Проектування, розробка і реалізація криптографічних систем»
Тема:
«Дослідження систем обміну повідомленнями та IP-телефонії»

Виконав:
Студент групи ФІ-22мн
Кушнір Олександр
Перевірив:
Селюх П. В.

Київ – 2023

Viber

Під час встановлення кожен основний пристрій Viber генерує одну 256-бітну пару ключів Curve-25519 під назвою ID Key. Приватна частина ключа зберігається на пристрої, а публічна частина ключа завантажується на сервери Viber. Додаткові пристрої (ПК і планшети) отримують закритий ключ від основного пристрою безпечним методом, описаним у розділі «Реєстрація вторинного пристрою» нижче. Крім того, кожен клієнт Viber генерує серію попередніх ключів. Кожен PreKey має дві 256-бітні пари ключів Curve-25519, які називаються Handshake Key і Ratchet Key. Усі закриті ключі зберігаються на пристрої, тоді як відкриті ключі завантажуються на сервер. Сервер зберігає кілька ключів на пристрій і запитує більше, коли вони падають нижче встановленого порогу. Кожен запит змушує клієнт генерувати додаткові ключі та завантажувати загальнодоступні частини на сервер.

Сеанс потрібно встановлювати лише один раз між кожними двома пристроями Viber, які бажають безпечно спілкуватися. Коли сеанс існує, його можна використовувати для надсилання необмеженої кількості повідомлень у будь-якому напрямку. Щоб надіслати захищене повідомлення, потрібні захищені сесії між пристроєм, що надсилає, і всіма пристроями одержувача, а також між пристроєм, що надсилає, та всіма іншими пристроями відправника. Так, наприклад, якщо користувач A, який має мобільний телефон і ПК, зареєстровані у Viber під одним обліковим записом, бажає спілкуватися з користувачем B, який має мобільний телефон і ПК, між кожною парою пристроїв необхідно встановити безпечні сесії. Сеанси між пристроями одного облікового запису встановлюються, коли пристрої реєструються. Між будь-якими двома пристроями потрібен лише один сеанс, і цей сеанс можна використовувати для синхронізації будь-якої кількості розмов з іншими обліковими записами Viber.

Щоб встановити сеанс з іншим обліковим записом, пристрій («Аліса»), який бажає встановити сеанс з одноранговим користувачем («Боб»), надсилає запит на сервер Viber із номером телефону одержувача. Сервер відповідає відкритим ідентифікаційним ключем однорангового пристрою та серією однорангових відкритих попередніх ключів, по одному на кожен пристрій, зареєстрований в обліковому записі «Боба». Пристроям «Боба» не потрібно бути онлайн, коли це станеться.

Пристрій, який надсилає повідомлення цільовому користувачеві, повинен шифрувати це повідомлення для кожного сеансу з кожним пристроєм цільового користувача. Для цього генерується ефемерний одноразовий 128-бітний симетричний ключ, який використовується для шифрування тіла повідомлення за допомогою алгоритму шифрування Salsa20. Потім цей ефемерний ключ повідомлення шифрується за допомогою сеансового ключа кожного одержувача. Пристрій-відправник

надсилає уніфіковане повідомлення на сервер, що містить один зашифрований шифртекст і набір зашифрованих ефемерних ключів. Розгортання на стороні сервера розрізає це повідомлення та доставляє відповідні частини до кожного цільового пристрою.

WhatsApp

Щоб користувачі WhatsApp могли спілкуватися один з одним безпечно та приватно, клієнт-відправник встановлює попарно зашифрований сеанс з кожним із пристроїв одержувача. Крім того, клієнт відправника встановлює попарно зашифрований сеанс з усіма іншими пристроями, пов’язаними з обліковим записом відправника.

Після встановлення цих попарно зашифрованих сеансів клієнтам не потрібно відновлювати нові сеанси з цими пристроями, якщо стан сеансу не втрачено, що може бути спричинено такою подією, як перевстановлення програми або зміна пристрою.

WhatsApp використовує цей підхід «клієнт-фанаут» для передачі повідомлень на кілька пристроїв, де клієнт WhatsApp передає одне повідомлення N кількість разів до N кількості різних пристроїв. Кожне повідомлення окремо шифрується за допомогою встановленого сеансу попарного шифрування з кожним пристроєм.

Щоб встановити сеанс:

1. Клієнт-ініціатор («ініціатор») запитує публічний ідентифікаційний ключ, публічний підписаний попередній ключ і один загальнодоступний одноразовий попередній ключ для кожного пристрою одержувача та кожного додаткового пристрою користувача-ініціатора (за винятком ініціатора).

2. Сервер повертає запитані значення відкритого ключа. Одноразовий попередній ключ використовується лише один раз, тому після запиту він видаляється зі сховища сервера. Якщо остання партія одноразових попередніх ключів одержувача була використана, а одержувач не поповнив їх, одноразовий попередній ключ не повертається. Крім того, для кожного пристрою-супутника (як для облікового запису ініціатора, так і для одержувача) сервер також повертає метадані зв’язування (Lmetadata), підпис облікового запису (Asignature) і підпис пристрою (Dsignature), які було завантажено пристроєм-супутником під час підключення.

3. Для кожного поверненого ключа, встановленого для супутнього пристрою, ініціатор повинен перевірити Asignature за CURVE25519_VERIFY_SIGNATURE(Iprimary, 0x0600 || Lmetadata || Icompanion) і Dsignature за CURVE25519_VERIFY_SIGNATURE(Icompanion, 0x0601 || Lmetadata || Icompanion || Iprimary) . Якщо будь-яка перевірка пристрою-супутника завершується невдачею, ініціатор негайно припиняє процес створення

сеансу шифрування та не надсилатиме жодних повідомлень на цей пристрій.

Після отримання ключів із сервера та перевірки ідентифікації кожного пристрою ініціатор починає встановлювати сеанс шифрування з кожним окремим пристроєм:

1. Ініціатор зберігає ідентифікаційний ключ одержувача як Irecipient, the Підписаний попередній ключ як отримувач, а одноразовий попередній ключ як Orecipient.

2. Ініціатор генерує ефемерну пару ключів Curve25519, Einitiator.

3. Ініціатор завантажує свій власний ідентифікаційний ключ як Ініціатор.

4. Ініціатор обчислює головний секрет як master_secret = ECDH(Iinitiator, Orecipient) || ECDH (Iinitiator, Orecipient) || ECDH (Iinitiator, Orecipient) || ECDH (Iinitiator, Orecipient). Якщо там не є одноразовим попереднім ключем, остаточний ECDH опущено.

5. Ініціатор використовує HKDF для створення кореневого ключа та ланцюжкових ключів master_secret.

Skype

Skype зберігає реєстраційну інформацію як на комп'ютері абонента, так і на сервері Skype. Skype використовує цю інформацію для автентифікації одержувачів дзвінків і гарантує, що абоненти, які бажають автентифікації, мають доступ до сервера Skype, а не до самозванця. Skype каже, що для цього використовує шифрування з відкритим ключем, як визначено RSA.

Сервер Skype має закритий ключ і розповсюджує відкритий аналог цього ключа з кожною копією програмного забезпечення. Під час реєстрації користувача користувач вибирає бажане ім'я користувача та пароль. Skype локально генерує відкритий і закритий ключі. Закритий ключ і хеш пароля зберігаються на комп'ютері користувача.

Потім із сервером Skype встановлюється сеанс із 256-бітним шифруванням AES. Клієнт створює ключ сеансу за допомогою свого генератора випадкових чисел.

Сервер Skype перевіряє, чи вибране ім'я користувача є унікальним і відповідає правилам іменування Skype. Сервер зберігає ім'я користувача та хеш пароля користувача у своїй базі даних.

Тепер сервер формує та підписує сертифікат ідентифікації для імені користувача, який пов'язує ім'я користувача, ключ перевірки та ідентифікатор ключа.

Для кожного виклику Skype створює сеанс із 256-бітним ключем сеансу. Цей сеанс існує доти, доки триває спілкування, і протягом фіксованого часу після цього. Skype безпечно передає сеансовий ключ

одержувачу дзвінка під час підключення дзвінка. Потім цей сеансовий ключ використовується для шифрування повідомлень в обох напрямках.

Весь трафік у сеансі шифрується за допомогою алгоритму AES, що працює в режимі цілочисельного лічильника (ICM). Skype шифрує поточний лічильник і сіль за допомогою ключа сеансу за допомогою 256-бітного алгоритму AES. Цей алгоритм повертає потік ключів, а потім обробляє XOR із вмістом повідомлення. Сесії Skype містять кілька потоків. Лічильник ICM залежить від потоку та розташування в потоці.

Skype використовує випадкові числа для кількох криптографічних цілей. Цілі включають захист від атак на відтворення, створення пар ключів RSA та створення половин ключів AES для шифрування вмісту. Безпека однорангового сеансу Skype значною мірою залежить від якості випадкових чисел, згенерованих обома кінцями сеансу Skype. Генерація випадкових чисел залежить від операційної системи.

Skype використовує стандартні криптографічні примітиви для досягнення своїх цілей безпеки. Криптографічні примітиви, які використовуються в Skype, це блоковий шифр AES, криптосистема з відкритим ключем RSA, схема заповнення підпису ISO 9796-2, хеш-функція SHA-1 і потоковий шифр RC4.

Ключове узгодження досягається за допомогою власного симетричного протоколу. Щоб захиститися від атаки відтворення, однорангові комп’ютери кидають виклик один одному випадковими 64-розрядними нонсами. Відповідь на виклик полягає в тому, щоб налаштувати виклик у приватний спосіб і повернути його, підписаного особистим ключем відповідача.

Однорангові вузли обмінюються сертифікатами ідентифікації та підтверджують, що ці сертифікати є законними. Оскільки сертифікат ідентифікації містить відкритий ключ, кожна сторона може підтверджувати підписи, створені іншим вузлом. Кожен вузол додає 128 випадкових бітів до 256-бітового ключа сеансу.

Telegram

Ключі генеруються за допомогою протоколу Діффі-Хеллмана.

Давайте розглянемо наступний сценарій: користувач A хоче ініціювати наскрізне зашифроване спілкування з користувачем B. Користувач A виконує messages.getDhConfig, щоб отримати параметри Діффі-Хеллмана: простий p і старший елемент g. Виконання цього методу перед кожною новою процедурою генерації ключа є життєво важливим. Має сенс кешувати значення параметрів разом із версією, щоб уникнути необхідності кожного разу отримувати всі значення. Якщо версія, збережена на клієнті, все ще актуальна, сервер поверне конструктор messages.dhConfigNotModified.

Очікується, що клієнт перевірить, чи є p безпечним 2048-бітним простим числом (це означає, що і p, і (p-1)/2 є простими, і що 22047 < p < 22048), і що g генерує циклічну підгрупу простого порядку (p-1)/2, тобто є квадратичним залишком за модулем p. Оскільки g завжди дорівнює 2, 3, 4, 5, 6 або 7, це легко зробити за допомогою квадратичного закону взаємності, що дає просту умову на p mod 4g, а саме: p mod 8 = 7 для g = 2; p mod 3 = 2 для g = 3; відсутність додаткової умови для g = 4; p mod 5 = 1 або 4 для g = 5; p mod 24 = 19 або 23 для g = 6; і p mod 7 = 3, 5 або 6 для g = 7. Після того, як g і p були перевірені клієнтом, має сенс кешувати результат, щоб уникнути повторення тривалих обчислень у майбутньому. Цей кеш може використовуватися для генерації ключа авторизації.

Якщо клієнту потрібна додаткова ентропія для генератора випадкових чисел, він може передати параметр random_length (random_length> 0), щоб сервер генерував власну випадкову послідовність відповідної довжини.

Важливо: використання випадкової послідовності сервера в необробленому вигляді може бути небезпечним, її потрібно поєднувати з послідовністю клієнта.

Клієнт A обчислює 2048-бітове число a (використовуючи достатню ентропію або випадковий параметр сервера; див. вище) і виконує messages.requestEncryption після передачі g_a := pow(g, a) mod dh_prime.

Користувач B отримує оновлення updateEncryption для всіх пов’язаних ключів авторизації (усіх авторизованих пристроїв) за допомогою конструктора чату encryptedChatRequested. Користувачеві має бути показана основна інформація про користувача A та запропоновано прийняти або відхилити запит.

Обидва клієнти мають перевірити, що g, g_a та g_b більші за одиницю та менші за p-1. Ми також рекомендуємо перевірити, чи g_a і g_b знаходяться між $2^{2048-64}$ і $p - 2^{2048-64}$.

Після того як користувач B підтвердить створення секретного чату з A в інтерфейсі клієнта, клієнт B також отримує оновлені параметри конфігурації для методу Діффі-Хеллмана. Після цього він генерує випадкове 2048-бітове число, b, використовуючи правила, подібні до правил для a.

Отримавши g_a від оновлення з encryptedChatRequested, він може негайно згенерувати остаточний спільний ключ: key = (pow(g_a, b) mod dh_prime). Якщо довжина ключа < 256 байт, додайте кілька початкових нульових байтів як заповнення, щоб ключ мав рівно 256 байт. Його відбиток, key_fingerprint, дорівнює останнім 64 бітам SHA1 (ключа).

Примітка 1: у цьому конкретному випадку SHA1 використовується навіть для секретних чатів MTProto 2.0.

Примітка 2: цей відбиток використовується як перевірка працездатності для процедури обміну ключами для виявлення помилок під час розробки

клієнтського програмного забезпечення — він не пов’язаний із візуалізацією ключів, що використовується на клієнтах як засіб зовнішньої автентифікації в секретних чатах. Ключові візуалізації на клієнтах створюються з використанням перших 128 біт SHA1 (початковий ключ), а потім перших 160 біт SHA256 (ключ, який використовувався під час оновлення секретного чату до рівня 46).

Клієнт B виконує messages.acceptEncryption після передачі йому g_b := pow(g, b) mod dh_prime та key_fingerprint.

Для всіх авторизованих пристроїв клієнта Б, крім поточного, оновлення updateEncryption надсилаються з конструктором encryptedChatDiscarded. Після цього єдиним пристроєм, який матиме доступ до секретного чату, є пристрій B, який здійснив виклик до messages.acceptEncryption.

Користувачеві A буде надіслано оновлення updateEncryption з конструктором encryptedChat для ключа авторизації, який ініціював чат.

За допомогою g_b з оновлення клієнт A також може обчислити ключ спільного ключа = (pow(g_b, a) mod dh_prime). Якщо довжина ключа < 256 байт, додайте кілька початкових нульових байтів як заповнення, щоб ключ мав рівно 256 байт. Якщо відбиток отриманого ключа ідентичний тому, який було передано encryptedChat, вхідні повідомлення можна надсилати та обробляти. В іншому випадку необхідно виконати messages.discardEncryption і повідомити користувача.

Щоб зберегти минуле спілкування в безпеці, офіційні клієнти Telegram ініціюють повторне введення ключа, коли ключ було використано для дешифрування та шифрування понад 100 повідомлень або він використовувався більше одного тижня, якщо ключ використовувався для шифрування. хоча б одне повідомлення. Потім старі ключі надійно викидаються, і їх неможливо відновити, навіть якщо є доступ до нових ключів, які зараз використовуються.

Зауважте, що ваш клієнт має підтримувати Forward Secrecy у секретних чатах, щоб бути сумісним з офіційними клієнтами Telegram.

Надсилання та отримання повідомлень у секретному чаті

Серіалізація та шифрування вихідних повідомлень

Створюється об’єкт TL типу DecryptedMessage, який містить повідомлення у вигляді звичайного тексту. Для зворотної сумісності об’єкт має бути загорнутий у конструктор decryptedMessageLayer із зазначенням підтримуваного рівня (починаючи з 46).

Отримана конструкція серіалізується як масив байтів за допомогою загальних правил TL. До результуючого масиву додається 4 байти, що містять довжину масиву без урахування цих 4 байтів.

Масив байтів доповнюється від 12 до 1024 випадкових байтів доповнення, щоб його довжина ділилася на 16 байтів. (У старішому

шифруванні MTProto 1.0 використовувалося лише від 0 до 15 байтів заповнення.)

Ключ повідомлення, msg_key, обчислюється як 128 середніх бітів SHA256 даних, отриманих на попередньому кроці, до яких додаються 32 байти ключа спільного ключа. (Для старішого шифрування MTProto 1.0 msg_key обчислювався інакше, як 128 молодших бітів SHA1 даних, отриманих на попередніх кроках, за винятком байтів заповнення.)

Для MTProto 2.0 ключ AES aes_key і вектор ініціалізації aes_iv обчислюються (ключ — спільний ключ, отриманий під час генерації ключа) таким чином:

msg_key_large = SHA256 (substr (ключ, 88+x, 32) + відкритий текст + випадкове_доповнення);

msg_key = substr (msg_key_large, 8, 16);

sha256_a = SHA256 (msg_key + substr (ключ, x, 36));

sha256_b = SHA256 (substr (ключ, 40+x, 36) + msg_key);

aes_key = substr (sha256_a, 0, 8) + substr (sha256_b, 8, 16) + substr (sha256_a, 24, 8);

aes_iv = substr (sha256_b, 0, 8) + substr (sha256_a, 8, 16) + substr (sha256_b, 24, 8);

Для MTProto 2.0 x=0 для повідомлень від автора секретного чату, x=8 для повідомлень у протилежному напрямку.

Для застарілої версії MTProto 1.0 msg_key, aes_key і aes_iv обчислювалися інакше (див. цей документ для довідки).

Дані шифруються за допомогою 256-бітного ключа aes_key і 256-бітного вектора ініціалізації aes-iv за допомогою шифрування AES-256 із нескінченним розширенням спотворення (IGE). Відбиток ключа шифрування key_fingerprint і ключ повідомлення msg_key додаються у верхній частині результуючого масиву байтів.

Зашифровані дані вбудовуються в виклик API messages.sendEncrypted і передаються на сервер Telegram для доставки іншій стороні секретного чату.