МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Проектування, розробка і реалізація криптографічних систем

“Дослідження систем обміну повідомленнями та IP-телефонії”

Виконали:

Студенти групи ФІ-22мн
Бондаренко Андрій
Яценко Артем

Київ – 2024

Мета: дослідження особливостей реалізації криптографічних механізмів системобміну повідомленнями та IP-телефонії.

Хід роботи

Viber

Термінологія

Viber-акаунт – це набір пристроїв, зареєстрованих у Viber на один телефонний номер. Акаунт складається з одного основного пристрою та (необов'язково) необмеженої кількості додаткових пристроїв. Повідомлення, відправлені або отримані будь-яким пристроєм, відображаються на всіх інших зареєстрованих пристроях цього акаунту, починаючи з моменту їх реєстрації (минулу історію переглянути не можна).

Основний пристрій – це мобільний телефон на iOS, Android чи Windows Phone, зареєстрований у сервісі Viber за допомогою телефонного номера, що обслуговується мобільним оператором.

Додатковий пристрій – зазвичай це iPad, планшет або настільний комп'ютер, який має бути зв'язаний з основним пристроєм.

ID-ключ – це довгострокова пара 256-бітних ключів Curve25519, які використовуються для ідентифікації Viber-акаунту. ID-ключ генерується лише основним пристроєм.

PreKeys – це набір середньострокових пар ключів Curve25519, які використовуються для встановлення безпечних сесій між пристроями. PreKeys генеруються окремо кожним пристроєм акаунту.

Сесія – це двосторонній безпечний віртуальний зв'язок між двома конкретними пристроями. Безпечні сесії автоматично встановлюються між усіма пристроями одного акаунту та між будь-якими двома пристроями різних акаунтів, які обмінюються повідомленнями. Безпечна сесія з іншим акаунтом у Viber позначається сірим замком на екрані розмови.

Довіра – це стан між двома акаунтами, який вказує на те, що акаунт-співрозмовник аутентифікований і що ніякий зловмисник не втрутився в сесію, встановлену між будь-якими двома пристроями акаунту. Користувачі можуть встановити довіру, слідуючи описаній нижче процедурі аутентифікації. Довірча сесія у Viber позначається зеленим замком.

Порушення довіри – це стан між двома акаунтами, який настає, коли ID-ключ довіреного співрозмовника змінився після встановлення довіри. Це може статися законним шляхом, якщо користувач змінив свій основний пристрій, перевстановив Viber або якщо ключі були втрачені через збій в системі зберігання. Однак це також може відбуватися, якщо хтось намагається перехопити розмову. Сесія з порушенням довіри у Viber позначається червоним замком.

Підготовка налаштування сесії

Під час встановлення кожен основний пристрій Viber генерує одну 256-бітну пару ключів Curve-25519, що називається ідентифікаційним ключем. Приватна частина ключа зберігається на пристрої, тоді як публічна частина ключа завантажується на сервери Viber. Додаткові пристрої (ПК та планшети) отримують приватний ключ від основного пристрою через безпечний метод, описаний у розділі "Реєстрація додаткового пристрою", нижче.

Крім того, кожен клієнт Viber генерує серію PreKeys. Кожен PreKey має дві 256-бітні пари ключів Curve-25519, що називаються ключем рукостискання та ключем Rachet. Приватні ключі зберігаються на пристрої, тоді як публічні ключі завантажуються на сервер. Сервер тримає кілька ключів на пристрій і запитує більше, коли вони опускаються нижче конфігурованого порога. Кожен запит змушує клієнта генерувати додаткові ключі та завантажувати публічні частини на сервер.

Установка безпечного сеансу

Сеанс потрібно встановити лише один раз між кожними двома пристроями Viber, які бажають безпечно спілкуватися. Як тільки сеанс створено, його можна використовувати для надсилання необмеженої кількості повідомлень в обох напрямках.

Для надсилання безпечного повідомлення між пристроєм відправника та всіма пристроями отримувача, а також між пристроєм відправника та всіма іншими пристроями відправника, повинні існувати безпечні сесії.
Так, наприклад, якщо користувач А, який має мобільний телефон і ПК, зареєстрований у Viber на одному обліковому записі, бажає спілкуватися з користувачем Б, який має мобільний телефон і ПК, необхідно встановити безпечні сесії між кожною парою пристроїв.

Сеанси між пристроями одного облікового запису встановлюються при реєстрації пристроїв. Для будь-яких двох пристроїв потрібна лише одна сесія, яку можна використовувати для синхронізації будь-якої кількості розмов з іншими обліковими записами Viber.

Для встановлення сесії з іншим обліковим записом пристрій («Аліса») надсилає запит на сервер Viber з номером телефону отримувача. Сервер відповідає публічним ID ключем отримувача та серією публічних PreKeys, по одному на кожен пристрій, зареєстрований на обліковому записі «Боба». Присутність пристроїв «Боба» в мережі не вимагається в цей момент.

Потім пристрій «Аліси» генерує дві пари ключів Curve-25519 на 256 біт як власні ключі рукостискання та ретчетингу і отримує Root Key наступним чином:

RootKey = SHA256(DH(IDAlice, HSBob) || DH(HSAlice, IDBob) || DH(HSAlice, HSBob))
Далі, використовуючи RootKey, генерується ключ сесії:
TempKey = HMAC_SHA256(RootKey, DH (RatchetAlice, RatchetBob))
New RootKey = HMAC_SHA256(TempKey, "root")
SessionKey = HMAC_SHA256(TempKey, "mesg")
DH означає використання алгоритму обміну ключами Elliptic-Curve Diffie-Hellman. HS означає ключ рукостискання.

Різні рядки, передані функціям HMAC, забезпечують те, що навіть якщо ключ сесії буде скомпрометовано, вихідний ключ не можна буде відновити. «Аліса» потім надсилає «Бобу» повідомлення про початок сесії, що містить її власний публічний ID, ідентифікатор Pre-Key «Боба», який

використовувався для цієї сесії, а також власні публічні ключі рукостискання та ретчетингу. Коли «Боб» виходить у мережу та отримує це повідомлення, він може відтворити ті ж Root і Session ключі, використовуючи ту саму процедуру DH.

Варто зазначити, що якщо сесія встановлена з основним пристроєм співрозмовника, але не може бути встановлена з додатковим пристроєм (наприклад, через відсутність PreKeys на сервері чи офлайн статус), розмова все одно буде захищена шифруванням з кінця в кінець, але цей додатковий пристрій не зможе розшифрувати ці повідомлення і, отже, не зможе їх відобразити.

Обмін повідомленнями

Пристрій, що надсилає повідомлення цільовому користувачеві, повинен шифрувати його для кожної сесії з кожним пристроєм цього користувача. Для цього генерується ефемерний одноразовий 128-бітний симетричний ключ, який використовується для шифрування тексту повідомлення за допомогою алгоритму шифрування Salsa20. Цей ефемерний ключ повідомлення потім шифрується за допомогою кожного сесійного ключа одержувача. Пристрій-відправник відсилає на сервер єдине повідомлення, що містить один зашифрований текст і набір зашифрованих ефемерних ключів. Сервер розподіляє це повідомлення та доставляє відповідні частини кожному цільовому пристрою.

Два пристрої по черзі змінюють сесійні ключі в процесі, відомому як "ретчінг". Кожного разу, коли змінюється напрямок розмови, пристрій, якому належить черга, випадково генерує нову пару ключів ретчінга та знову виконує наступну послідовність: TempKey = HMAC_SHA256(RootKey, DH(RatchetAlice, RatchetBob)) Новий RootKey = HMAC_SHA256(TempKey, "root") SessionKey = HMAC_SHA256(TempKey, "mesg") з Ratchetthis_device, який є приватною частиною новоствореної пари ключів. Разом з кожним повідомленням також відправляється публічна частина Ratchetthis_device. Одержувач проводить DH зі своїм останнім приватним ретчінгом разом з публічним ретчінгом відправника. Цей подвійний ретчінг досягає двох речей: по-перше, безперервний ретчінг забезпечує передню та задню конфіденційність, так що навіть якщо ключі скомпрометовані, минулі та майбутні повідомлення не можуть

бути розшифровані. По-друге, алгоритм підтримує автентифікацію піра, оскільки ланцюг DH основних ключів починається з ID ключів обох пристроїв. Якщо ID ключ піра довіряється в будь-який момент, то весь ланцюг можна вважати надійним.

Зашифровані дзвінки

Кожна сторона дзвінка генерує ефемерну 256-бітну пару ключів Curve25519. Публічна частина підписується приватним ID ключем пристрою та обмінюється між двома пристроями під час фази налаштування дзвінка. Інша сторона автентифікує запит за допомогою публічного ID ключа піра.

Кожен пристрій перевіряє підпис та виконує обчислення DH для отримання одноразового сесійного ключа. Сесійний ключ дійсний тільки протягом тривалості цього конкретного дзвінка і не зберігається в нелетючій пам'яті.

Потік RTP аудіо або аудіо/відеодзвінка конвертується у SRTP та шифрується за допомогою алгоритму Salsa20, використовуючи сесійний ключ.

Передача фото, відео та файлів

Клієнт-відправник генерує ефемерний симетричний ключ Salsa20 і шифрує файл. Зашифрований файл разом з підписом HMAC завантажується на сервери Viber. Додатковий підпис MD5 розраховується над зашифрованими даними та відправляється разом з файлом, щоб надати серверу зберігання простий спосіб перевірки цілісності передачі, незалежно від шифрування від кінця до кінця.

Відправник потім відсилає Viber повідомлення одержувачу з ID завантаженого файлу та ключем шифрування. Це повідомлення зашифроване від кінця до кінця, використовуючи зашифровану сесію між відправником та одержувачем.

Одержувач генерує посилання для завантаження з ID файлу, завантажує зашифрований файл та розшифровує його, використовуючи ключ.

Безпечні групи

Всі учасники безпечної групи діляться спільним секретним ключем (симетричним ключем шифрування Salsa20), який не відомий Viber або третім сторонам.

Для нових груп цей спільний секрет генерується творцем групи та відправляється всім учасникам, використовуючи зазначені вище безпечні сесії один на один. Для не безпечних груп, що були створені з попередніми версіями Viber, цей секрет генерує перший учасник, який відправляє повідомлення до групового чату після того, як всі учасники групи оновилися до безпечної версії. У додатку Viber будь-який учасник групи може додати додаткових учасників до групи. Ці учасники отримають секрет від учасника групи, що їх додав.

Секрет групи просувається вперед за допомогою HMAC-SHA256 з кожним відправленим повідомленням. Кожне групове повідомлення містить номер послідовності, який вказує на кількість разів, коли була викликана хеш-функція. Різні клієнти завжди продовжують з того місця, де закінчилося останнє повідомлення, і продовжують ланцюг хешування з цієї точки, тому ключі не повторюються. Передня конфіденційність підтримується за допомогою одностороннього алгоритму хешування; навіть якщо ключ скомпрометований, минулі розмови не можуть бути розшифровані. Минулі ключі відкидаються клієнтом і не зберігаються у нелетючій пам'яті, за винятком короткого вікна минулих хешів, яке використовується в умовах гонки, коли два або більше учасників одночасно пишуть до групи.

Реєстрація додаткових пристоїв

Одна з ключових особливостей екосистеми Viber — це концепція додаткових пристроїв. Додатковий пристрій — це комп'ютер, iPad або планшет, який синхронізується з мобільним телефоном користувача та відображає історію обміну повідомленнями, включаючи вхідні та вихідні повідомлення.
Шифрування від кінця до кінця на додаткових пристроях в Viber працює наступним чином:

Шифрування здійснюється окремо для кожного пристрою. Якщо користувач А відправляє повідомлення користувачу Б, який має два пристрої, то користувачу А потрібно здійснити окремі сесії шифрування від кінця до кінця з обома пристроями та зашифрувати дані двічі, кожен раз використовуючи різний набір ключів. Аутентифікація здійснюється лише один раз для всього облікового запису. Якщо користувач А довіряє користувачу Б, довіра автоматично застосовується до всіх пристроїв користувача Б, а не лише до одного.

Аутентифікація здійснюється шляхом обміну приватним ID-ключем між усіма пристроями одного облікового запису. ID-ключ генерується лише основним пристроєм та передається на додаткові пристрої під час реєстрації за допомогою безпечного методу, який виконується наступним чином:
Додатковий пристрій генерує ефемерний 256-бітний ключовий набір Curve-25519.

Пристрій тоді генерує QR-код, який містить "Viber UDID" пристрою (публічно доступний унікальний ідентифікатор пристрою, створений Viber) плюс публічну частину ефемерного ключового набору.

Користувач сканує QR-код своїм основним пристроєм. Основний пристрій також генерує 256-бітний ключовий набір Curve-25519 та виконує DH-обчислення з публічним ключем з QR. Результат хешується за допомогою SHA256 для створення спільного секрету.

Основний пристрій потім шифрує свій власний приватний ID-ключ цим секретом та відправляє його разом із публічною частиною ефемерного ключа через сервери Viber до цільового пристрою, ідентифікованого за його UDID, як прочитано з QR-коду. Шифртекст підписується за допомогою HMAC-SHA256.

Додатковий пристрій отримує повідомлення, виконує ті ж DH-обчислення та хеш, щоб отримати той самий секрет, та використовує його для розшифрування приватного ID-ключа основного пристрою.
ID-ключ є частиною DH-ланцюга, що створює спільні секрети для сесій один на один. Тому без коректного ID додатковий пристрій не може брати

участі у жодних захищених розмовах, у яких бере участь основний пристрій.

Аутентифікація

Аутентифікація забезпечує засіб ідентифікації, що жоден "людина посередник" не компрометує безпеку шифрування від кінця до кінця. У додатку Viber аутентифікація виконується в контексті аудіо- (або аудіо/відео) дзвінка Viber.

Під час дзвінка кожен користувач може натиснути на екран блокування, щоб побачити числовий рядок, який обчислюється так:

Обидва пристрої виконують DH-обчислення, використовуючи свій власний приватний ID-ключ та публічний ID-ключ однолітка, опублікований під час фази налаштування дзвінка.

Результат DH хешується за допомогою SHA256 і обрізається до 160 біт. Ці 160 бітів перетворюються на рядок з 48 десяткових символів (0-9). Обидві сторони повинні бачити один і той же рядок чисел і можуть порівняти їх, прочитавши їх учаснику дзвінка. Якщо обидві сторони чують, що інша сторона читає вголос той самий рядок чисел, який вони бачать на своєму екрані, це дає дуже високий ступінь впевненості в тому, що ID-ключі не були змінені та що в цій розмові немає "людини посередника".

Перевірка ID-ключа захищає як захищені дзвінки, так і захищені сесії 1-1. У дзвінках ID-ключ використовується для підпису обміну ключами DH. У 1-1 чатах ID-ключ функціонує як корінь ланцюга DH, що веде до генерації спільного секрету. В свою чергу, сесія 1-1 захищає групові сесії, оскільки групові ключі обмінюються протягом сесій 1-1.

WhatsApp

WhatsApp Messenger дозволяє людям обмінюватися повідомленнями (включаючи чати, групові чати, зображення, відео, голосові повідомлення та файли), ділитися статусами та здійснювати дзвінки через WhatsApp по всьому світу. Повідомлення WhatsApp, голосові та відеодзвінки між

відправником і одержувачем, які використовують клієнтське програмне забезпечення WhatsApp, використовують протокол Signal.

Протокол Signal, розроблений Open Whisper Systems, є основою для наскрізного шифрування WhatsApp. Цей протокол наскрізного шифрування призначений для запобігання доступу третіх осіб і WhatsApp до відкритого тексту повідомлень або дзвінків. Через ефемерну природу криптографічних ключів, навіть у ситуації, коли поточні ключі шифрування з пристрою користувача фізично скомпрометовані, вони не можуть бути використані для розшифрування раніше переданих повідомлень.

Користувач може мати кілька пристроїв, кожен з яких має власний набір ключів шифрування. Якщо ключі шифрування одного пристрою зламані, зловмисник не зможе використати їх для розшифрування повідомлень, надісланих на інші пристрої, навіть ті, що зареєстровані на того самого користувача. WhatsApp також використовує наскрізне шифрування для шифрування історії повідомлень, що передаються між пристроями, коли користувач реєструє новий пристрій.

Терміни

Основний пристрій - це пристрій, який використовується для реєстрації облікового запису WhatsApp за допомогою номера телефону. Кожен обліковий запис WhatsApp асоціюється з одним основним пристроєм. Цей основний пристрій може використовуватися для зв'язування додаткових супутніх пристроїв з обліковим записом. Підтримувані платформи основного пристрою включають Android та iPhone.
Додатковий пристрій - пристрій, який підключається до існуючого облікового запису WhatsApp через основний пристрій. Платформи основних пристроїв, такі як iPhone та Android, не підтримують можливість підключення як супутній пристрій.
Пара ключів ідентифікації – довготривала пара ключів Curve25519, що генерується під час установки.
Підписаний Pre Key – середньотривала пара ключів Curve25519, що генерується під час установки, підписана ключем ідентифікації, і ротується періодично.

One-Time Pre Keys – черга пар ключів Curve25519 для одноразового використання, що генерується під час установки та поповнюється за потребою.
Root Key – 32-байтове значення, яке використовується для створення Chain Keys.
Chain Key – 32-байтове значення, що використовується для створення Message Keys.
Message Key – 80-байтове значення, що використовується для шифрування вмісту повідомлень. 32 байти використовуються для ключа AES-256, 32 байти для ключа HMAC-SHA256, і 16 байтів для IV.
Таємний ключ зв'язування - 32-байтове значення, що генерується на супутньому пристрої та повинне передаватися через захищений канал на основний пристрій, використовується для верифікації HMAC зв'язувального навантаження, отриманого з основного пристрою. Передача цього ключа з супутнього пристрою на основний виконується шляхом сканування QR-коду.
Метадані зв'язування - закодований блок метаданих, призначений супутньому пристрою під час зв'язування, використовується разом із ключем ідентифікації супутнього пристрою для ідентифікації підключеного супутника в клієнтах WhatsApp.
Підписаний список даних пристроїв - закодований список, що ідентифікує поточно підключені супутні пристрої на момент підпису, підписаний основним пристроєм ключем ідентифікації з префіксом 0x0602.

Реєстрація клієнта

Під час реєстрації клієнт WhatsApp передає на сервер свій публічний ідентифікаційний ключ, публічний підписаний Pre Key (разом з його підписом) та пакет публічних One-Time Pre Keys. Сервер WhatsApp зберігає ці публічні ключі, асоційовані з ідентифікатором користувача.

Щоб зв'язати додатковий пристрій з обліковим записом WhatsApp, основний пристрій користувача спочатку має створити підпис облікового запису, підписуючи публічний ідентифікаційний ключ нового пристрою, а додатковий пристрій має створити підпис пристрою, підписуючи публічний ідентифікаційний ключ основного пристрою. Після створення обох підписів можна встановити сеанси з кінцевим шифруванням з додатковим пристроєм.

Для підключення додаткового пристрою:

1. Клієнт додаткового пристрою відображає свій публічний ідентифікаційний ключ (Icompanion) та згенерований ефемерний таємний ключ зв'язування (Lcompanion) у QR-коді зв'язування. Lcompanion ніколи не відправляється на сервер WhatsApp.
2. Основний клієнт сканує QR-код зв'язування та зберігає Icompanion на диску.
3. Основний клієнт завантажує свій власний ідентифікаційний ключ як Iprimary.
4. Основний клієнт генерує метадані зв'язування як Lmetadata та оновлені дані списку пристроїв з новим додатковим як ListData.
5. Основний клієнт генерує підпис облікового запису для додаткового, Asignature = CURVE25519_SIGN(Iprimary, 0x0600 || Lmetadata || Icompanion).
6. Основний клієнт генерує підпис списку пристроїв для оновлених даних списку пристроїв, ListSignature = CURVE25519_SIGN(Iprimary, 0x0602 || ListData).
7. Основний клієнт серіалізує дані зв'язування (Ldata), містячи Lmetadata, Iprimary та Asignature.
8. Основний клієнт генерує HMAC зв'язування, PHMAC = HMACSHA256(Lcompanion, Ldata).
9. Основний клієнт надсилає ListData, ListSignature, Ldata та PHMAC на сервер WhatsApp.
10. Сервер зберігає ListData та ListSignature, та пересилає Ldata та PHMAC додатковому пристрою.
11. Додатковий пристрій перевіряє PHMAC, декодує Ldata на Lmetadata, Iprimary та Asignature, та перевіряє Asignature.
12. Додатковий пристрій зберігає Lmetadata та Iprimary на диску.
13. Додатковий пристрій генерує підпис пристрою для себе, Dsignature = CURVE25519_SIGN(Icompanion, 0x0601 || Lmetadata || Icompanion || Iprimary).
14. Додатковий пристрій завантажує Lmetadata, Asignature, Dsignature, Icompanion, публічний підписаний Pre Key додаткового пристрою (разом з його підписом) та пакет публічних One-Time Pre Keys на сервер WhatsApp.

15. Сервер зберігає завантажені дані, пов'язані з ідентифікатором користувача у поєднанні з ідентифікатором конкретного пристрою.

Початок налаштування сесії

Для того, щоб користувачі WhatsApp могли безпечно та приватно спілкуватися між собою, клієнт відправника створює парну зашифровану сесію з кожним пристроєм отримувача. Окрім того, клієнт відправника створює парну зашифровану сесію з усіма іншими пристроями, асоційованими з обліковим записом відправника. Як тільки ці парні зашифровані сесії будуть створені, клієнти не потребують створення нових сесій з цими пристроями, якщо не втрачено стан сесії, що може бути спричинено такою подією, як перевстановлення додатку чи зміна пристрою.

WhatsApp використовує підхід "клієнт-розгортання" для передачі повідомлень на кілька пристроїв, де клієнт WhatsApp передає одне повідомлення N кількість разів на N кількість різних пристроїв. Кожне повідомлення зашифроване індивідуально за допомогою створеної парної зашифрованої сесії з кожним пристроєм.

Щоб створити сесію:

1. Ініціюючий клієнт («ініціатор») запитує публічний ключ ідентифікації, публічний підписаний Pre Key та один публічний разовий Pre Key для кожного пристрою отримувача та кожного додаткового пристрою ініціюючого користувача (за винятком ініціатора).
2. Сервер повертає запитані значення публічних ключів. Разовий Pre Key використовується лише один раз, тому він видаляється зі сховища сервера після запиту. Якщо остання партія разових Pre Keys отримувача була використана, і отримувач не поповнив їх, разовий Pre Key не буде повернений. Додатково, для кожного супутнього пристрою (як для облікового запису ініціатора, так і отримувача), сервер також повертає Linking Metadata (Lmetadata), підпис облікового запису (Asignature) та підпис пристрою (Dsignature), які були завантажені супутнім пристроєм під час зв'язування.
3. Для кожного поверненого набору ключів для супутнього пристрою, ініціатор повинен перевірити Asignature за допомогою CURVE25519_VERIFY_SIGNATURE(Iprimary, 0x0600 || Lmetadata || Icompanion), та Dsignature за допомогою

CURVE25519_VERIFY_SIGNATURE(Icompanion, 0x0601 || Lmetadata || Icompanion || Iprimary). Якщо перевірка не вдалася для супутнього пристрою, ініціатор негайно припиняє процес побудови сесії шифрування і не відправляє жодних повідомлень на цей пристрій.

Отримавши ключі від сервера та перевіривши ідентичність кожного пристрою, ініціатор починає встановлювати сесію шифрування з кожним окремим пристроєм:

1. Ініціатор зберігає ключ ідентифікації отримувача як Irecipient, підписаний Pre Key як Srecipient, та разовий Pre Key як Orecipient.
2. Ініціатор генерує ефемерну пару ключів Curve25519, Einitiator.
3. Ініціатор завантажує свій власний ключ ідентифікації як Iinitiator.
4. Ініціатор розраховує основний секрет як master_secret = ECDH(Iinitiator, Srecipient) || ECDH(Einitiator, Irecipient) || ECDH(Einitiator, Srecipient) || ECDH(Einitiator, Orecipient). Якщо разового Pre Key немає, останній ECDH опускається.
5. Ініціатор використовує HKDF для створення Root Key та Chain Keys з master_secret.

Отримання налаштувань сесії

Для того, щоб користувачі WhatsApp могли безпечно та приватно спілкуватися між собою, клієнт відправника створює парну зашифровану сесію з кожним пристроєм отримувача. Окрім того, клієнт відправника створює парну зашифровану сесію з усіма іншими пристроями, асоційованими з обліковим записом відправника. Як тільки ці парні зашифровані сесії будуть створені, клієнти не потребують створення нових сесій з цими пристроями, якщо не втрачено стан сесії, що може бути спричинено такою подією, як перевстановлення додатку чи зміна пристрою.

WhatsApp використовує підхід "клієнт-розгортання" для передачі повідомлень на кілька пристроїв, де клієнт WhatsApp передає одне повідомлення N кількість разів на N кількість різних пристроїв. Кожне повідомлення зашифроване індивідуально за допомогою створеної парної зашифрованої сесії з кожним пристроєм.

Щоб створити сесію:

1. Ініціюючий клієнт («ініціатор») запитує публічний ключ ідентифікації, публічний підписаний Pre Key та один публічний разовий Pre Key для кожного пристрою отримувача та кожного додаткового пристрою ініціюючого користувача (за винятком ініціатора).
2. Сервер повертає запитані значення публічних ключів. Разовий Pre Key використовується лише один раз, тому він видаляється зі сховища сервера після запиту. Якщо остання партія разових Pre Keys отримувача була використана, і отримувач не поповнив їх, разовий Pre Key не буде повернений. Додатково, для кожного супутнього пристрою (як для облікового запису ініціатора, так і отримувача), сервер також повертає Linking Metadata (Lmetadata), підпис облікового запису (Asignature) та підпис пристрою (Dsignature), які були завантажені супутнім пристроєм під час зв'язування.
3. Для кожного поверненого набору ключів для супутнього пристрою, ініціатор повинен перевірити Asignature за допомогою CURVE25519_VERIFY_SIGNATURE(Iprimary, 0x0600 || Lmetadata || Icompanion), та Dsignature за допомогою CURVE25519_VERIFY_SIGNATURE(Icompanion, 0x0601 || Lmetadata || Icompanion || Iprimary). Якщо перевірка не вдалася для супутнього пристрою, ініціатор негайно припиняє процес побудови сесії шифрування і не відправляє жодних повідомлень на цей пристрій.

Отримавши ключі від сервера та перевіривши ідентичність кожного пристрою, ініціатор починає встановлювати сесію шифрування з кожним окремим пристроєм:

1. Ініціатор зберігає ключ ідентифікації отримувача як Irecipient, підписаний Pre Key як Srecipient, та разовий Pre Key як Orecipient.
2. Ініціатор генерує ефемерну пару ключів Curve25519, Einitiator.
3. Ініціатор завантажує свій власний ключ ідентифікації як Iinitiator.
4. Ініціатор розраховує основний секрет як master_secret = ECDH(Iinitiator, Srecipient) || ECDH(Einitiator, Irecipient) || ECDH(Einitiator, Srecipient) || ECDH(Einitiator, Orecipient). Якщо разового Pre Key немає, останній ECDH опускається.
5. Ініціатор використовує HKDF для створення Root Key та Chain Keys з master_secret.

Обмін повідомленнями

Після встановлення сесії клієнти обмінюються повідомленнями, захищеними ключем повідомлення за допомогою AES256 у режимі CBC для шифрування та HMAC-SHA256 для аутентифікації. Клієнт використовує розсилку клієнта для всіх обмінюваних повідомлень, що означає, що кожне повідомлення шифрується для кожного пристрою з відповідною парною сесією.

Ключ повідомлення змінюється для кожного переданого повідомлення і є ефемерним, так що ключ повідомлення, використаний для шифрування повідомлення, не може бути відтворений зі стану сесії після передачі або отримання повідомлення.

Ключ повідомлення походить від ланцюгового ключа відправника, який "рухається вперед" з кожним відправленим повідомленням. Крім того, з кожним повідомленням виконується нова угода ECDH для створення нового ланцюгового ключа. Це забезпечує пряму таємницю за допомогою комбінації негайного "хеш-руху" та "DH-руху" в обидва кінці.

Обчислення ключа повідомлення з Chain Key

Кожного разу, коли потрібен новий Ключ повідомлення для відправника повідомлення, він обчислюється так:

1. Ключ повідомлення = HMAC-SHA256(Chain Key, 0x01).
2. Потім Chain Key оновлюється як Chain Key = HMAC-SHA256(Chain Key, 0x02).

Це спричиняє «перемотування» Chain Key вперед, а також означає, що збережений Message Key не може бути використаний для виведення поточних або минулих значень Chain Key.

Обчислення Chain Key з Root Key

Кожного разу при передачі повідомлення разом з ним публікується ефемерний Curve25519 публічний ключ. Після отримання відповіді розраховуються новий Chain Key та Root Key наступним чином:

1. ephemeral_secret = ECDH(Ephemeralsender, Ephemeralrecipient).
2. Chain Key, Root Key = HKDF(Root Key, ephemeral_secret).

Ланцюг використовується лише для відправлення повідомлень від одного користувача, тому ключі повідомлень не повторно використовуються. Завдяки методу розрахунку Message Keys та Chain Keys, повідомлення

можуть надходити з затримкою, не в порядку, або навіть можуть загубитися, без будь-яких проблем.

Узгодженість у чаті пристроїв

У чатах із скрізнім шифруванням, для кожного вихідного повідомлення в рамках сесії попарного шифрування, включно з тими, що відправляються під час налаштування сесії, відправник включає інформацію про список пристроїв відправника та отримувача у зашифрований вміст. Ця інформація включає:

1. Часову мітку найсвіжішого Підписаного списку пристроїв відправника.
2. Прапорець, що вказує, чи має відправник на даний момент підключені супутні пристрої.
3. Часову мітку найсвіжішого Підписаного списку пристроїв отримувача.
4. Прапорець, що вказує, чи відомо про підключені супутні пристрої у отримувача.

Під час виконання "клієнтського розгалуження" до власних пристроїв пункти 3 та 4 вище продовжують відноситися до отримувача оригінального повідомлення.

Передача мультимедійних файлів та інших вкладень

Великі вкладення будь-якого типу (відео, аудіо, зображення або файли) також зашифровані скрізнь:

1. Пристрій користувача WhatsApp, який відправляє повідомлення ("відправник"), генерує ефемерний 32-байтний ключ AES256 і ефемерний 32-байтний ключ HMAC-SHA256.
2. Відправник шифрує вкладення за допомогою ключа AES256 в режимі CBC з випадковим IV, потім додає MAC шифротексту, використовуючи HMAC-SHA256.
3. Відправник завантажує зашифроване вкладення у сховище blob.
4. Відправник передає звичайне зашифроване повідомлення отримувачу, яке містить ключ шифрування, ключ HMAC, хеш SHA256 зашифрованого blob та посилання на blob у сховищі blob.

5. Усі приймаючі пристрої розшифровують повідомлення, отримують зашифрований blob зі сховища blob, перевіряють хеш SHA256 цього blob, перевіряють MAC і розшифровують відкритий текст.

Групові повідомлення

Шифрування від кінця до кінця повідомлень, які надсилаються до груп WhatsApp, використовує вже встановлені парні зашифровані сесії, як описано в розділі "Початок налаштування сесії", для розподілу компонента "Sender Key" протоколу Signal Messaging Protocol.

При першому надсиланні повідомлення в групу генерується "Sender Key", який розподіляється між всіма пристроями учасників групи за допомогою парних зашифрованих сесій. Вміст повідомлення шифрується використовуючи "Sender Key", забезпечуючи ефективний та безпечний розподіл повідомлень, що надсилаються до груп.

Коли учасник групи WhatsApp вперше надсилає повідомлення в групу:

1. Відправник генерує випадковий 32-байтовий Chain Key.
2. Відправник генерує випадкову пару ключів Curve25519 Signature Key.
3. Відправник поєднує 32-байтовий Chain Key та публічний ключ з Signature Key у повідомлення Sender Key.
4. Відправник індивідуально шифрує Sender Key для кожного учасника групи, використовуючи парний протокол обміну повідомленнями, описаний раніше.

Для всіх подальших повідомлень в групу:

1. Відправник отримує Message Key з Chain Key та оновлює Chain Key.
2. Відправник шифрує повідомлення, використовуючи AES256 у режимі CBC.
3. Відправник підписує зашифрований текст за допомогою Signature Key.
4. Відправник передає єдине зашифроване повідомлення на сервер, який виконує розподіл на стороні сервера для всіх учасників групи.

"Hash ratchet" Chain Key відправника повідомлення забезпечує пряму секретність. Коли учасник групи виходить, всі учасники групи очищають свій Sender Key і починають заново.

У відомостях про відповідність пристроїв у чаті інформація включається при розподілі "Sender Key" та потім виключається з подальших повідомлень, зашифрованих за допомогою Sender Key.

Налаштування дзвінка

Голосові та відеодзвінки WhatsApp зашифровані end-to-end. Коли користувач WhatsApp ініціює голосовий або відеодзвінок:

1. Ініціатор дзвінка ("ініціатор") налаштовує зашифровані сесії з кожним із пристроїв одержувача (як описано у розділі "Initiating Session Setup"), якщо вони ще не були налаштовані.
2. Ініціатор генерує набір випадкових 32-байтових SRTP master secrets для кожного з пристроїв одержувача.
3. Ініціатор відправляє повідомлення про вхідний дзвінок на кожен з пристроїв одержувача. Кожен пристрій одержувача отримує це повідомлення, яке містить end-to-end зашифрований SRTP master secret.
4. Якщо відповідач приймає дзвінок з одного з пристроїв, розпочинається зашифрований SRTP дзвінок, захищений SRTP master secret, створеним для цього пристрою.

SRTP master secret зберігається в пам'яті клієнтського пристрою та використовується лише під час дзвінка. Сервери WhatsApp не мають доступу до SRTP master secrets.

Групові дзвінки

Групові дзвінки WhatsApp також зашифровані end-to-end. На відміну від дзвінків один на один, де ключі налаштовуються лише раз, у групових дзвінках ключі скидаються кожного разу, коли учасник приєднується або виходить з дзвінка.

Скидання ключів у групових дзвінках відбувається за наступними кроками:

1. Коли учасник приєднується або виходить з дзвінка, сервер WhatsApp довільно вибирає одного з активних учасників як розподілювача ключів.
2. Розподілювач ключів генерує випадковий 32-байтовий SRTP master secret.

3. Розподілювач ключів налаштовує зашифровану сесію з активним пристроєм кожного підключеного учасника поточного групового дзвінка (як описано у розділі "Initiating Session Setup"), якщо вони ще не були налаштовані.
4. Розподілювач ключів ініціює одне повідомлення з end-to-end зашифрованим SRTP master secret для кожного учасника. Коли ці повідомлення доставляються, розпочинається зашифрований SRTP груповий дзвінок.

Зауважте, що в груповому дзвінку учасник стає активним, коли він ініціює груповий дзвінок або приймає запрошення до групового дзвінка з будь-якого зі своїх пристроїв. Таким чином, кожен активний учасник має рівно один активний пристрій.

SRTP master secret зберігається в пам'яті та перезаписується, коли генерується та доставляється новий SRTP master secret. Щоб продовжити розшифровку даних, зашифрованих старим ключем, поки всі учасники переходять на новий ключ, стара SRTP криптосесія зберігається протягом до 5 секунд після оновлення групи.

Сервери WhatsApp не мають доступу до них і не можуть отримати доступ до фактичних аудіо та відеомедіа.

Для забезпечення якості дзвінка та уникнення конфліктних ситуацій від суперечливих дій користувачів, сервер WhatsApp зберігає стан поточного групового дзвінка (наприклад, список учасників, ініціатор дзвінка) та метадані медіа (наприклад, роздільна здатність, тип медіа). З цією інформацією сервер WhatsApp може транслювати зміни у складі учасників та вибирати одного як розподілювача ключів для ініціації скидання ключів.

Верифікація ключів

Користувачі WhatsApp мають також можливість перевіряти ключі своїх пристроїв і пристроїв користувачів, з якими вони спілкуються у чатах з end-to-end шифруванням, щоб вони могли підтвердити, що несанкціонована третя сторона (або сам WhatsApp) не здійснила атаку

"людина посередині". Верифікація може бути здійснена шляхом сканування QR-коду або порівняння 60-значного числа між двома основними пристроями. Користувачі WhatsApp також можуть вручну верифікувати окремі супутні пристрої, використовуючи основний пристрій для перевірки того ж QR-коду або 60-значного числа.
QR-код містить:

1. Версію.
2. Ідентифікатор користувача для обох сторін.
3. Повний 32-байтний публічний Identity Key для всіх пристроїв обох сторін.

Коли один пристрій сканує QR-код іншого, ключі порівнюються, щоб переконатися, що те, що є в QR-коді, відповідає Identity Key, який отриманий з сервера.
60-значне число розраховується шляхом конкатенації двох 30-значних числових відбитків для Identity Keys кожного пристрою користувача. Щоб розрахувати 30-значний числовий відбиток:

1. Лексикографічно впорядкувати публічні Identity Keys для всіх пристроїв користувача та конкатенувати їх.
2. Ітеративно хешувати SHA-512 сортовані Identity Keys та ідентифікатор користувача 5200 разів.
3. Взяти перші 30 байтів остаточного хеш-виводу.
4. Розділити 30-байтовий результат на шість 5-байтових частин.
5. Конвертувати кожну 5-байтову частину в 5 цифр, інтерпретуючи кожну 5-байтову частину як велике беззнакове ціле в big-endian форматі та зменшуючи його modulo 100000.
6. Конкатенувати шість груп по п'ять цифр у тридцять цифр.

Транспортна безпека

Спілкування між клієнтами WhatsApp та серверами чату WhatsApp здійснюється через окремий зашифрований канал, який використовує Noise Pipes з Curve25519, AES-GCM та SHA256 з фреймворку Noise Protocol для тривалих інтерактивних з'єднань. Це надає клієнтам декілька важливих переваг:

1. Дуже швидке та легке налагодження з'єднання та його відновлення.

2. Шифрування метаданих для приховування їх від несанкціонованих спостерігачів у мережі. Інформація про ідентичність користувача, який підключається, не розкривається.
3. На сервері не зберігаються секрети автентифікації клієнтів. Клієнти автентифікують себе за допомогою пари ключів Curve25519, тому сервер зберігає лише публічний ключ автентифікації клієнта. У випадку компрометації бази даних користувачів сервера, жодні приватні дані для автентифікації не будуть розкриті.

Skype

Skype — це система для голосових дзвінків через Інтернет за допомогою технології VoIP, створений компанією Skype Technologies SA. Ця платформа працює на принципах одноранговой мережі, дозволяючи користувачам здійснювати дзвінки через Інтернет, а не через звичайну телефонну мережу. У Skype можна знаходити інших користувачів та відправляти їм повідомлення.

Для захисту зв'язку між клієнтами Skype використовується 256-бітне шифрування AES/Rijnadel. Однак, коли дзвінок здійснюється на звичайний або мобільний телефон, частина дзвінка, що проходить через загальнодоступну комутовану телефонну мережу (PSTN), не зашифрована. Публічні ключі користувачів підтверджуються сервером Skype за допомогою 1536-бітних або 2048-бітних RSA сертифікатів. Шифрування Skype є інтегрованою частиною його протоколу і є непомітним для користувачів. Приватні розмови в Skype, такі як аудіодзвінки, текстові повідомлення та передача файлів, можуть використовувати наскрізне шифрування, яке, можливо, потрібно активувати вручну.

Політика безпеки в компанії передбачає такі основні пункти:
1. Унікальність імен користувачів.
2. Необхідність надання користувачами свого імені та пароля або інших даних для автентифікації.
3. Взаємне підтвердження особистості та прав користувачів при кожному з'єднанні. Перед початком сеансу необхідна перевірка доказів один одного.

4. Шифрування повідомлень між користувачами Skype, не доступне проміжним вузлам. Однак, існують докази, що Microsoft (власник Skype) має доступ до цих повідомлень у незашифрованому вигляді.

Skype зберігає реєстраційні дані на комп'ютері користувача та на сервері. Для автентифікації та забезпечення безпеки використовується шифрування з відкритим ключем, як у RSA. Сервер має приватний ключ і розповсюджує публічний ключ. При реєстрації користувач обирає ім'я та пароль, а також генерує відкритий і приватний ключі, зберігаючи їх локально.

Сеанс з сервером Skype використовує 256-бітне шифрування AES. Сервер перевіряє унікальність імені користувача і зберігає ім'я та хеш пароля. Потім сервер формує та підписує сертифікат ідентифікації, пов'язуючи ім'я користувача з ключем перевірки та ідентифікатором ключа.

Одноранговa угода про ключ
Кожен виклик Skype генерує 256-бітний сеансовий ключ. Цей ключ існує протягом усього спілкування та додатковий визначений час після нього. Skype безпечно надсилає цей ключ отримувачеві дзвінка при встановленні з'єднання. Використовуючи цей ключ, повідомлення шифруються в обох напрямках.

Криптографія сеансу
Весь трафік під час сеансу шифрується через AES у режимі цілочисельного лічильника (ICM). Skype шифрує поточне значення лічильника та сіль, використовуючи 256-бітний AES ключ сеансу. Цей метод створює потік ключів, які потім комбінуються з вмістом повідомлення через XOR.
Сесії Skype мають декілька потоків, де значення ICM залежить від потоку та його місця розташування.

Генерація випадкових чисел
Skype використовує випадкові числа для різних криптографічних завдань, включаючи захист від повторних атак, створення RSA ключів та половини AES ключів для шифрування даних. Безпека сеансу Skype суттєво

залежить від якості випадкових чисел, що генеруються обома сторонами сеансу. Якість цих чисел залежить від операційної системи.

Криптографічні примітиви
Skype використовує стандартні криптографічні примітиви, включаючи блоковий шифр AES, систему з відкритим ключем RSA, схему підпису ISO 9796-2, хеш-функцію SHA-1 та потоковий шифр RC4 для забезпечення безпеки.

Протокол Узгодження Ключів
Ключі узгоджуються через приватний симетричний протокол. Для захисту від атак повторення, використовуються випадкові 64-бітні числа для виклику одне одного. Відповідь на виклик включає приватне налаштування та повернення цього числа, підписаного особистим ключем. Сертифікати ідентифікації обмінюються між вузлами, що дозволяє перевірити їхню справжність. Оскільки сертифікат містить відкритий ключ, кожна сторона може перевірити підписи іншої сторони. До 256-бітового ключа сеансу додаються 128 випадкових бітів.

Автоматичне Оновлення
Починаючи з версії 5.6, автоматичні оновлення не можна вимкнути в Mac OS та Windows, хоча у Windows з версії 5.9 це можливо у певних випадках.

Схема Прослуховування
Китайські, російські та американські правоохоронні органи мають доступ до прослуховування розмов Skype та до географічного розташування користувачів. Часто для цього достатньо простого запиту, без судового дозволу. Microsoft свідомо додала цю можливість після придбання Skype у 2011 році, перемикаючи шифрування зі сторони клієнта на серверне, що дозволяє перехоплювати незашифровані дані.

Фактичні та потенційні недоліки
Незважаючи на шифрування користувацьких сесій Skype, інший трафік, як-от дзвінки, може бути доступний для сторонніх. Особливу увагу варто приділити ризикам для безпеки комп’ютерів та мереж користувачів. У жовтні 2005 року було виявлено та виправлено пару недоліків безпеки. Ці

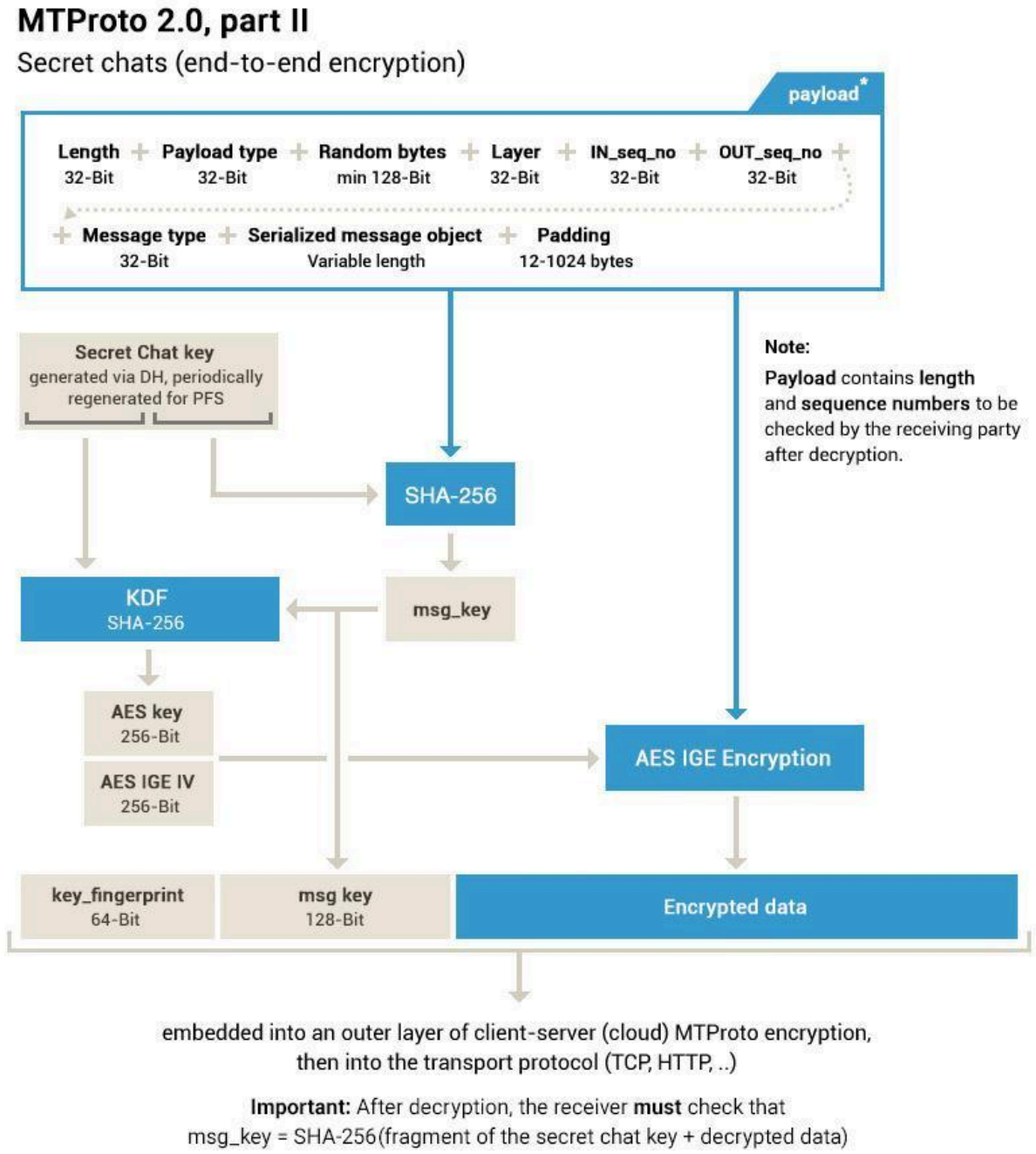
недоліки дозволили хакерам запустити ворожий код на комп'ютерах із уразливими версіями Skype. Перша помилка безпеки торкнулася лише комп'ютерів Microsoft Windows . Це дозволяло зловмиснику використовувати переповнення буфера для збою системи або змусити її виконати довільний код. Зловмисник може надати неправильну URL-адресу за допомогою формату URI Skype і спонукати користувача попросити його виконати атаку. Друга помилка безпеки вплинула на всі платформи; він використовував переповнення буфера на основі купи , щоб зробити систему вразливою.

Проблеми безпеки включають:

- Відсутність експертної перевірки коду Skype, якщо б він був відкритим.
- У 2012 році російський користувач виявив проблему безпеки, яка дозволяла захопити обліковий запис за електронною адресою.
- Запис інформації про дзвінки у файлі на комп'ютері користувача, доступному для зловмисників.
- Skype використовує пропускну здатність інших користувачів, що задокументовано в EULA.
- Супервузли і ретранслятори у Skype використовують певну пропускну здатність.
- Функція передачі файлів Skype не інтегрована з антивірусними програмами.
- Skype не документує всю свою комунікаційну діяльність.
- Програма споживає пропускну здатність, навіть будучи неактивною.
- Skype не забороняє паралельні мережі, схожі на себе.
- До версії 3.0.0.216 Skype створював файл, який міг читати дані BIOS.
- Обробник URI не перевіряє всі потенційні формати файлів.
- Незашифровані пакети з рекламою в Skype виявляють вразливість.
- Можливість компанії перехоплювати комунікацію.
- Skype для Linux доступається до папки профілю Firefox, а для Mac - до системної адресної книги.
- Запитання щодо доступу Skype до адресної книги без інтеграції.
- Федеральна комісія зі зв'язку США вимагає дозволити прослуховування в цифрових телефонних мережах.

Telegram

Секретні чати - це чати "один на один", в яких повідомлення шифруються за допомогою ключа, який мають лише учасники чату. Слід зазначити, що схема наскрізного шифрування секретних чатів відрізняється від тієї, що використовується для хмарних чатів:



Примітка про MTProto 2.0

Основні відмінності від версії 1.0 наступні:

- Замість SHA-1 використовується SHA-256;
- У обчисленні msg_key беруть участь байти padding;
- msg_key залежить не лише від повідомлення, яке шифрується, але й від частини секретного ключа чату;
- Використовується 12...1024 байт padding замість 0...15 байт padding у версії 1.0.

Генерація ключів

Ключі генеруються за допомогою протоколу Діффі-Хеллмана.
Розглянемо наступний сценарій: Користувач A хоче ініціювати наскрізний зашифрований зв'язок з користувачем B.

Відправлення запиту

Користувач A виконує messages.getDhConfig для отримання параметрів Діффі-Хеллмана: простого числа p і елемента високого порядку g. Виконання цього методу перед кожною новою процедурою генерації ключів є дуже важливим. Значення параметрів має сенс кешувати разом з версією, щоб уникнути необхідності кожного разу отримувати всі значення. Якщо версія, що зберігається на клієнті, все ще актуальна, сервер поверне конструктор messages.dhConfigNotModified.

Клієнт повинен перевірити, чи є p безпечним 2048-бітним простим числом (мається на увазі, що і p, і (p-1)/2 є простими, і що 2^2047 < p < 2^2048), і що g породжує циклічну підгрупу простого порядку (p-1)/2, тобто є квадратичним залишком mod p. Оскільки g завжди дорівнює 2, 3, 4, 5, 6 або 7, це легко зробити за допомогою квадратичного закону взаємності, що дає просту умову для p mod 4g, а саме: p mod 8 = 7 для g = 2; p mod 3 = 2 для g = 3; без додаткових умов 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 клієнт перевірив g і p, має сенс закешувати результат, щоб уникнути повторення довгих обчислень у майбутньому. Цей кеш можна використовувати спільно з тим, що використовується для генерації ключа авторизації.
Якщо клієнту потрібна додаткова ентропія для генератора випадкових чисел, він може передати параметр random_length (random_length> 0), щоб сервер згенерував власну випадкову послідовність відповідної довжини.

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

Клієнт A обчислює 2048-бітне число a (використовуючи достатню ентропію або випадкову послідовність сервера) 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 (всіх авторизованих пристроїв) за допомогою конструктора чату encryptedChatRequested. Користувачеві має бути показано основну інформацію про користувача A і запропоновано прийняти або відхилити запит.

Обидва клієнти повинні перевірити, що g, g_a і g_b більші за одиницю і менші за p-1. Рекомендується також перевірити, що g_a і g_b знаходяться між 2^{2048-64} і p - 2^{2048-64}.

Прийом запиту

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

Отримавши g_a від оновлення із encryptedChatRequested, він може одразу згенерувати остаточний спільний ключ: key = (pow(g_a, b) mod dh_prime). Якщо довжина ключа < 256 байт, додаємо кілька початкових нульових байт для padding - так, щоб довжина ключа дорівнювала 256 байт. Його відбиток, key_fingerprint, дорівнює 64 останнім бітам SHA1 (key).

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

Примітка 2: цей відбиток використовується як перевірка адекватності процедури обміну ключами для виявлення помилок при розробці клієнтського програмного забезпечення - він не пов'язаний з візуалізацією ключа, що використовується на клієнтах як засіб зовнішньої автентифікації

в секретних чатах. Візуалізація ключів на клієнтах генерується з використанням перших 128 біт SHA1 (початковий ключ), а потім перших 160 біт SHA256 (ключ, який використовується, коли секретний чат було оновлено до рівня 46).

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

Для всіх авторизованих пристроїв клієнта B, окрім поточного, оновлення updateEncryption надсилаються за допомогою конструктора encryptedChatDiscarded. Після цього єдиним пристроєм, який матиме доступ до секретного чату, буде пристрій B, який здійснив виклик 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 спільний ключ key = (pow(g_b, a) mod dh_prime). Якщо довжина ключа < 256 байт, додайте кілька початкових нульових байт як заповнення - так, щоб ключ мав довжину рівно 256 байт. Якщо відбиток отриманого ключа збігається з тим, що було передано до encryptedChat, вхідні повідомлення можна надсилати та обробляти. В іншому випадку слід виконати messages.discardEncryption і повідомити про це користувача.

Ідеальна передня секретність

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 TL-об'єкт типу DecryptedMessage, який містить повідомлення у відкритому вигляді. Для зворотної сумісності об'єкт повинен бути обгорнутий у конструктор decryptedMessageLayer із зазначенням підтримуваного рівня (починаючи з 46).

Отримана конструкція серіалізується у вигляді масиву байт з використанням загальних правил TL. До масиву додаються 4 байти, що містять довжину масиву, не враховуючи ці 4 байти.
Байтовий масив доповнюється випадковими байтами від 12 до 1024, щоб його довжина була кратною 16 байтам. (У старому шифруванні MTProto 1.0 використовувалося лише від 0 до 15 байт заповнення).
Ключ повідомлення, msg_key, обчислюється як 128 середніх бітів SHA256 даних, отриманих на попередньому кроці, до яких додаються 32 байти зі спільного ключа. (Для старішого шифрування MTProto 1.0 msg_key обчислювався інакше, як 128 молодших бітів SHA1 даних, отриманих на попередніх кроках, за винятком байт заповнення).
Для MTProto 2.0 ключ AES aes_key та вектор ініціалізації aes_iv (ключ - це спільний ключ, отриманий під час генерації ключів) обчислюються наступним чином:
msg_key_large = SHA256 (substr (key, 88+x, 32) + plaintext + random_padding);
msg_key = substr (msg_key_large, 8, 16);
sha256_a = SHA256 (msg_key + substr (key, x, 36));
sha256_b = SHA256 (substr (key, 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 творця секретного чату, x=8 для повідомлень у зворотному напрямку.
У застарілому MTProto 1.0 msg_key, aes_key та aes_iv обчислювалися по-іншому.
Дані шифруються 256-бітовим ключем aes_key і 256-бітовим вектором ініціалізації aes-iv з використанням шифрування AES-256 з нескінченним

розширенням (IGE). Відбиток ключа шифрування key_fingerprint і ключ повідомлення msg_key додаються у верхній частині результуючого масиву байт.
Зашифровані дані вбудовуються у виклик API messages.sendEncrypted і передаються на сервер Telegram для доставки іншому учаснику секретного чату.

Оновлення до MTProto 2.0 з MTProto 1.0

Якщо обидві сторони в секретному чаті використовують принаймні рівень 73, вони повинні використовувати тільки MTProto 2.0 для всіх вихідних повідомлень. Деякі з перших отриманих повідомлень можуть використовувати MTProto 1.0, якщо під час створення секретного чату не було обговорено достатньо високий початковий рівень. Після отримання першого повідомлення, зашифрованого за допомогою MTProto 2.0 (або першого повідомлення з рівнем 73 або вище), всі повідомлення з вищими порядковими номерами також повинні бути зашифровані за допомогою MTProto 2.0.
Поки поточний рівень нижче 73, кожна сторона повинна спробувати розшифрувати отримані повідомлення за допомогою MTProto 1.0, і якщо це не вдасться (msg_key не збігається), спробувати MTProto 2.0. Як тільки надійде перше повідомлення, зашифроване за допомогою MTProto 2.0 (або рівень буде підвищено до 73), немає необхідності намагатися розшифрувати наступні повідомлення за допомогою MTProto 1.0 (якщо тільки клієнт все ще не чекає на усунення деяких прогалин).

Розшифровка вхідного повідомлення

Наведені вище кроки виконуються у зворотному порядку.
При отриманні зашифрованого повідомлення необхідно перевірити, що msg_key дійсно дорівнює 128 середнім бітам хешу SHA256 розшифрованого повідомлення, до яких додаються 32 байти, взяті зі спільного ключа.
Якщо рівень повідомлення перевищує рівень, який підтримується клієнтом, користувач має бути повідомлений про те, що версія клієнта застаріла, і йому буде запропоновано оновити її.

Порядкові номери

Для захисту від можливих маніпуляцій необхідно інтерпретувати всі повідомлення в оригінальному порядку. Секретні чати підтримують спеціальний механізм обробки лічильників seq_no незалежно від сервера. Зверніть увагу, що для сумісності з офіційними клієнтами Telegram ваш клієнт повинен підтримувати порядкові номери в секретних чатах.

Надсилання зашифрованих файлів

Усі файли, що надсилаються до секретних чатів, шифруються одноразовими ключами, які жодним чином не пов'язані із загальним ключем чату. Перед відправкою зашифрованого файлу передбачається, що адреса зашифрованого файлу буде приєднана до зашифрованого повідомлення за допомогою параметра file методу messages.sendEncryptedFile, а ключ для безпосереднього розшифрування буде відправлений в тілі повідомлення (параметр key в конструкторах decryptedMessageMediaPhoto, decryptedMessageMediaVideo і decryptedMessageMediaFile.
Перед відправкою файлу в секретний чат обчислюються 2 випадкових 256-бітних числа, які будуть служити ключем AES і вектором ініціалізації, що використовуються для шифрування файлу. Шифрування AES-256 з нескінченним розширенням Garble (IGE) використовується аналогічним чином.
Відбиток ключа обчислюється наступним чином:
digest = md5(key + iv)
fingerprint = substr(digest, 0, 4) XOR substr(digest, 4, 4)
Зашифрований вміст файлу зберігається на сервері так само, як і в хмарних чатах: по частинах за допомогою викликів upload.saveFilePart. Наступний виклик messages.sendEncryptedFile присвоїть збереженому файлу ідентифікатор і надішле його адресу разом із повідомленням. Одержувач отримає оновлення з параметром encryptedMessage, а параметр file міститиме інформацію про файл.
Вхідні та вихідні зашифровані файли можна пересилати в інші секретні чати за допомогою конструктора inputEncryptedFile, щоб не зберігати той самий вміст на сервері двічі.

Робота з вікном оновлень

Секретні чати пов'язані з конкретними пристроями (точніше, з ключами авторизації), а не з користувачами. Звичайна поштова скринька, яка використовує pts для опису статусу клієнта, не підходить, оскільки вона призначена для довготривалого зберігання повідомлень і доступу до них з різних пристроїв.
Для вирішення цієї проблеми вводиться додаткова тимчасова черга повідомлень. Коли надсилається оновлення щодо повідомлення з секретного чату, надсилається нове значення qts, що допомагає відновити різницю, якщо була довга перерва у з'єднанні або у випадку втрати оновлення.
Зі збільшенням кількості подій значення qts збільшується на 1 з кожною новою подією. Початкове значення не може (і не буде) дорівнювати 0.
Той факт, що події з тимчасової черги були отримані і збережені клієнтом, підтверджується явно викликом методу messages.receivedQueue або неявно викликом updates.getDifference (передається значення qts, а не кінцевий стан). Усі повідомлення, які було підтверджено як доставлені клієнтом, а також будь-які повідомлення старші за 7 днів, можуть бути (і будуть) видалені з сервера.
Після деавторизації черга подій відповідного пристрою буде примусово очищена, а значення qts стане неактуальним.

Оновлення до нових шарів

Ваш клієнт завжди повинен зберігати максимальний рівень, який, як відомо, підтримується клієнтом на іншій стороні секретного чату. Коли секретний чат створюється вперше, це значення має бути ініціалізовано на 46. Це значення віддаленого рівня має завжди оновлюватися одразу після отримання будь-якого пакета, що містить інформацію верхнього рівня, тобто:
будь-якого повідомлення секретного чату, що містить layer_no у своєму decryptedMessageLayer з layer>=46, або
службове повідомлення decryptedMessageActionNotifyLayer, обгорнуте так, ніби воно є конструктором decryptedMessageService застарілого рівня 8 (конструктор decryptedMessageService#aa48327d).

Повідомлення віддаленого клієнта про ваш локальний рівень

Для того, щоб повідомити віддаленого клієнта про ваш локальний рівень, ваш клієнт повинен надіслати повідомлення з розшифрованим типом MessageActionNotifyLayer. Це повідомлення повинно бути загорнуте у конструктор відповідного рівня.
Існує два випадки, коли ваш клієнт повинен повідомити віддаленого клієнта про свій локальний рівень:

1. Одразу після створення нового секретного чату, одразу після успішного обміну секретним ключем.
2. Одразу після оновлення локального клієнта для підтримки нового рівня секретного чату. У цьому випадку сповіщення мають бути надіслані до всіх існуючих на даний момент секретних чатів. Зауважте, що це необхідно лише при оновленні до нових рівнів, які містять зміни в реалізації секретних чатів (наприклад, вам не потрібно робити це при оновленні клієнта з рівня 46 до рівня 47).

Встановлення дзвінків

Before a call is ready, some preliminary actions have to be performed. The calling party needs to contact the party to be called and check whether it is ready to accept the call. Besides that, the parties have to negotiate the protocols to be used, learn the IP addresses of each other or of the Telegram relay servers to be used (so-called reflectors), and generate a one-time encryption key for this voice call with the aid of Diffie--Hellman key exchange. All of this is accomplished in parallel with the aid of several Telegram API methods and related notifications. This document covers details related to key generation, encryption and security.

Генерація ключів

Обмін ключами Діффі-Хеллмана, а також весь протокол, який використовується для створення нового голосового виклику, дуже схожий на той, що використовується для секретних чатів. Ми рекомендуємо ознайомитися зі статтею за посиланням, перш ніж продовжити.
Однак ми внесли деякі важливі зміни, щоб полегшити процес перевірки ключа. Нижче наведено весь обмін між двома сторонами, що спілкуються, абонентом (A) і співрозмовником (B), через сервери Telegram (S).
A виконує messages.getDhConfig, щоб дізнатися 2048-бітове просте число Діффі-Хеллмана p і генератор g. Очікується, що клієнт перевірить, чи є p

безпечним простим числом, і виконає всі перевірки безпеки, необхідні для секретних чатів.
A вибирає випадкове значення a, 1 < a < p-1, і обчислює g_a:=power(g,a) mod p (256-байтне число) і g_a_hash:=SHA256(g_a) (довжиною 32 байти).
A викликає (надсилає на сервер S) phone.requestCall, який має поле g_a_hash:bytes, серед інших. Для цього виклику це поле має бути заповнене g_a_hash, а не самою g_a.
Сервер S виконує перевірку конфіденційності і надсилає оновлення updatePhoneCall з конструктором phoneCallRequested на всі активні пристрої B. Це оновлення, окрім ідентифікатора A та інших відповідних параметрів, містить поле g_a_hash, заповнене значенням, отриманим від A.
B приймає виклик на одному зі своїх пристроїв, зберігає отримане значення g_a_hash для цього екземпляра протоколу створення голосового виклику, вибирає випадкове значення b, 1 < b < p-1, обчислює g_b:=power(g,b) mod p, виконує всі необхідні перевірки безпеки і викликає метод phone.acceptCall, який має поле g_b:bytes (серед інших), яке має бути заповнене значенням самого g_b (а не його хешем).
Сервер S надсилає updatePhoneCall з конструктором phoneCallDiscarded на всі інші пристрої, які авторизував B, щоб запобігти прийняттю того самого дзвінка на будь-якому іншому пристрої. З цього моменту сервер S працює лише з тими пристроями B, які першими викликали phone.acceptCall.
Сервер S надсилає A оновлення updatePhoneCall з конструктором phoneCallAccepted, що містить значення g_b, отримане від B.
A виконує всі звичайні перевірки безпеки для g_b і a, обчислює ключ Діффі-Хелмана key:=power(g_b,a) mod p і його відбиток key_fingerprint:long, що дорівнює молодшим 64 бітам SHA1(key), так само, як і у випадку з секретними чатами. Потім A викликає метод phone.confirmCall, що містить g_a:bytes і key_fingerprint:long.
Сервер S надсилає B оновлення updatePhoneCall з конструктором phoneCall, що містить значення g_a у полі g_a_or_b:bytes та key_fingerprint:long
У цей момент B отримує значення g_a. Він перевіряє, що SHA256(g_a) дійсно дорівнює раніше отриманому значенню g_a_hash, виконує всі звичайні перевірки безпеки Діффі-Хеллмана і обчислює ключ key:=power(g_a,b) mod p і його відбиток, що дорівнює молодшим 64 бітам SHA1(key). Потім він перевіряє, що цей відбиток дорівнює значенню

key_fingerprint:long, отриманому від іншої сторони, як перевірка коректності реалізації.
На цьому обмін ключами Діффі-Хеллмана завершено, і обидві сторони мають 256-байтовий спільний секретний ключ, який використовується для шифрування всіх подальших обмінів між A і B.
Дуже важливо приймати кожне оновлення тільки один раз для кожного екземпляра протоколу генерації ключів, відкидаючи будь-які дублікати або альтернативні версії вже отриманих і оброблених повідомлень (оновлень).

Шифрування

Тут описано шифрування голосових і відеодзвінків, реалізоване в додатках Telegram версії 7.0 і вище.
Бібліотека голосових і відеодзвінків Telegram використовує оптимізовану версію MTProto 2.0 для надсилання та отримання пакетів, що складаються з одного або декількох наскрізних зашифрованих повідомлень різного типу (список кандидатів у лідери, відеоформати, статус віддаленого відео, дані аудіопотоку, дані відеопотоку, повідомлення з підтвердженням або порожнє).
Цей документ описує лише процес шифрування, не враховуючи кодування та мережево-залежні частини.
Бібліотека починає працювати з:
Ключ шифрування, спільний для сторін, згенерований вище.
Інформація про те, чи є виклик вихідним або вхідним.
Два канали передачі даних: сигнальний, що надається API Telegram, і транспортний, заснований на WebRTC.
Обидва канали передачі даних ненадійні (повідомлення можуть губитися), але сигналізація повільніша і надійніша.

Шифрування даних дзвінків

Структура пакета (decrypted_body) складається з декількох повідомлень та їхніх порядкових номерів, об'єднаних разом.
decrypted_body = message_seq1 + message_body1 + message_seq2 + message_body2
Кожне decrypted_body є унікальним, оскільки не може бути двох однакових seq-номерів першого повідомлення. Якщо потрібно повторно відправити лише старі повідомлення, спочатку до пакета додається порожнє повідомлення з новим унікальним seq.

Ключ шифрування використовується для обчислення 128-бітного msg_key, а потім 256-бітного aes_key і 128-бітного aes_iv:
msg_key_large = SHA256 (substr(key, 88+x, 32) + decrypted_body);
msg_key = substr (msg_key_large, 8, 16);
sha256_a = SHA256 (msg_key + substr (key, x, 36));
sha256_b = SHA256 (substr (key, 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, 4) + substr (sha256_a, 8, 8) + substr (sha256_b, 24, 4);
x залежить від того, чи є виклик вихідним або вхідним, а також від типу з'єднання:
x = 0 для вихідного + транспорт
x = 8 для вхідного + транспорт
x = 128 для вихідного + сигналізація
x = 136 для вхідного + сигналізація
Це дозволяє програмам вирішувати, які типи пакетів будуть надсилатися до яких з'єднань, і працювати в цих з'єднаннях незалежно (з власним лічильником послідовностей для кожного з них).
Отримані aes_key та aes_iv використовуються для шифрування decrypted_body:
encrypted_body = AES_CTR (decrypted_body, aes_key, aes_iv)
Пакет, який буде надіслано, складається з msg_key та encrypted_body:
packet_bytes = msg_key + encrypted_body
При отриманні пакет розшифровується за допомогою key і msg_key, після чого msg_key порівнюється з відповідним підрядком SHA256. Якщо перевірка не вдалася, пакет має бути відкинутий.

Захист від Replay атак

Кожен з однорангових вузлів має власний 32-бітний лічильник вихідних повідомлень seq, що монотонно збільшується, починаючи з 1. Цей лічильник seq додається до кожного надісланого повідомлення і збільшується на 1 для кожного нового повідомлення. Жодні два номери seq першого повідомлення у пакеті не можуть бути однаковими. Якщо потрібно повторно відправити лише старі повідомлення, спочатку до пакета додається порожнє повідомлення з новим унікальним номером. Коли лічильник seq досягає значення 2^30, виклик має бути перерваний.

Кожен пірінг зберігає значення seq всіх отриманих (і оброблених) ним повідомлень, які перевищують max_received_seq - 64, де max_received_seq - це найбільше значення seq, отримане до цього часу.
Якщо отримано пакет, перше повідомлення якого має seq менше або дорівнює max_received_seq - 64, або його seq вже було отримано, то повідомлення відкидається. В іншому випадку значення seq всіх вхідних повідомлень запам'ятовуються і max_received_seq коригується. Це гарантує, що жодні два пакети не будуть оброблені двічі.

Верифікація ключів

Щоб перевірити ключ і переконатися, що не відбувається MITM-атака, обидві сторони об'єднують секретний ключ зі значенням g_a абонента (A), обчислюють SHA256 і використовують його для генерації послідовності смайликів. Точніше, хеш SHA256 розбивається на чотири 64-бітних цілих числа; кожне з них ділиться на загальну кількість використовуваних смайликів (наразі 333), а залишок використовується для вибору конкретних смайликів. Специфіка протоколу гарантує, що порівняння чотирьох смайликів з набору 333 є достатнім для запобігання прослуховуванню (MiTM-атака на DH) з ймовірністю 0.9999999999.
Це відбувається тому, що замість стандартного обміну ключами Діффі-Хеллмана, який вимагає лише двох повідомлень між сторонами, використовується протокол Diffie-Hellman:
A->B : (генерує a і) відправляє g_a := g^a
B->A : (генерує b та істинний ключ (g_a)^b, потім) надсилає g_b := g^b
A : обчислює ключ (g_b)^a
ми використовуємо його модифікацію з трьома повідомленнями, яка добре працює, коли обидві сторони онлайн (що також є вимогою для голосових дзвінків):
A->B : (генерує a і) надсилає g_a_hash := hash(g^a)
B->A : (зберігає g_a_hash, генерує b і) відправляє g_b := g^b
A->B : (обчислює ключ (g_b)^a, потім) надсилає g_a := g^a
B : перевіряє hash(g_a) == g_a_hash, потім обчислює ключ (g_a)^b
Ідея полягає в тому, що A зобов'язується використовувати певне значення a (і g_a), не розкриваючи його B. B повинен вибрати своє значення b і g_b, не знаючи справжнього значення g_a, так що він не може спробувати різні значення b, щоб змусити остаточний ключ (g_a)^b мати якісь певні властивості (наприклад, фіксовані молодші 32 біти SHA256(key)). На

цьому етапі B приймає певне значення g_b, не знаючи значення g_a. Після цього A має надіслати своє значення g_a; він не може змінити його, навіть якщо тепер знає g_b, тому що інша сторона B прийме тільки те значення g_a, яке має хеш, вказаний у першому повідомленні обміну.
Якщо якийсь самозванець видає себе за A або B і намагається здійснити атаку типу "людина посередині" на цей обмін ключами за методом Діффі-Хелмана, то вищеописане все ще залишається в силі. Сторона A згенерує спільний ключ з B - або тим, хто видає себе за B - не маючи другого шансу змінити свою експоненту a в залежності від значення g_b, отриманого від іншої сторони; а самозванець не матиме шансу адаптувати своє значення b в залежності від g_a, тому що він повинен взяти на себе зобов'язання щодо значення g_b до того, як дізнається g_a. Те ж саме стосується генерації ключів між самозванцем і стороною B.
Використання хеш-зобов'язань в обміні DH обмежує зловмисника лише одним припущенням, щоб згенерувати правильну візуалізацію в своїй атаці, а це означає, що використання трохи більше 33 біт ентропії, представлених чотирма емодзі у візуалізації, достатньо, щоб зробити успішну атаку вкрай малоймовірною.

Посилання

- Viber Encryption Overview
- WhatsApp Security Whitepaper
- Skype security - Wikipedia
- End-to-End Encryption, Secret Chats
- End-to-End Encrypted Voice and Video Calls