Protokoły bezpieczeństwa aplikacji UNIFIC

Mateusz Goślinowski

Abstrakt

Ten dokument zawiera opis protokołu bezpieczeństwa aplikacji UNIFIC, przede wszystkim założenia dotyczące anonimowości, szyfrowania i odzyskiwania danych oraz sposób realizacji tych założeń przez oprogramowanie serwera i klienta.

Szczególny nacisk kładę na opisanie poszczególnych procedur, wymienianych danych i wykorzystywanych protokołów kryptograficznych.

Używane w tekście skróty:

1. **AES:** Advanced Encryption Standard, schemat szyfrowania kluczem symetrycznym. W projekcie będzie używany z kluczem 256 bitowym, w trybie CTR z wyrównaniem bajtów w standardzie PKCS #5 oraz w trybie GCM (bez wyrównania bajtów).
2. **DH:** Protokół wymiany klucza Diffiego - Hellmana. W projekcie używany w standardzie Elliptic Curve Diffie - Hellman. Używana krzywa eliptyczna to NIST P-521.
3. **HMAC:** Protokół autentykacji wiadomości, korzystający wewnętrznie z kryptograficznej funkcji hashującej (w projekcie to SHA-256) i ustalonego wcześniej klucza. W projekcie używany przez użytkowników do weryfikacji przychodzących wiadomości pod kątem ich autentyczności.
4. **PBKDF2:** Password-Based Key Derivation Function, protokół wyprowadzania klucza symetrycznego z podanego przez użytkownika hasła. Produkcja klucza odbywa się poprzez wielokrotne (w projekcie 25000-krotne) zastosowywanie funkcji pseudolosowej (w projekcie HMAC-SHA-256) do podanego przez użytkownika ciągu znaków, wzbogaconego o losowe ziarno. W efekcie produkowany jest 128- lub 256- bitowy klucz symetryczny AES. W poniższym dokumencie wyrażenie" zaszyfrowane hasłem" oznaczać będzie "zaszyfrowane algorytmem AES (jak wyżej), z kluczem wyprowadzonym z hasła użytkownika poprzez algorytm PBKDF2".

Spis treści

1 Założenia

1.1 Wymagania ogólne:

UNIFIC ma być w pełni bezpieczną i w pełni anonimową aplikacją do komunikowania się z najbliższymi znajomymi i rodziną. Ma za zadanie zapewnić dyskrecję wymiany danych przy jednoczesnej szybkości i niezawodności. Podstawowe założenia składają się więc na następujące główne punkty:

- **Poufność:** Serwer trzyma w bazie danych jedynie informacje niezbędne do autentykacji użytkowników i kopie zapasowe (zaszyfrowanych) wiadomości. W szczególności, nie powinien mieć możliwości odczytania lub odszyfrowania żadnych wiadomości oraz poznania żadnych informacji o użytkownikach. Protokół zakłada więc **szyfrowanie end-to-end**. Serwer jedynie *pośredniczy* w przesyłaniu danych między użytkownikami.
- **Bezpieczeństwo:** Wszelkie dane wymieniane między klientem a serwerem są szyfrowane zgodnie z nowoczesnymi standardami (AES, ECDH, HMAC-SHA256). W szczególności, aplikacja nie zakłada istnienia bezpiecznego łącza pomiędzy nią a serwerem. Nawet całość transkryptu komunikacji między klientami a serwerem nie ujawnia żadnych dodatkowych informacji na temat zawartości wiadomości, ani tożsamości rozmówców. Dodatkowo, dzięki wbudowanym procedurom autentykacji, podrobienie lub zmiana treści wiadomości również będzie niemożliwa.
- **Szybkość:** Wymiana wiadomości między użytkownikami odbywa się w czasie rzeczywistym, bez zauważalnego dodatkowego nakładu związanego z szyfrowaniem. Użyte protokoły powinny więc w szczególności pozwalać na szybkie szyfrowanie dużej ilości danych.

1.2 Wymagania szczegółowe bezpieczeństwa i funkcjonalności dla poszczególnych protokołów

1.2.1 Rejestracja

- Zewnętrzny przeciwnik nie może być w stanie szybko "zapchać" serwera, wysyłając w sposób ciągły zapytania o rejestrację nowych osób. W celu weryfikacji faktycznej chęci rejestracji stosowany może być krótki test proof of work. Można też serwerowo ograniczyć częstotliwość zapytań np. z jednego adresu IP.
- Po wykonaniu protokołu użytkownik i serwer powinni mieć ustalony wspólny sekret, służący do późniejszej komunikacji, oraz awaryjną drogę odzyskiwania konta. Podsłuchujący przeciwnik nie może mieć o nich żadnych istotnych informacji. Dodatkowo użytkownikowi zostaje przypisane unikalne *uid*.

1.2.2 Dodawanie nowych kontaktów

- Po wykonaniu protokołu obaj klienci powinni mieć ustalone (różne!) klucze do wzajemnej komunikacji oraz autentykacji wiadomości.
- Zewnętrzny przeciwnik nie powinien być w stanie zaingerować w protokół z zewnątrz, w szczególności podszyć się pod drugą stronę lub podsłuchać jakiekolwiek informacje. Po części pomaga w tym sama specyfikacja NFC, jako protokołu komunikacji o zaięgu kilkunastu cm.

- Serwer powinien być powiadomiony o zainicjowaniu komunikacji.
- Klucze służące do komunikacji powinny być przechowywane w bezpiecznym miejscu na urządzeniu, aby po przechwyceniu urządzenia przez osobę trzecią klucze pozostawały przed nim ukryte.

1.2.3 Wysyłanie wiadomości

- Przesyłane wiadomości muszą być nie do odszyfrowania przez kogokolwiek oprócz nadawcy i odbiorcy (w szczególności przez serwer, dostawcę telefonu czy przeciwnika podsłuchującego / ingerującego w komunikację między klientami a serwerem).
- Wiadomości muszą być weryfikowalne. Adresat musi mieć pewność, że nadawca rzeczywiście wysłał określoną wiadomość, i że wysłał ją dokładnie w takiej treści, w jakiej ona przyszła. Dodatkowo, powinny być niemożliwe ataki typu *replay*, w których przeciwnik wysyła do którejś ze stron kopię transkryptu poprzedniej komunikacji, aby została ponownie przesłana taka sama wiadomość.
- Serwer powinien przyjmować wiadomości jedynie pomiędzy połączonymi kontaktami parami użytkowników.

1.2.4 Przechowywanie danych

- Wszelkie informacje, które nie dotyczą wiadomości, takie jak status użytkownika, imię czy nazwisko (potrzebne do zainstalowania aplikacji na innym urządzeniu) będą przechowywane na serwerze w formie zaszyfrowanej hasłem użytkownika. Jednocześnie użytkownik może wyrazić również zgodę i chęć na przechowywanie na serwerze swoich kluczy, oczywiście również zaszyfrowanych hasłem. Ułatwi to przenoszenie aplikacji z telefonu na telefon, ale za cenę bezpieczeństwa, o czym użytkownik musi być explicite poinformowany.

1.2.5 Odzyskiwanie konta i kluczy

- W przypadku zgubienia hasła dostępu do konta, aplikacja powinna udostępniać metody odzyskiwania konta. Użytkownik, chcący odzyskać konto, powinien prawidłowo podać uprzednio ustalone dane odzyskiwania (np. email, pytanie pomocnicze etc).
- Podszycie się pod jakiegokolwiek użytkownika w celu uzyskania dostępu do jego konta poprzez system odzyskiwania musi być niemożliwe.
- W miarę możliwości serwer może, jak zostało to wspomniane powyżej, udostępniać określone miejsce w bazie danych, w którym użytkownicy mogą zapisywać swoje klucze, zaszyfrowane własnymi hasłami. W przypadku zagubienia hasła przez użytkownika, klucze te są nie do odzyskania poprzez serwer.
- Aplikacja powinna udostępniać możliwość renegocjacji lub przywrócenia kluczy z poszczególnymi rozmówcami w przypadku ich utraty (np. po reinstalacji aplikacji). Preferowanym sposobem jest ponowne połączenie z konkretnymi osobami, gdyż procedura ta przeprowadzana poprzez nieuczciwy serwer może potencjalnie prowadzić do ujawnienia kluczy serwerowi (serwer jako man-in-the-middle).

2 Implementacja

2.1 Rejestracja

Google Firebase udostępnia system kontrolowanej autentykacji i rejestracji użytkowników przy wykorzystaniu większości popularnych sposobów logowania. W aplikacji używana będzie rejestracja poprzez adres email, login i hasło. Każdemu użytkownikowi przypisywany jest podczas rejestracji na serwerze automatycznie identyfikator. Bezpieczeństwo tego protokołu opiera się na bezpieczeństwie wbudowanych funkcji Firebase.

2.2 Dodawanie nowych kontaktów

Dodawanie kontaktu jest bardzo prostą operacją. Osoby chcące dodać siebie do kontaktów, zbliżają telefony. Poprzez NFC i protokół DH użytkownicy ustalają klucze do komunikacji oraz do autentykacji, po czym powiadamiają serwer o dodaniu siebie nawzajem do listy swoich kontaktów. Po uzyskaniu pozytywnej odpowiedzi przez serwer, użytkownicy wymieniają poprzez serwer wiadomości próbne. Jeśli zakończy się to powodzeniem, dodawanie kontaktu uznaje się za zakończone.

Klucze powinny być przechowywane w bezpiecznym miejscu w urządzeniu tak, aby wszelkie próby wyekstrahowania ich po kradzieży telefonu były nieudane (np. wykorzystując wbudowany w system Android *keystore*).

2.3 Zmiana danych użytkownika

Użytkownik, chcący zmienić swoje dane, takie jak imię, nazwisko czy status, wysyła je w formie zaszyfrowanej hasłem do serwera. Jednocześnie do wszystkich znajomych wysyłana jest poprzez serwer specjalna wiadomość konfiguracyjna (która nie wyświetli się im w powiadomieniach), zawierająca w swojej treści nowe imię, nazwisko, status oraz link lub całość zdjęcia profilowego. Szczegóły na temat wysyłania danych do innych użytkowników poniżej.

2.4 Wysyłanie wiadomości

Implicite zakładam, że komunikacja między klientami a serwerem jest szyfrowana poprzez ustalone podczas rejestracji klucze. Niech sk będzie ustalonym między użytkownikami A a B kluczami do szyfrowania wiadomości, a mk kluczami do autentykacji. Oznaczmy treść wiadomości przez M, oraz niech $\mathrm{AES}_{key}(iv, m)$, $\mathrm{HMAC}_{key}(m)$ będą odpowiednimi funkcjami do szyfrowania i tagowania wiadomości.

1. Użytkownik A wybiera nieużyte wcześniej do komunikacji z B ID wiadomości i wyprowadza z niego wektor inicjalizacyjny IV, którego użyje do zaszyfrowania wiadomości poprzez AES. IV musi być całkowicie unikalne dla całości komunikacji z danym rozmówcą.

2. Użytkownik A przygotowuje do wysłania pakiet z wiadomością, który w niezmienionej formie zostanie dostarczony do B poprzez serwer. Składa się on z:

$$\{\mathrm{IV},\ \mathrm{ID},\ \mathrm{HMAC}_{mk}(IV \parallel ID \parallel \mathrm{AES}_{sk}(IV, M)),\ \mathrm{AES}_{sk}(IV, M)\}$$

   gdzie $a \parallel b$ jest konkatenacją a i b.

3. A wysyła do serwera paczkę. Serwer ją odszyfrowuje, sprawdza czy B jest w kontaktach A i wysyła do A informację, że wiadomość została wysłana. Serwer zapisuje paczkę na dysku, po czym powiadamia B, że jest do odebrania wiadomość (poprzez Firebase Cloud Messaging).

4. B odszyfrowuje paczkę. Następnie weryfikuje, czy czy HMAC zgadza się z resztą wiadomości oraz czy ID wiadomości nie pojawiło się już wcześniej. Jeśli wszystko jest w porządku, odszyfrowuje wiadomość i wysyła do serwera, że wiadomość została odczytana. Ten informuje o tym fakcie A.

2.5 Odzyskiwanie konta i kluczy

Może się tak zdarzyć, że użytkownik straci wszelkie swoje klucze, np. odinstalowując aplikację lub zmieniając telefon. W przypadku ponownej instalacji użytkownik może zadeklarować, że był już zarejestrowany i podać swój adres mailowy. Jeśli zgadza się on z adresem pewnego użytkownika, to zostaną na niego wysłane dane potrzebne aplikacji do ponownego zalogowania. Po ponownym połączeniu zostanie też przywrócony dostęp do (zaszyfrowanych) wiadomości.

Jednak będą one nie do odczytania, bo użytkownik po resecie konta nie ma też kluczy, które ustalił ze znajomymi. W tym przypadku istnieją trzy opcje:

- Użytkownik mógł skorzystać wcześniej z opcji zapisywania kluczy w chmurze UNIFIC. Będzie to miejsce, w którym składowane będą wszystkie klucze użytkownika, oczywiście zaszyfrowane uprzednio przez niego znanym tylko jemu hasłem.

- Załóżmy jednak, że użytkownik tego hasła nie pamięta. Wówczas może wygenerować pary (klucz publiczny, klucz prywatny) RSA i wysłać klucz publiczny do serwera, który poprosi jego znajomych, aby zaszyfrowali tym kluczem ich klucze symetryczne, i mu je odeśle. Następnie użytkownik odszyfruje te klucze swoim kluczem prywatnym.

- Załóżmy znów, że użytkownicy nie ufają jednak serwerowi (co, jeśli serwer sam sobie wygenerował pary kluczy i w ten sposób chce poznać klucze do wiadomości użytkowników?). Wówczas nie pozostaje nic innego, niż ponowne parowanie użytkowników poprzez zbliżenie telefonów. Aplikacja umożliwi mechanizm transportu klucza podczas zbliżenia, jeśli zgodzi się na to drugi użytkownik.

Co jednak, jeśli obaj użytkownicy naraz zgubią klucze? Wtedy wiadomości są bezpowrotnie tracone. Jest to cena za szyfrowanie end-to-end, ponieważ serwer w żadnej chwili nie posiada żadnej informacji dającej dostęp do wiadomości lub kluczy.