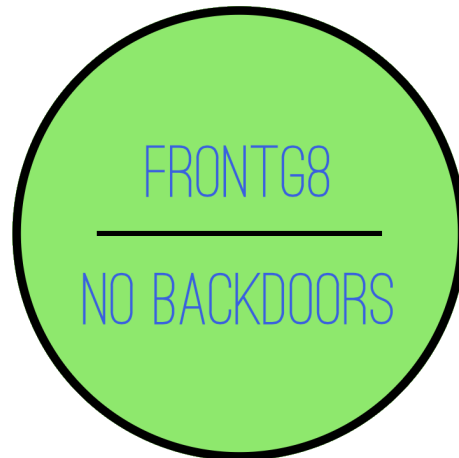




Projekt: frontg8

Crypto Design Dokument

Felix Morgner - fmorgner@hsr.ch
Tobias Stauber - tstauber@hsr.ch
Ueli Bosshard - ubosshar@hsr.ch



Änderungsgeschichte

Datum	Version	Änderung	Autor
08.04.2015	1.0		Tobias Stauber
27.04.2015	1.1	Kleine Änderungen an der Krypto	Tobias Stauber
04.05.2015	1.2	Ergänzungen und Erklärungen	Tobias Stauber



Inhalt

Änderungsgeschichte	2
Inhalt.....	3
1. Einführung	4
1.1 Zweck	4
1.2 Gültigkeitsbereich	4
1.3 Übersicht	4
2. Einführung	5
2.1 Datenspeicher auf Client	5
2.2 Datenspeicher auf Server	5
3. Abläufe.....	5
3.1 ECDH-Schlüssel erstellen	5
3.2 Symmetrische Schlüssel aushandeln	5
3.3 Schlüssel ableiten	5
3.4 Verschlüsseln.....	5
3.5 Entschlüsseln.....	5
3.6 Entscheidungen	6
4. Cryptoverfahren.....	6
4.1 Verwendung von Diffie Hellman	6
4.1.1 Key-Negotiation	6
4.2 Symmetrische Schlüssel	6
4.2.1 Session-Key	6
4.2.2 Cipher für symmetrische Ver- und Entschlüsselung	7
4.2.3 HMAC für das Signieren der Nachrichten	7
4.2.4 Verschlüsselung der Daten	7



1. Einführung

1.1 Zweck

Dieses Dokument zeigt die Krypto-Architektur für das Projekt frontg8 und ermöglicht es relevante Designentscheidungen nachzuvollziehen.

1.2 Gültigkeitsbereich

Das Dokument gilt für die Entwicklung der Software frontg8 während des SE2 Projekts im Frühlingssemester 2015.

1.3 Übersicht

Dieses Dokument erklärt die Verwendung von Kryptographie. Alle anderen Informationen zur Implementierung finden sich im „Software Architektur Dokument“.

2. Einführung

2.1 Datenspeicher auf Client

Beim Login muss der Nutzer seinen Username und sein Passwort eingeben. Von diesem Passwort wird mit PBKDF2(SHA-256) ein Schlüssel abgeleitet. Dabei wird der Username als Salt verwendet. Mit diesem Schlüssel werden die Contacts verschlüsselt. Wir haben uns hier für SHA-256 entschieden, weil sowohl die Sicherheit, wie auch der Rechenaufwand sich in einem vertretbaren Bereich befinden. Der Rechenaufwand ist auf einer 32Bit Maschine gut und auf einer 64Bit Maschine mit truncating sehr gut.

2.2 Datenspeicher auf Server

Beim Starten des Servers wird nach einem Passwort gefragt, das mittels PBKDF2- SHA256 zu einem symmetrischen Schlüssel der Länge 128Bit verarbeitet wird. Dieser Schlüssel wird eingesetzt um die Journaleinträge mittels AES128-CBC zu verschlüsseln.

3. Abläufe

3.1 ECDH-Schlüssel erstellen

Die Public-/Privatekeys werden auf der secp521r1 gewählt. Wir nutzen diese Kurve, weil die ECC-Bibliothek welche wir nutzen, noch nicht fähig ist "nicht NIST Kurven" zu nutzen.

3.2 Symmetrische Schlüssel aushandeln

Der Schlüssel, welcher aus dem ECDH-Handshake hervorgeht ist ein symmetrischer Schlüssel mit einer Länge von 512Bit.

3.3 Schlüssel ableiten

Die Schlüssel werden mittels KDF3 (ISO18033) vom Session-Schlüssel abgeleitet. Für den SK.C verwenden wir die ersten 256Bit des SK und für den SK.S verwenden wir die letzten 256Bit des SK. Hier kann durch zugeben von [OtherInfo] die Ableitung der Schlüssel manipuliert werden. So kann eine fortlaufende Folge von unterschiedlichen Schlüsseln ermöglicht werden. Das Aushandeln der erforderlichen Informationen dafür kann z.B. über den Server erfolgen.

3.4 Verschlüsseln

Der Contact erhält eine Data-Message, welche er mit dem SK.C verschlüsselt. Für die Verschlüsselung verwenden wir AES256-CTR. Den Counter initialisieren wir jeweils mit einem 4 Byte Zufallswert, welchen wir vorne an die Nachricht anhängen. Dies wurde notwendig, da ansonsten gleiche Nachrichten immer identifiziert werden konnten. Über die verschlüsselte Daten-Nachricht wird ein HMAC(SHA-256) gebildet. Als Schlüssel für die HMAC-Funktion dient der SK.S. Der HMAC wird an die verschlüsselte Daten-Nachricht angehängt. Danach wird die verschlüsselte Nachricht zurückgereicht.

3.5 Entschlüsseln

Der Contact erhält ein Blob. Zuerst wird der Blob geteilt, indem die vier header- Counter-Bytes und der HMAC-trailer abgeschnitten werden. Der HMAC wird mit dem verbleibenden Cryptoteil des Blobs gebildet und wenn die Beiden HMAC's übereinstimmen wird die verschlüsselte Nachricht abgespeichert, der verschlüsselte Cryptoteil entschlüsselt und die daraus entstandene Daten-Nachricht zurückgegeben.

3.6 Entscheidungen

- *Wir verwenden elliptische Kurven, da die notwendige Rechenleistung geringer ist als bei RSA.*
- *Wir verwenden HMAC, da dies uns sowohl das Prüfen der Integrität, wie auch der Authentizität ermöglicht.*
- *Wir verwenden SHA-256, da die Sicherheit ausreicht und die Geschwindigkeit sowohl auf 32Bit wie auch auf 64Bit annehmbar ist.*
- *Wir verwenden KDF3, weil es eine unseren Sicherheitsansprüchen genügende Schlüsselableitungsfunktion ist. Zudem kann durch das zufügen von optionalen Informationen die Schlüsselableitung pseudozufällig beeinflusst werden. Dadurch werden in Zukunft fortlaufende Schlüsselwechsel möglich.*
- *Wir haben uns für AES-256/CTR entschieden, da die Laufzeit nicht markant von AES-128 abweicht. Countermode wählten wir, damit wir nicht an 128Bit Blöcke gebunden sind.*
- *Da identische Nachrichten bei identischem Counter initial-value immer gleich aussehen würden, initialisieren wir den Counter mit einem 32Bit cryptographic random value. Den initial-value transportieren wir, indem wir ihn vor die Nachricht setzen.*
- *Wir haben uns für PBKDF2/SHA-256 mit 4096 Runden entschieden, da PBKDF1 keine Schlüssel die länger als 160Bit sind erzeugen kann.*
- *Als HASH-Funktion für die Identifizierung der Nachrichten verwenden wir SHA256, aus den selben Gründen wie oben genannt.*
- *Ein memory wipe kann unter Python nicht zuverlässig umgesetzt werden. Bei unseren Nachforschungen sind wir zwar auf einige lowlevel Methoden gestossen, allerdings sind die durch den Zugriff auf C nur schwer handhabbar.*
- *Da die Contact-Objekte serialisiert persistent gespeichert werden und diese die Schlüssel beinhalten, müssen diese verschlüsselt gespeichert werden.*
- *Die Nachrichten selbst werden nicht zusätzlich verschlüsselt.*

4. Cryptoverfahren

4.1 Verwendung von Diffie Hellman

4.1.1 Key-Negotiation

Asymmetrische Schlüssel (Curve 'secp521r1')

Da Bibliotheken wie Pycrypto oder pyelliptic auf OpenSSL zugreifen ist curve25519 nicht verfügbar. Ansonsten wäre diese unsere erste Wahl gewesen. Länge des ausgehandelten symmetrischen Schlüssels: 512Bit

4.2 Symmetrische Schlüssel

4.2.1 Session-Key

Länge: 512Bit

Session-Key.Crypto

Länge: 256Bit

Abgeleitet von SK mit HMAC(KDF3)

Session-Key.Signing

Länge: 256Bit

Abgeleitet von SK mit HMAC(KDF3)



4.2.2 Cipher für symmetrische Ver- und Entschlüsselung

AES-256 im CTR-Mode

Schlüssel: Session-Key.Crypto

Counter-Initial-value: Cryptographical random 4 byte int

4.2.3 HMAC für das Signieren der Nachrichten

HMAC(SHA-256)

Schlüssel: Session-Key.Signing

4.2.4 Verschlüsselung der Daten

Symmetrischer Schlüssel

Länge: 256Bit

Abgeleitet von einem Passwort und dem Username mit PBKDF2(SHA-256)

Nutzt auch eine AES-256 CTR Cipher