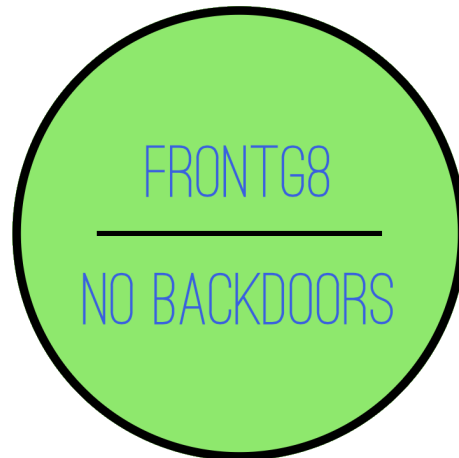




Projekt: frontg8

Domainanalyse

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



Änderungsgeschichte

Datum	Version	Änderung	Autor
12.03.15	1.0		Ueli Bosshard
14.03.15	1.1	Komplette Überarbeitung	Ueli Bosshard
19.03.15	1.2	Nochmalige komplette Überarbeitung	Ueli Bosshard
21.03.15	1.3	Korrekturen gemäss Sitzungsprotokoll	Ueli Bosshard
26.05.15	1.4	Kleine Korrekturen	Ueli Bosshard



Inhalt

Änderungsgeschichte	2
Inhalt.....	3
1. Einführung	6
1.1 Zweck	6
1.2 Gültigkeitsbereich	6
1.3 Referenzen	6
1.4 Übersicht	6
2. Systemübersicht	7
2.1 Context.....	7
2.2 Problem	7
2.3 Lösung	7
2.3.1 Server.....	7
2.3.2 Client.....	8
2.3.3 Referenzen:.....	8
3. Domain Modell	9
3.1 Strukturdiagramm	9
3.2 Klassen	10
3.2.1 Message	10
3.2.2 KeyRevocationMessage	10
3.2.3 BlacklistMessage	10
3.2.4 EncryptedMessage	10
3.2.5 MessageRequestMessage	10
3.2.6 NotifyMessage.....	10
3.2.7 EncryptedControlMessage	10
3.2.8 ControlMessage	10
3.2.9 KeyExchange	11
3.2.10 SessionAbortion	11
3.2.11 EncryptedControlMessage	11
3.2.12 DataMessage.....	11
3.2.13 Contact.....	11
3.2.14 PublicKey.....	11
3.2.15 Session	11
3.3 Wichtige Konzepte	12
3.3.1 Messages	12
3.3.2 Blacklist	12
3.3.3 Sessionhandling	13

3.3.4 Kontakte, Session, PublicKey	13
4. Systemsequenzdiagramme	14
4.1 Übersicht Usecases	14
4.1.1 User/Client Usecases	14
4.1.2 User/User Usecases	14
4.1.3 Admin/Server und Client/Server Usecases	15
4.1.4 Abusecases.....	15
4.2 User/Client Usecases.....	16
4.2.1 UC00_C: Client starten.....	16
4.2.2 UC01_C: Nachricht an XY senden	22
4.2.3 UC03_C: Kontakt verwalten.....	27
4.2.4 UC04_C: Gruppen verwalten	28
4.2.5 UC05_C: Sperren von Publickeys	29
4.3 User/User Usecases	31
4.3.1 UC00_N: Konfiguration editieren.....	31
4.3.2 UC01_N: Publickeys austauschen	32
4.4 Admin/Server und Client/Server Usecases	33
4.4.1 UC00_S: Server starten	33
4.4.2 UC01_S: Verbindung annehmen.....	34
4.4.3 UC02_S: Clientsession bearbeiten	35
4.5 Abusecases.....	41
4.5.1 AC00: DDOS-Attack on Server.....	41
4.5.2 AC01: Senden von Nachrichten mit defektem MAC	41
4.5.3 AC02: Manipulieren von Blacklist	41
5. Systemoperationen.....	42
5.1 Contract C01: Schlüsselaustausch	42
5.2 Contract C02: Kontakt hinzufügen	42
5.3 Contract C03: SendMSG	42
5.4 Contract C04: ReadMSG.....	42
6. Requirements	43
6.1 Funktionale Anforderungen	43
6.1.1 Netzwerkbasiert.....	43
6.1.2 Asynchrone Kommunikation.....	43
6.1.3 Symmetrische Verschlüsselung.....	43
6.2 Nichtfunktionale Anforderungen	43
6.2.1 Performance	43
6.2.2 Geheimhaltung	43
6.2.3 Wahrung der Privatsphäre.....	43



6.2.4 Kryptografische Stärke	44
6.2.5 Qualität	44
6.3 Constraints	45
7. Protokollspezifikation	46
7.1 Äusseres Nachrichtenformat.....	46
7.1.1 Beschreibung der Felder	46
7.2 Nachrichten-Typen.....	46
7.2.1 MSGTypeEncrypted - Verschlüsselte Nachricht.....	46
7.2.2 MSGTypeBlacklist - Blacklist Nachricht	47
7.2.3 MSGTypeRevocation - Revozierungs Nachricht.....	47
7.2.4 MSGTypeNotification - Benachrichtigungs Nachricht.....	47
7.2.5 MSGTypeMessageRequest - Nachrichten Anforderungs Nachricht	47

1. Einführung

1.1 Zweck

Dieses Dokument zeigt das Domainmodell für das Projekt frontg8.

1.2 Gültigkeitsbereich

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

1.3 Referenzen

Alle hier aufgeführten Informationen wurden im Redmine erarbeitet und sind unter <https://redmine.frontg8.ch/redmine> im Wiki bzw. unter Dokumente zu finden.

1.4 Übersicht

Als erstes geben wir eine Systemübersicht, die einen Überblick verschaffen soll. Nachfolgend finden sich die Strukturdiagramme für den Client bzw. für den Server von frontg8. Ausserdem sind einige Gedanken dazu im Kapitel „Wichtige Konzepte“ festgehalten. Anschliessend folgen die Kapitel mit den Systemsequenzdiagrammen, den Requirements und die Protokollspezifikation.

2. Systemübersicht

2.1 Context

Alice und Bob möchten miteinander kommunizieren. Als Kommunikationsweg möchten sie gerne das Internet benutzen, da es praktisch weltweit und zu jeder Zeit verfügbar ist. Für beide steht die Geheimhaltung und die Wahrung ihrer Anonymität gegenüber dritten im Vordergrund.

2.2 Problem

Alice und Bob erwägen verschiedene existierende Systeme zum Austausch von Nachrichten:

- [WhatsApp](#): Dieses System verfügt (noch) nicht über Ende-zu-Ende-Verschlüsselung [heise1etoe] was bereits die erste Anforderung (Geheimhaltung) verletzt. Des Weiteren ist WhatsApp dafür bekannt in Meta-Daten, wie MAC-Adressen und IMEIs, zu sammeln [daten1whda], was Anforderung zwei (Anonymität) verletzt. Erschwerend kommt für sie hinzu, dass WhatsApp keine OpenSource-Software ist, und sie somit nicht überprüfen können, ob das System Design-Schwächen enthält bzw. sie keine Möglichkeit haben solche Schwächen zu beheben.
- [TextSecure](#): TextSecure verwendet starke Ende-zu-Ende-Verschlüsselung [frosch1hsit] was bereits Anforderung eins erfüllt. Jedoch ist Anforderung zwei auch bei TextSecure nicht erfüllt, da es standardmässig die Übertragung von Telefonbuchkontakten [psw1tss] verwendet um eine initiale Kontaktaufnahme durchzuführen.
- [Threema](#): Auch Threema verwendet starke Ende-zu-Ende-Verschlüsselung [threema1wp]. Doch auch Threema verwendet Telefonbuchkontakte als Möglichkeit zur Auffindung von Kommunikationspartnern [threema1wp], was wiederum Anforderung zwei verletzt.

Keines dieser Systeme kann gleichzeitig die Kernanforderungen von Alice und Bob erfüllen.

2.3 Lösung

Frontg8 fokussiert sich auf die Lösung der oben erwähnten Anforderungen. Die Benutzer des Systems sind jeweils im Besitz eines Public-Private-Keypairs welches sie zur gegenseitigen Identifikation und zum Erarbeiten eines gemeinsamen Schlüssels verwenden. Momentan wird der Schlüsselaustausch nicht über den Server durchgeführt. Alice und Bob müssen ihre Schlüssel über einen anderen Weg (Out-of-Band) austauschen. Die empfohlene Methode dafür ist es, sich physisch zu treffen. Andere Möglichkeiten, wie zum Beispiel der Austausch über einen anderen sicheren Kanal (z.B. PGP-Mail), werden nicht ausgeschlossen, erfordern jedoch die Sicherstellung des Vertrauens in das verwendete Austausch-System.

Beide Clients ermitteln mittels ECDH ein gemeinsames Geheimnis und erzeugen daraus mittels PRF ein gemeinsamen symmetrischen Verschlüsselungs-Schlüssel und einen Schlüssel für den MAC.

2.3.1 Server

Frontg8 ist ein Client-Server-System welches quelloffen ist, und somit in Eigenregie betrieben werden kann. Die Serverapplikation sammelt ausser der IP-Adresse, welche zur Etablierung der Verbindung benötigt wird, im regulären Betrieb keine Daten und auch die IP-Adresse wird nur temporär im Speicher gehalten. Der Server ist für den Transport der Nachrichten und das Management von Revozierungseinträgen verantwortlich. Der Server verwaltet eine "Blacklist" deren Einträge die Fingerprints von widerrufenen Public-Keys sind.

Der Server bewahrt die eingereichten Nachrichten in einem Puffer-Speicher für einen einstellbaren Zeitraum (Standard ist 3 Tage) auf und entfernt sie anschliessend. Es wird dem Server-Betreiber empfohlen die mitgelieferten Skripten zur Erzeugung des Pufferspeichers zu verwenden. Die Skripte

erzeugen einen nicht permanenten Speicherbereich, womit verhindert werden soll, dass die Nachrichten jemals in persistenten Speicher geschrieben werden. Diese Massnahme soll verhindern, dass die verschlüsselten Nachrichten leicht zum Zwecke einer Offline-Analyse entwendet werden. Der Server benachrichtigt bei einer eingehenden Nachricht alle momentan mit ihm verbundenen Clients über den Eingang. Wie momentan nicht verbundene Clients die Nachrichten empfangen, ist im Abschnitt Client erläutert.

2.3.2 Client

Der Client ist für die Aufbewahrung der gemeinsamen Geheimnisse, sowie des privaten Geheimnisses verantwortlich. Des Weiteren speichert der Client die für ihn bestimmten Nachrichten für einen konfigurierbaren Zeitraum in verschlüsseltem Zustand ab, bevor er sie endgültig löscht. Da alle Clients alle Nachrichten bekommen, muss der Client feststellen können, ob die Nachricht für ihn bestimmt ist, ohne zu versuchen, die möglicherweise lange Nachricht zu entschlüsseln. Zu diesem Zweck ist jeder Nachricht ein HMAC angehängt. Dies ermöglicht es dem empfangenden Client einzig den HMAC zu prüfen. Ist der Client in der Lage die Gültigkeit des HMACs zu bestätigen, ist erstens mit grosser Sicherheit die Nachricht für diesen Client bestimmt und zweitens geklärt von welchem Absender sie stammt. Kann der empfangende Client den HMAC nicht verifizieren, verwirft er die Nachricht.

Wie zuvor beschrieben erhält der Client eine Benachrichtigung für jede eingegangene Nachricht, solange er eine aktive Netzwerkverbindung zum Server besitzt. Falls der Client momentan nicht über eine Verbindung zum Server verfügt, wird bei der nächsten Etablierung einer Verbindung vom Client eine Message-Request-Message an den Server gesendet. Diese Nachricht enthält den Hash der letzten vom Client empfangenen Nachricht. Der Client erhält als Antwort vom Server eine Notify-Message, welche die Anzahl der seither eingegangenen Nachrichten, sowie die Nachrichten enthält.

Die Revozierung von Public-Keys wird mittels Key-Revocation-Messages durchgeführt und ist endgültig. Zum Zeitpunkt der Revozierung muss der revozierende Client seinen Public-Key dem Server gegenüber preisgeben und somit ist zu diesem Zeitpunkt eine Zuordnung des Public-Keys zur momentanen IP-Adresse des revozierenden Clients möglich. Der Aufbau der Key-Revocation-Message ist im Kapitel Protokollspezifikation erläutert. Sobald ein Client eine Revozierungsnachricht gesendet hat, trägt der Server den revozierten Schlüssel in seine Blacklist ein und benachrichtigt alle Clients über den neuen Eintrag. Jeder Client fragt beim Start den Server nach der aktuellen Blacklist um sicher zu gehen, auf dem neusten Stand zu sein.

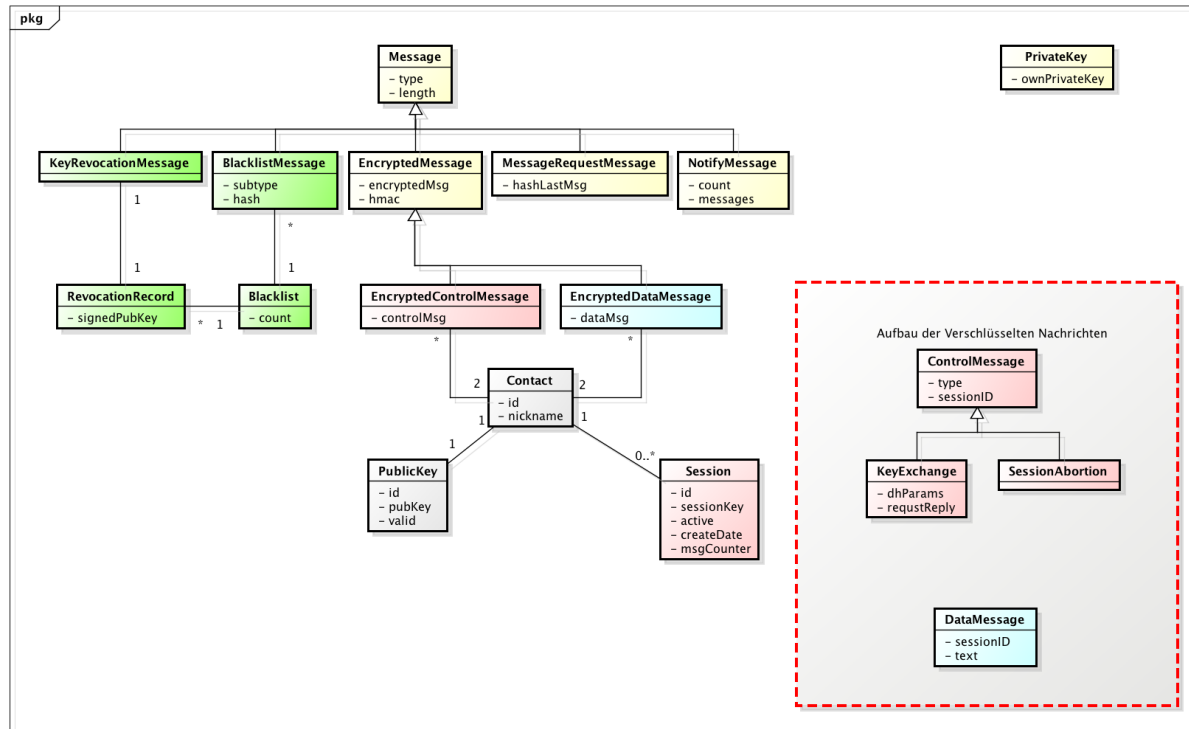
2.3.3 Referenzen:

- [heise1etoe] WhatsApp bekommt Ende-zu-Ende-Verschlüsselung - <http://www.heise.de/newsticker/meldung/WhatsApp-bekommt-Ende-zu-Ende-Verschlueselung-2459903.html>
- [daten1whda] WhatsApp und Datenschutz – Antworten auf die wichtigsten Fragen - <https://www.datenschutzbeauftragter-info.de/whatsapp-und-datenschutz-antworten-auf-die-wichtigsten-fragen/>
- [psw1tss] TextSecure: die quelloffene, verschlüsselte Alternative - <https://www.psw-group.de/blog/textsecure-die-quelloffene-verschlueselte-alternative/1223>
- [frosch1hsit] "How Secure is TextSecure?" - <https://eprint.iacr.org/2014/904.pdf>
- [threema1wp] Threema Cryptography Whitepaper - https://threema.ch/press-files/cryptography_whitepaper.pdf

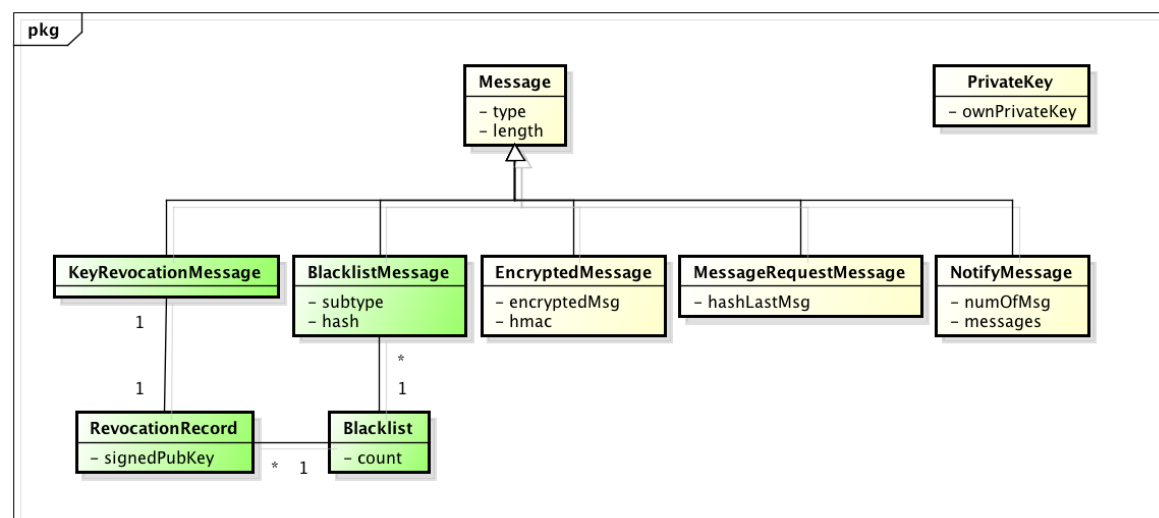
3. Domain Modell

3.1 Strukturdiagramm

Client



Server



Legende:

- grüne Klasse: Blacklisting
- rote Klasse: Sessionhandling
- blaue Klasse: Nutzdatenübermittlung
- graue Klasse: Informationen über Kommunikationspartner

3.2 Klassen

3.2.1 Message

Attribut	Typ	Beschreibung
type	Enum	Typ der Nachricht
Length	Ganzzahl	Länge der Nachricht

3.2.2 KeyRevocationMessage

Attribut	Typ	Beschreibung
signedPublicKey	Byte[]	Public-Key und Unterschrift darüber

3.2.3 BlacklistMessage

Attribut	Typ	Beschreibung
signedBlacklist	Byte[]	Blacklist Einträge
count	Ganzzahl	Anzahl Einträge
subtype	Enum	Subtyp der Nachricht
hash	Byte[]	hash der Blacklist

3.2.4 EncryptedMessage

Attribut	Typ	Beschreibung
encryptedMsg	Byte[]	Verschlüsselte Nachricht
hmac	Byte[]	MAC über die Nachricht

3.2.5 MessageRequestMessage

Attribut	Typ	Beschreibung
hashLastMsg	Byte[]	Hash der letzten empfangen Nachricht

3.2.6 NotifyMessage

Attribut	Typ	Beschreibung
count	Ganzzahl	Anzahl Nachrichten im Bundle
messages	Message[]	Nachrichten

3.2.7 EncryptedControlMessage

Attribut	Typ	Beschreibung
controlMsg	ControlMessage	Verschlüsselte Steuer-Nachricht

3.2.8 ControlMessage

Attribut	Typ	Beschreibung
type	Enum	Subtyp der Message
sessionId	Byte[]	UUID der Session

3.2.9 KeyExchange

Attribut	Typ	Beschreibung
dhParams	Byte[]	Public Key
requestReply	Enum	Angabe ob request oder reply

3.2.10 SessionAbortion

Attribut	Typ	Beschreibung
sessionID	Byte[]	UUID der Session

3.2.11 EncryptedControlMessage

Attribut	Typ	Beschreibung
dataMsg	DataMessage	Verschlüsselte Daten-Nachricht

3.2.12 DataMessage

Attribut	Typ	Beschreibung
sessionID	Byte[]	UUID der Session
text	String	Text der Nachricht

3.2.13 Contact

Attribut	Typ	Beschreibung
id	Byte[]	UUID des Kontakts
nickname	String	Angezeigter Name für einen Kontakt

3.2.14 PublicKey

Attribut	Typ	Beschreibung
id	Byte[]	Fingerprint des PubKeys
pubKey	Byte[]	Der Public Key
valid	Bool	Flag ob revoziert oder nicht

3.2.15 Session

Attribut	Typ	Beschreibung
id	Byte[]	UUID der Session
sessionKey	Byte[]	Symmetrischer Schlüssel der Session
active	Bool	Flag ob aktiv oder nicht
createDate	Date	Erstellungsdatum der Session
msgCounter	Ganzzahl	Anzahl Nachrichten in der Session

3.3 Wichtige Konzepte

3.3.1 Messages

Zwischen Server und Client werden verschiedene Arten von Messages ausgetauscht. Wir unterscheiden folgende Typen:

3.3.1.1 KeyRevocationMessage

Die KeyRevocation-Message enthält einen Public-Key, dessen Blob mit dem Private-Key unterschrieben ist, womit gegenüber dem Server gezeigt wird, dass der Absender im Besitz dieses Schlüssels ist. Der Server setzt dann den Schlüssel auf seine Blacklist.

3.3.1.2 BlacklistMessage

Der Client entscheidet beim Verbindungsaufbau, ob er bereits im Besitz der aktuellen Blacklist ist. Falls dies nicht der Fall ist, so kann er einen Request an den Server stellen, der darauf die aktuelle Blacklist ausliefert.

3.3.1.3 EncryptedMessage

Alle Nachrichten vom Typ Encrypted-Messages werden vom Server nur zwischengespeichert. Ausschliesslich die beteiligten Clients wissen, wer mit wem was austauscht. Kann ein Client eine Encrypted-Msg entschlüsseln, so kann er auch entscheiden, ob es sich um eine Control-Msg oder um eine Data-Msg handelt.

3.3.1.4 MessageReqMessage

Die MessageRequest-Message dient dazu, den Server nach allen Nachrichten seit dem letzten Abruf zu fragen.

3.3.1.5 NotifyMessage

Die Notify Message gibt an, wie viele Nachrichten geliefert werden und liefert zugleich die betreffenden Encrypted Messages.

3.3.1.6 ControlMessage

Die Control-Message dient zur Aushandlung des symmetrischen Session-Keys zwischen zwei Clients. Diese Funktionalität wird im Rahmen des SE2-Projekts noch nicht umgesetzt.

3.3.1.7 DataMessage

Die Data-Message schliesslich ist für die Übertragung der Text-Nachricht zuständig. Die Daten werden dabei mit dem symmetrischen Session-Key verschlüsselt.

3.3.2 Blacklist

Die Blacklist dient dazu den Public-Key von Endgeräten als ungültig auszuweisen, falls dieser vom Benutzer als nicht mehr vertrauenswürdig gemeldet wird. Dazu muss der Benutzer eine KeyRevocationMessage an den Server senden, worauf dieser den Schlüssel in seine Blacklist aufnimmt. Jeder andere Benutzer kann dann anhand der Blacklist überprüfen, ob die verwendeten Zertifikate noch gültig sind.



3.3.3 Sessionhandling

Session Handling wird in dieser Version des Systems noch nicht durchgeführt. Zu einem späteren Zeitpunkt wird es das gezielte Abbauen einer Session, sowie das erneuern des Session-Schlüssels erlauben.

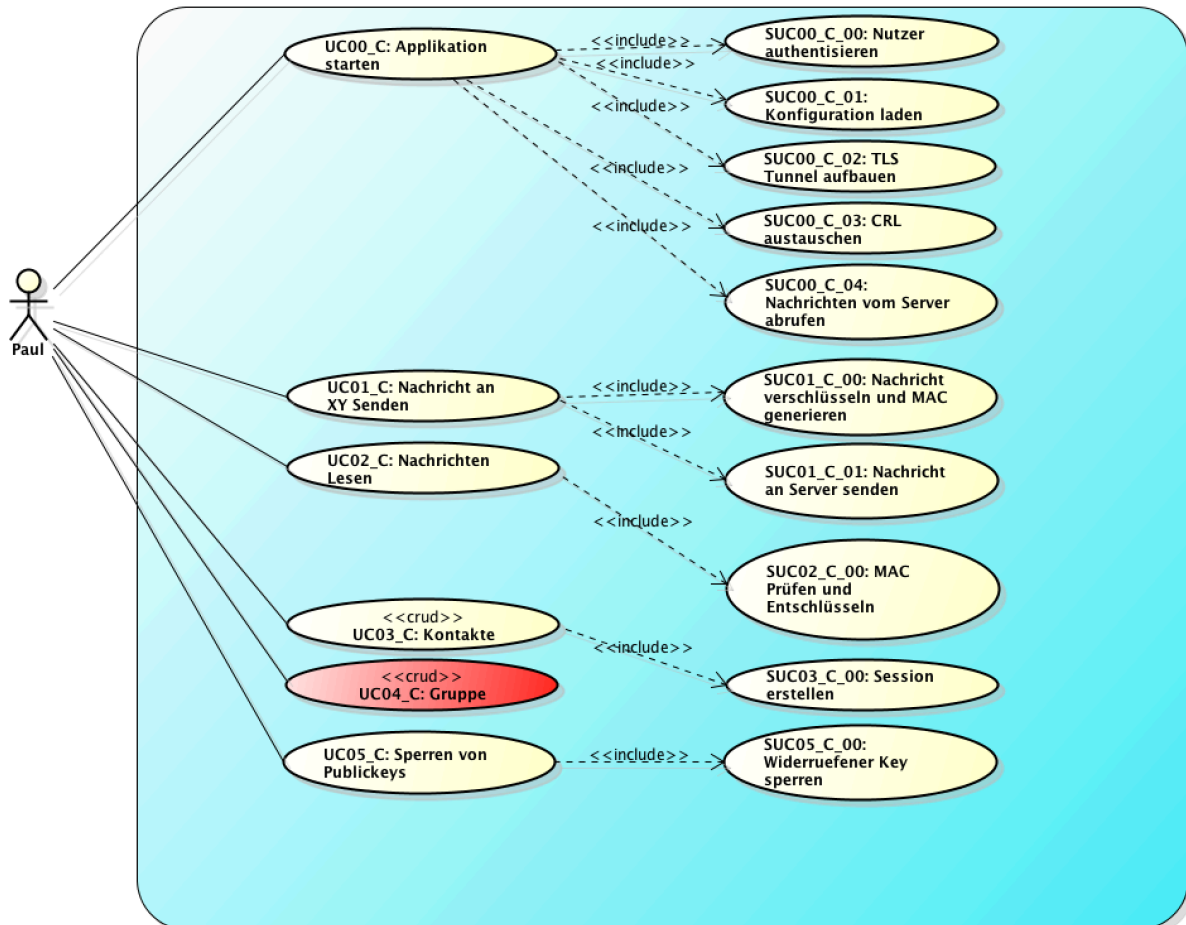
3.3.4 Kontakte, Session, PublicKey

Der Client besitzt ein Adressbuch mit seinen Kontakten. Zu jedem seiner Kontakte speichert der Client den Public-Key des jeweiligen Kontakts, aus dem er dann den Session Key ableiten kann.

4. Systemsequenzdiagramme

4.1 Übersicht Usecases

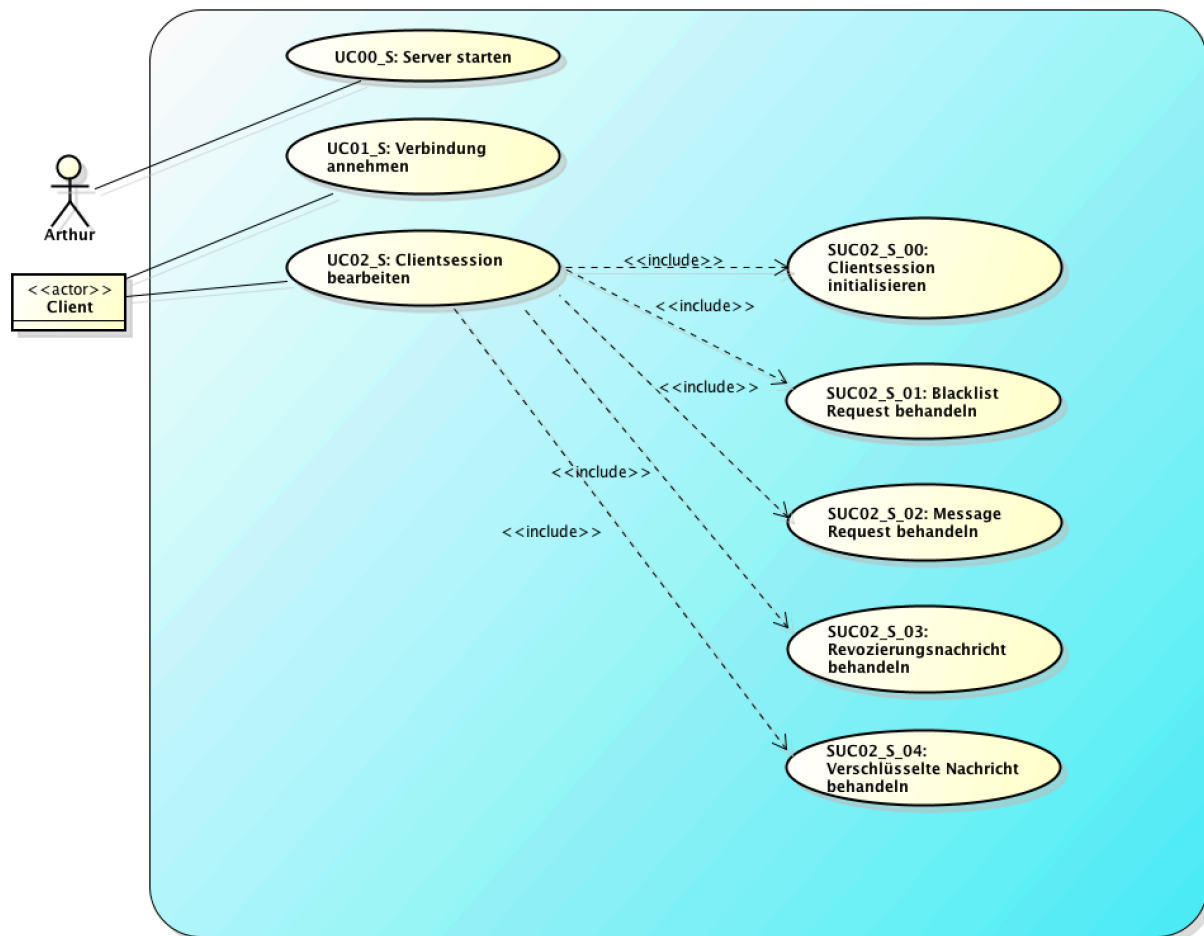
4.1.1 User/Client Usecases



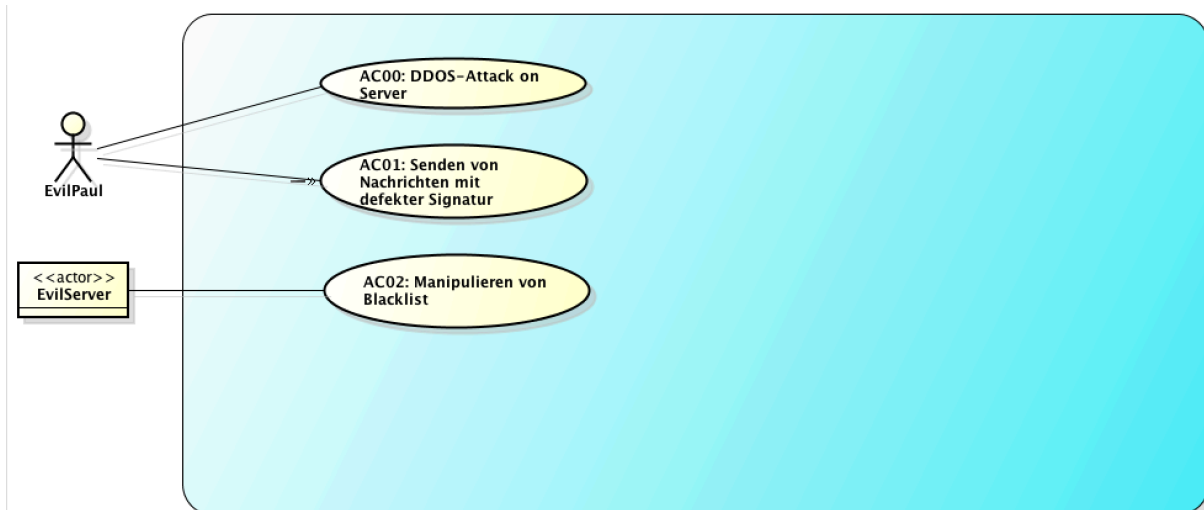
4.1.2 User/User Usecases



4.1.3 Admin/Server und Client/Server Usecases



4.1.4 Abusecases



4.2 User/Client Usecases

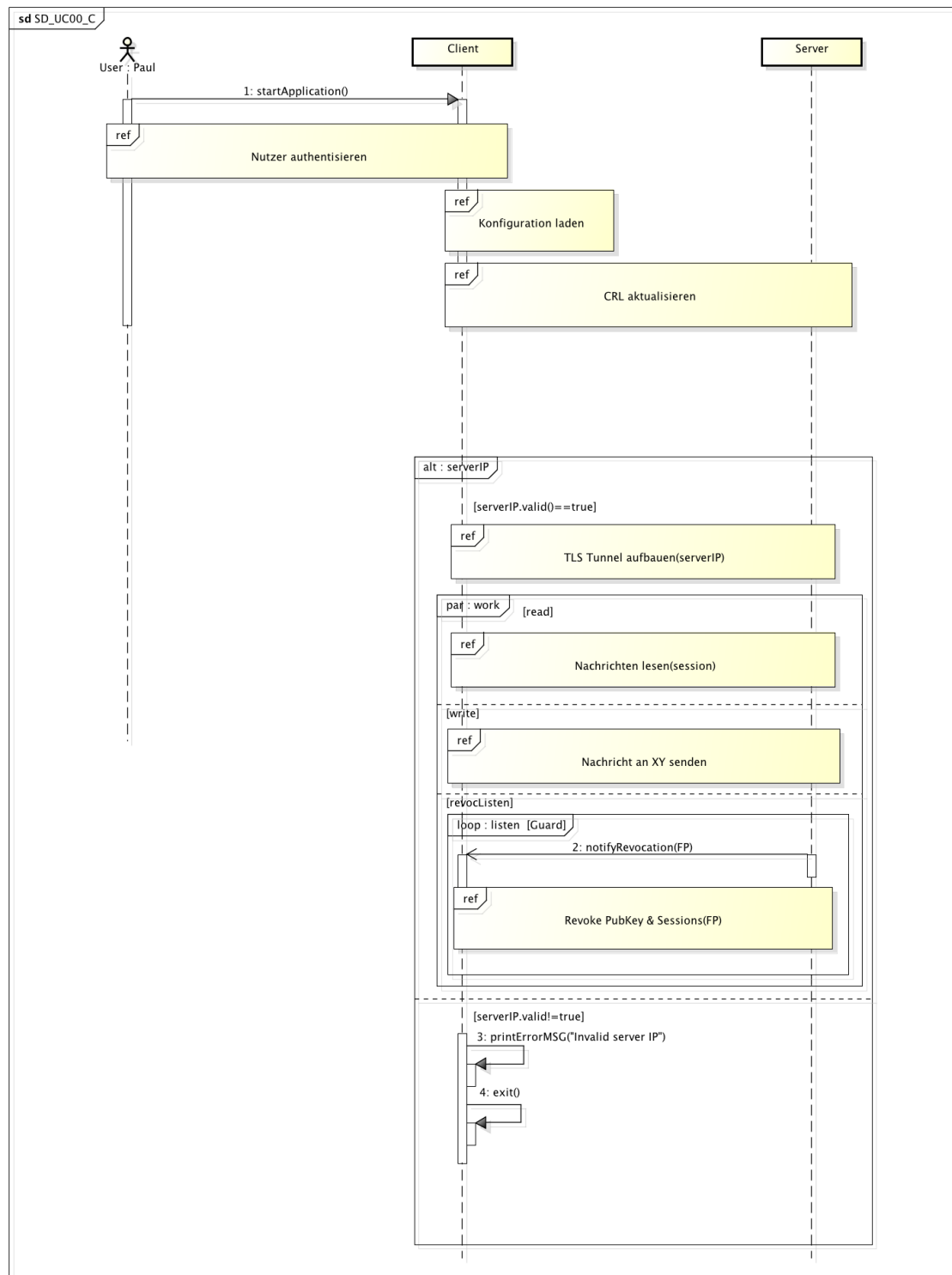
4.2.1 UC00_C: Client starten

Title [Client starten]

Primary Actor: [Chatteilnehmer]

Scope: Client

(Story): SUC00_C_00, SUC00_C_01, SUC00_C_02, SUC00_C_03, SUC00_C_04, danach müssen folgende Funktionen parallel ausgeführt werden (CRL SUC06_C_00, UC01_C, UC02_C)



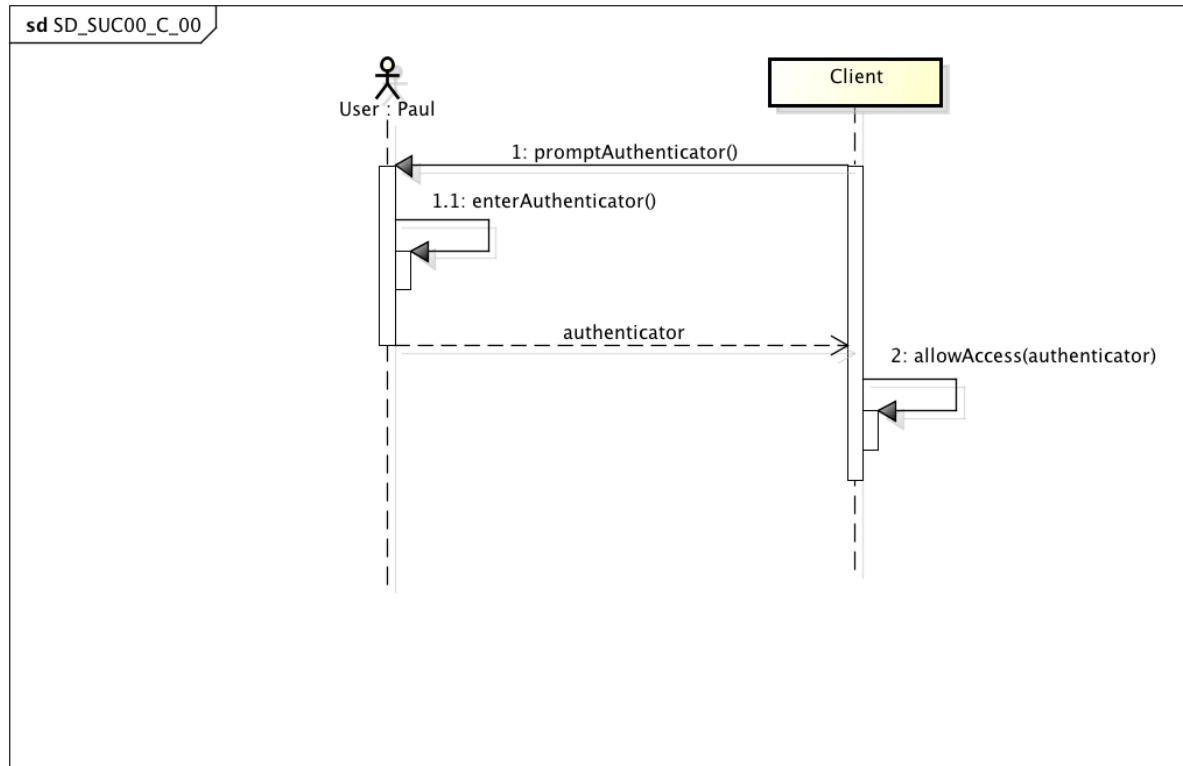
4.2.1.1 SUC00_C_00: Nutzer authentisieren

Title [Nutzer authentisieren]

Primary Actor: [Client]

Scope: Client

(Story): C fordert N auf sich zu authentisieren. Nachdem sich N authentisiert hat, werden die MSG's entschlüsselt.



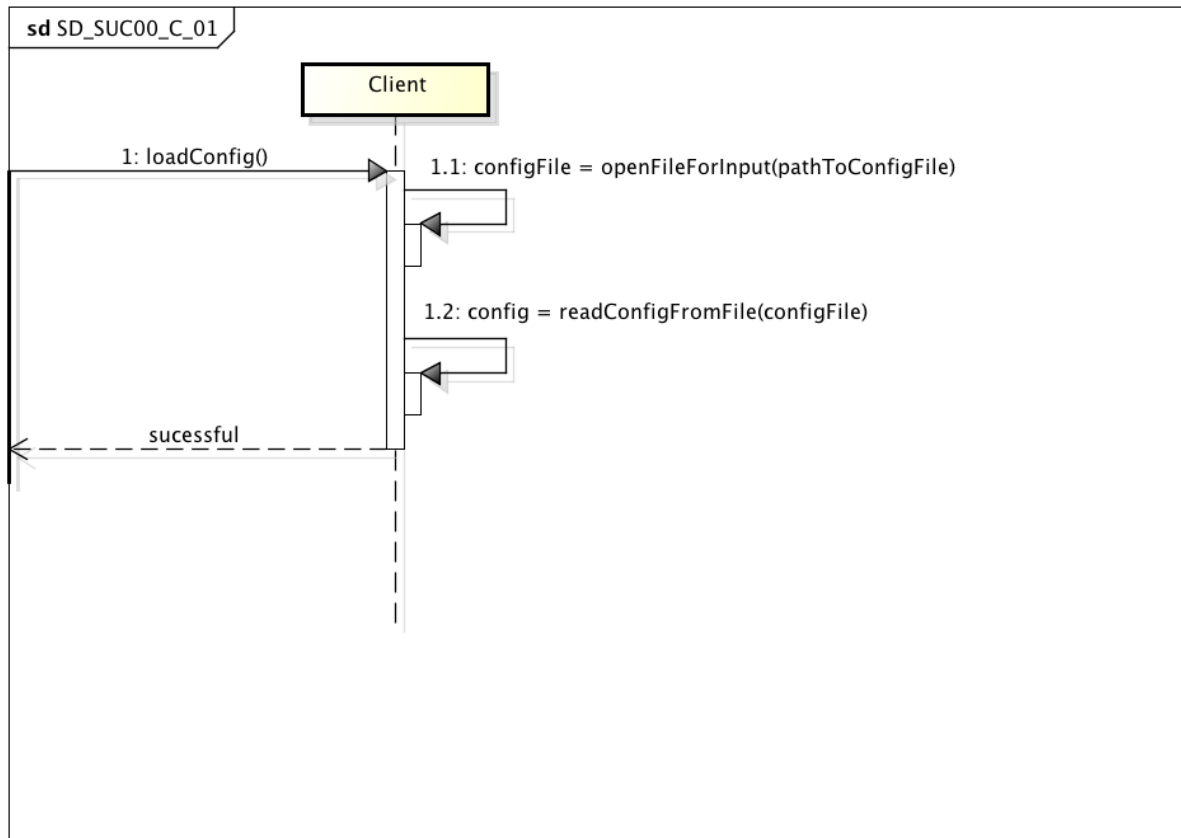
4.2.1.2 SUC00_C_01: Konfiguration laden

Title [Konfiguration laden]

Primary Actor: [Client]

Scope: Client

(Story): C lädt seine Schlüssel, die Verbindungsdaten (serverIp, das Zertifikat von S, usw.), den letzten MSGHash und den letzten Blacklist-Hash.



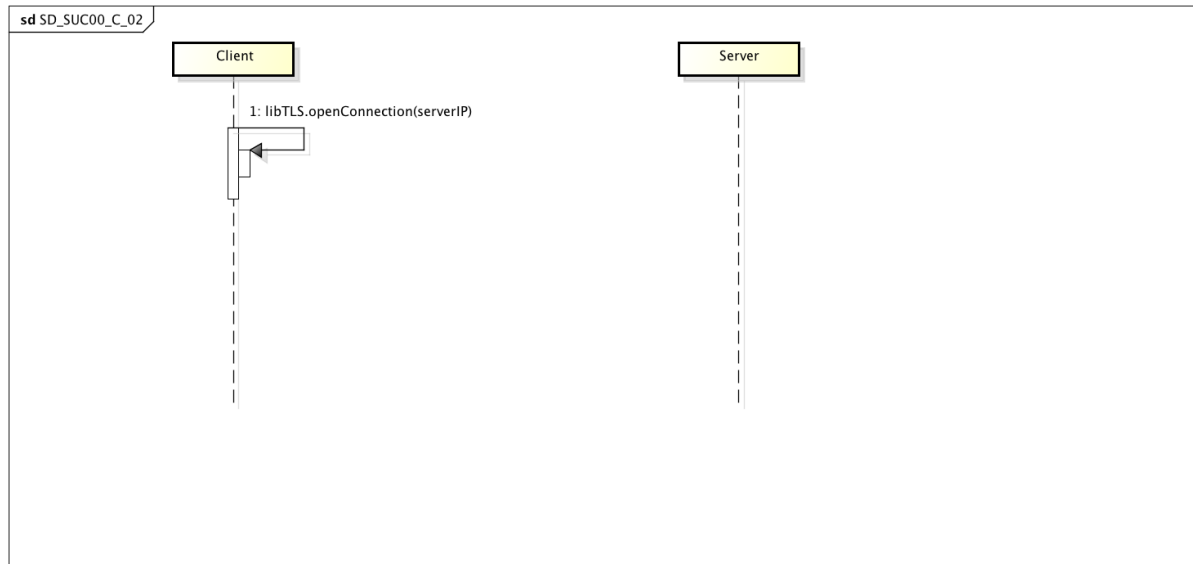
4.2.1.3 SUC00_C_02: TLS Tunnel aufbauen

Title [TLS Tunnel aufbauen]

Primary Actor: [Client]

Scope: Client / Server

(Story): C baut TLS-Tunnel zu S auf (Dabei wird nur dem Serverzertifikat vertraut, nicht aber dem Rootcert, Intermediatecert usw.)



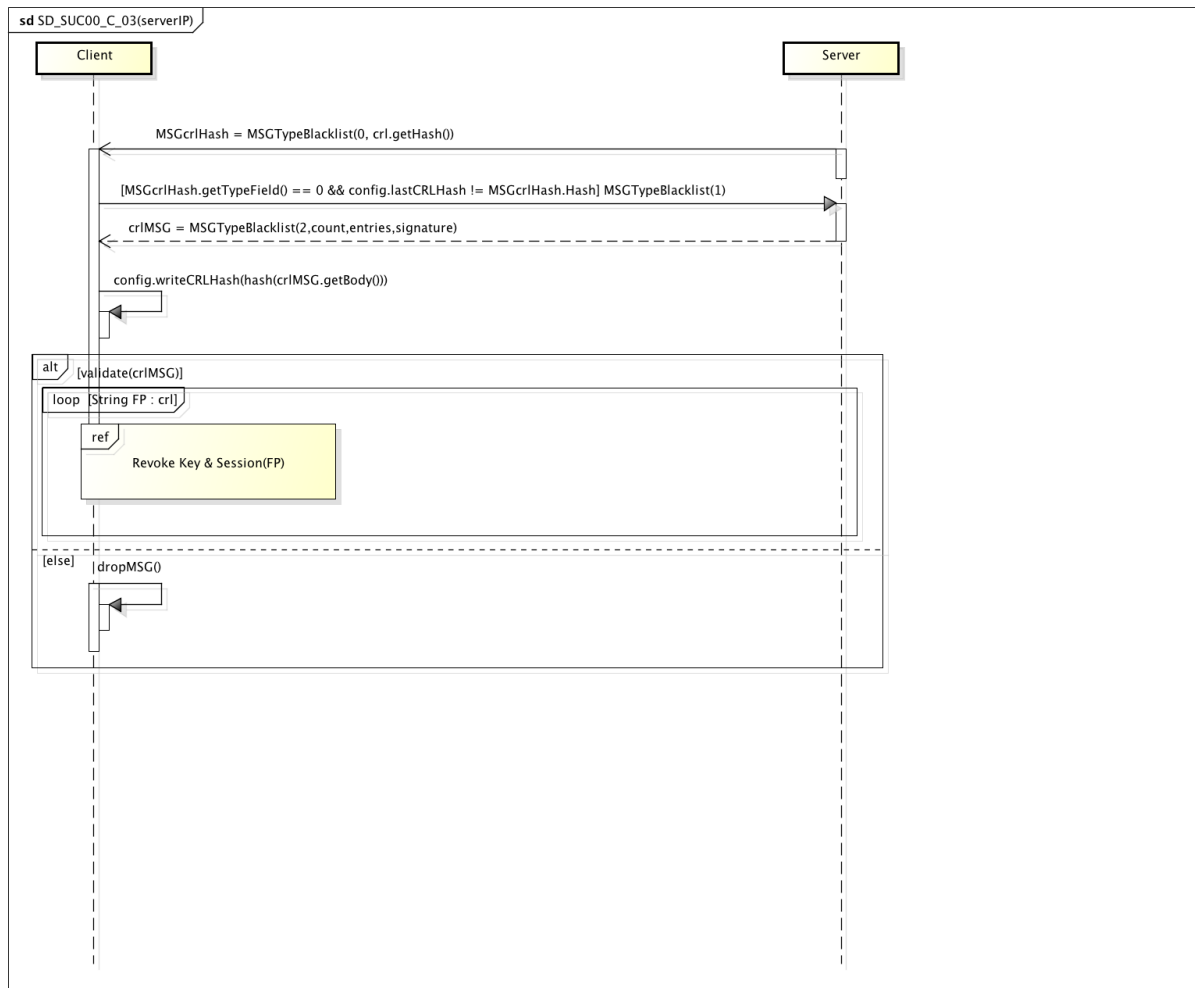
4.2.1.4 SUC00_C_03: Blacklist austauschen

Title [Blacklist austauschen]

Primary Actor: [Client / Server]

Scope: Client / Server

(Story): Nach dem Aufbau des TLS-Tunnels sendet S den Hash seiner Blacklist an C. Wenn die Hashes übereinstimmen, folgt darüber keine weitere Kommunikation. Wenn die Hashes nicht übereinstimmen, sendet C ein Blacklist-Request an S, welcher mit einer BlacklistMessage, welche die aktuelle Blacklist enthält, antwortet. Dann wird für jeden Eintrag in der Blacklist SUC07_C_00 aufgerufen.



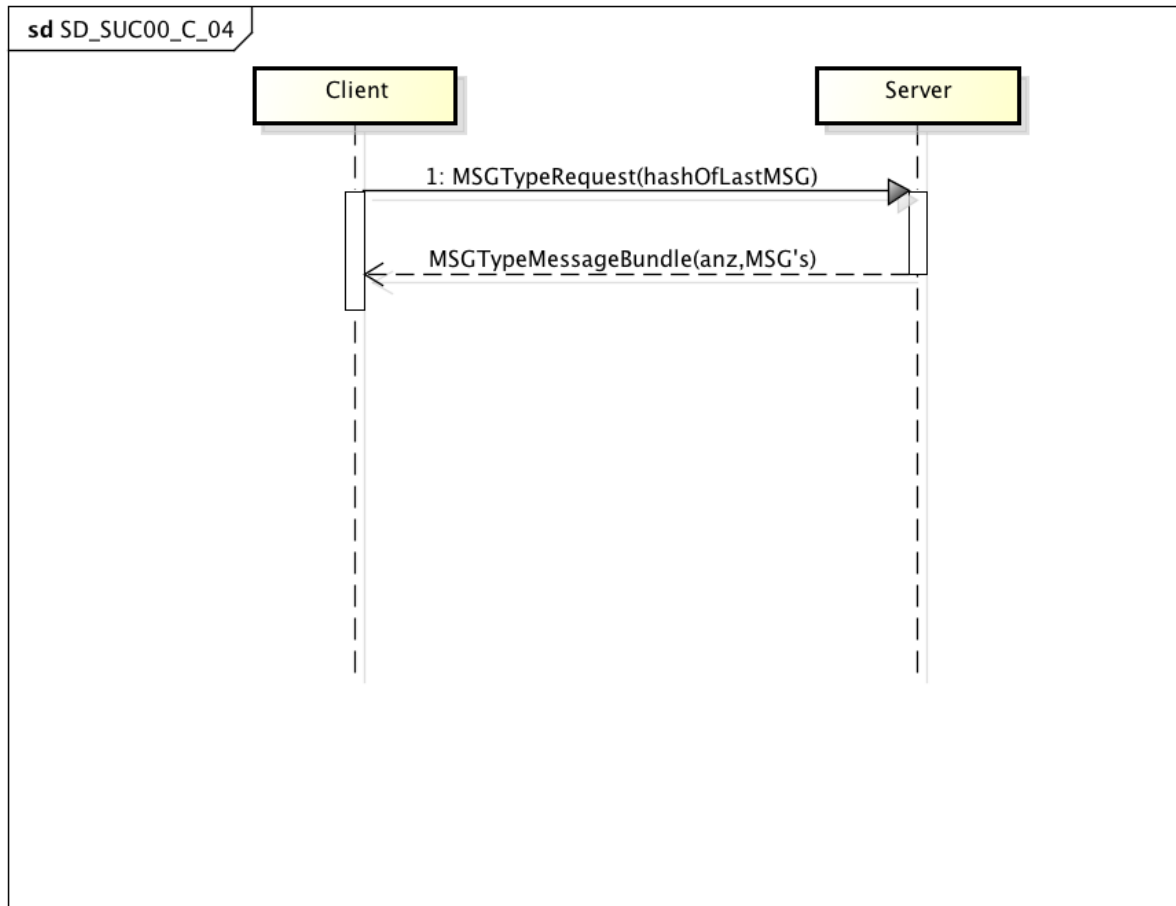
4.2.1.5 SUC00_C_04: Nachrichten vom Server abrufen

Title [Nachrichten vom Server abrufen]

Primary Actor: [Client / Server]

Scope: Client / Server

(Story): C sendet eine MessageTypeRequestMessage welche den Hash der letzten empfangenen MSG an S.



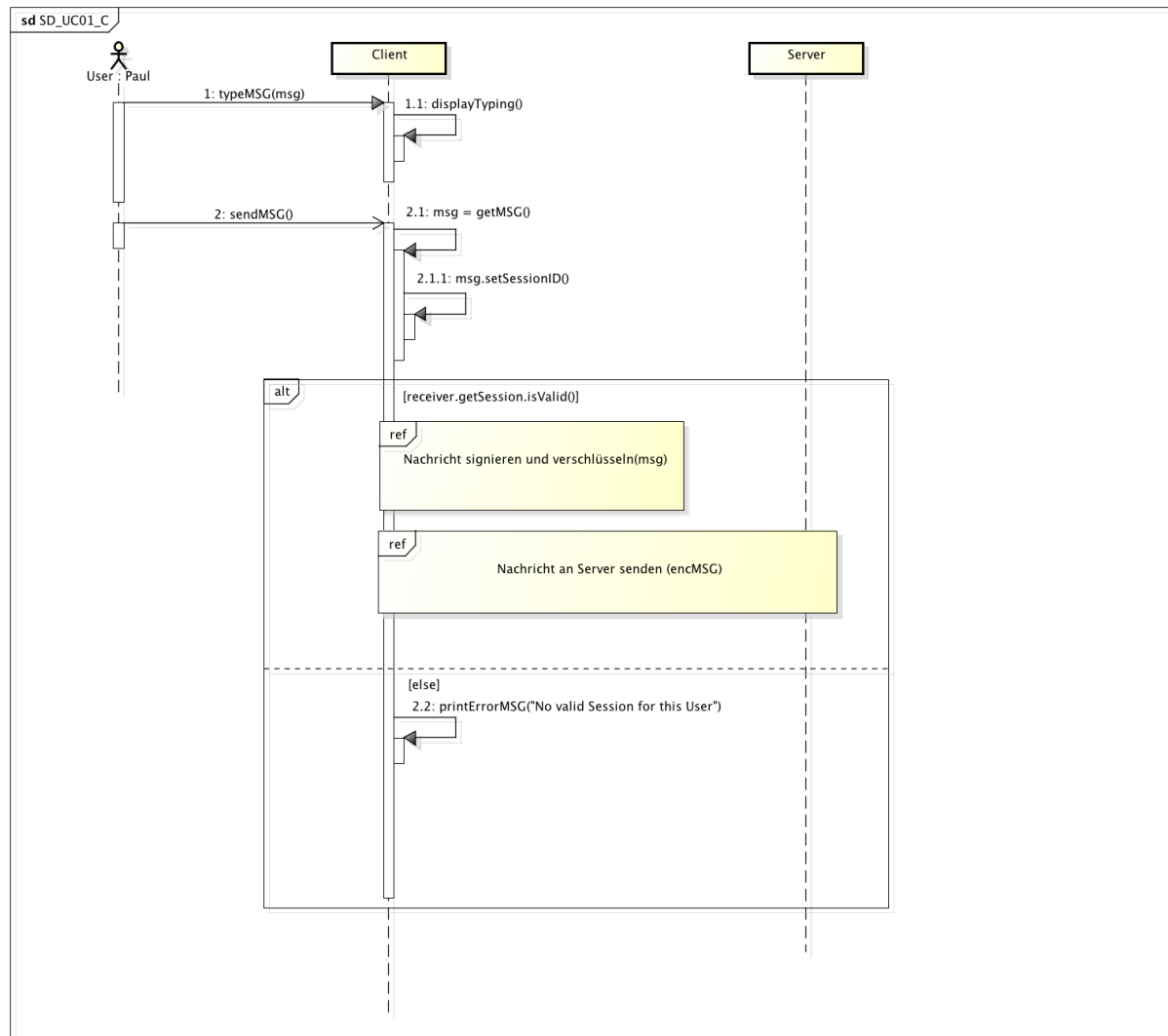
4.2.2 UC01_C: Nachricht an XY senden

Title [Nachricht an XY senden]

Primary Actor: [Chatteilnehmer]

Scope: Client

(Story): N gibt die MSG ein, sie wird im angezeigt und er kann sie mit "senden" versenden, darauf wird SUC01_C_00 aufgerufen.



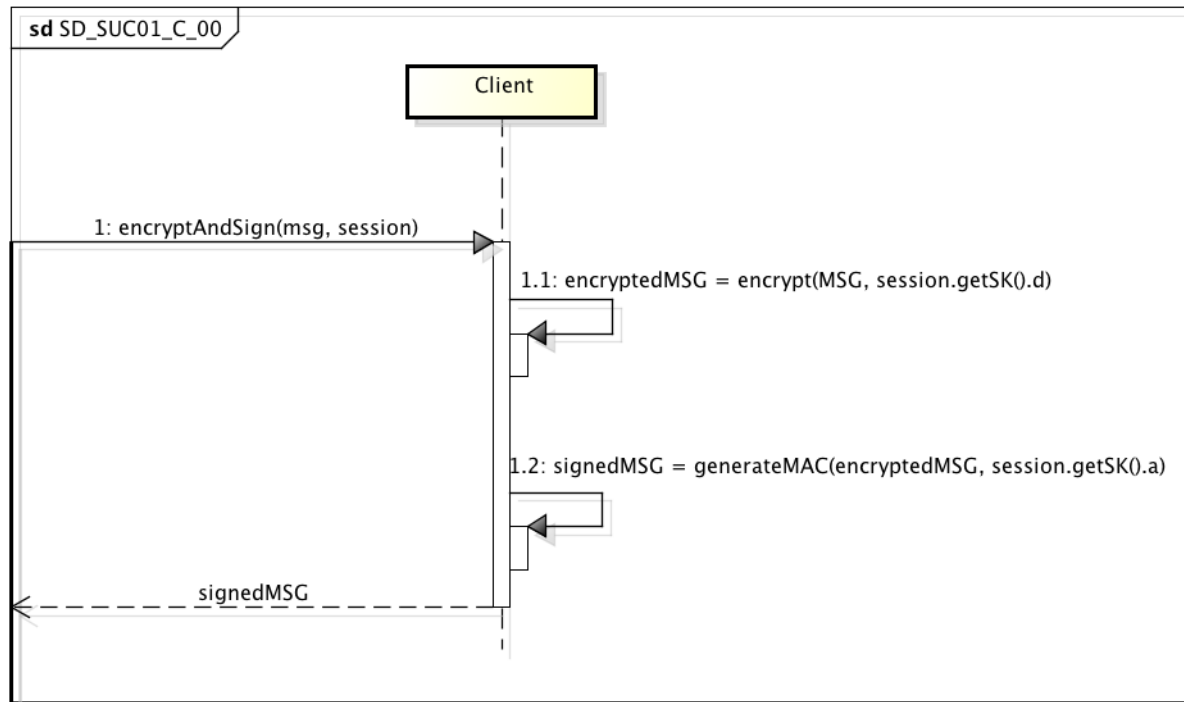
4.2.2.1 SUC01_C_00: Nachricht verschlüsseln und MAC generieren

Title [Nachricht verschlüsseln und MAC generieren]

Primary Actor: [Client]

Scope: Client

(Story): C verschlüsselt die MSG und generiert anschliessend den MAC.



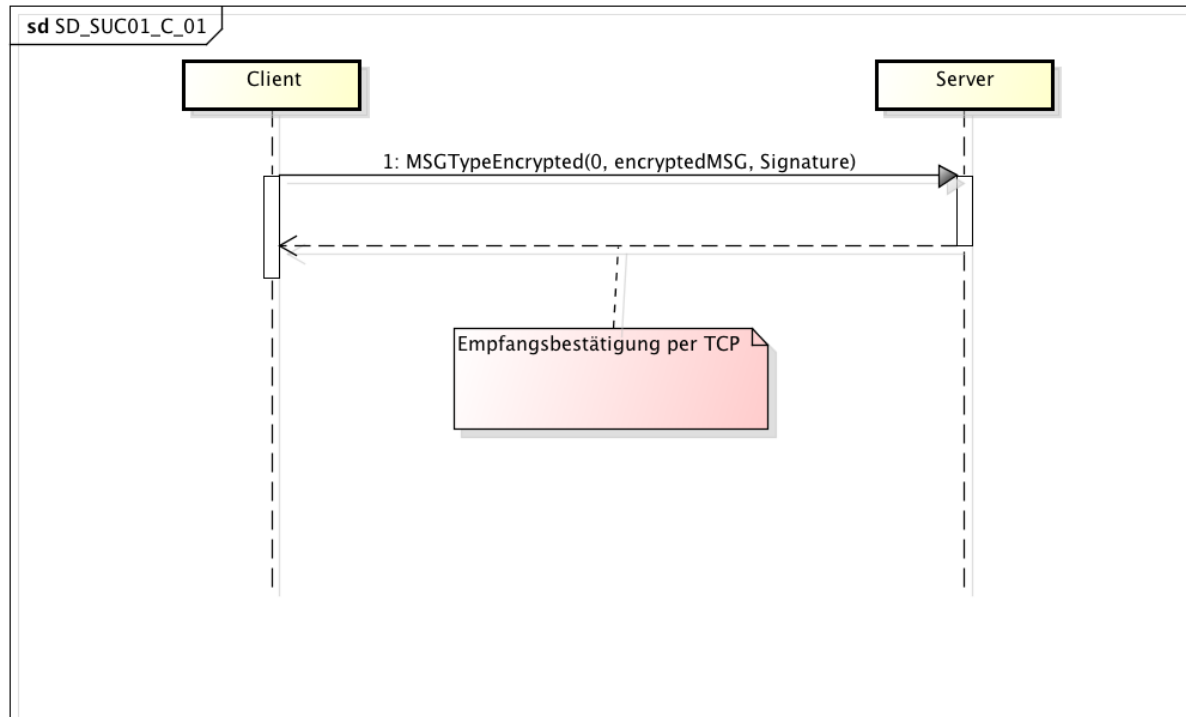
4.2.2.2 SUC01_C_01: Nachricht an Server senden

Title [Nachricht an Server senden]

Primary Actor: [Client]

Scope: Client / Server

(Story): C sendet die verschlüsselte MSG an den Server.



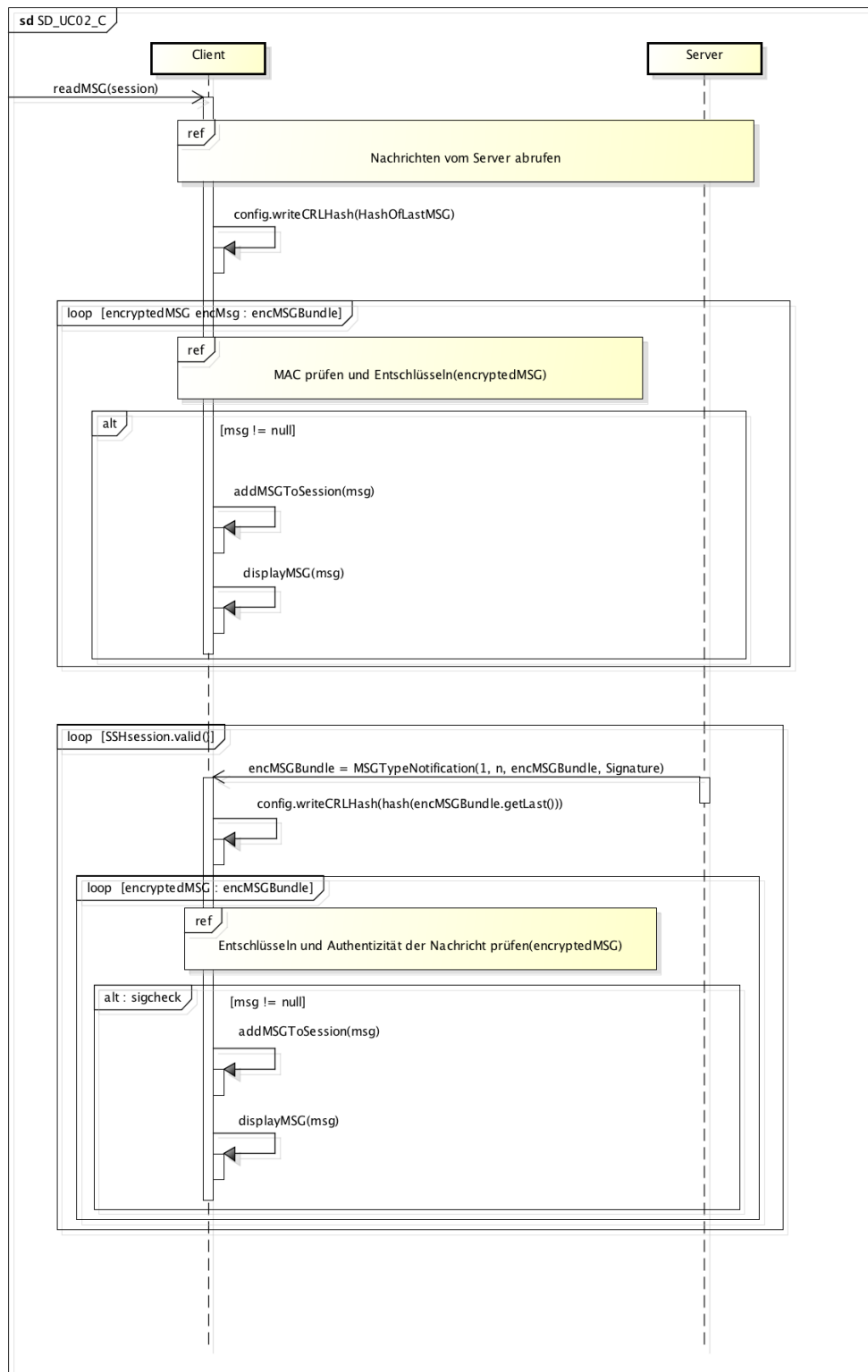
4.2.2.3 UC02_C: Nachrichten lesen

Title [Nachrichten lesen]

Primary Actor: [Chatteilnehmer]

Scope: Client

(Story): C wartet auf MSG's von S. Wenn er welche empfängt speichert er den Hash der letzten empfangenen MSG, dann wird SUC04_C_00 ausgeführt. Die verbleibenden MSG's werden dem N angezeigt.



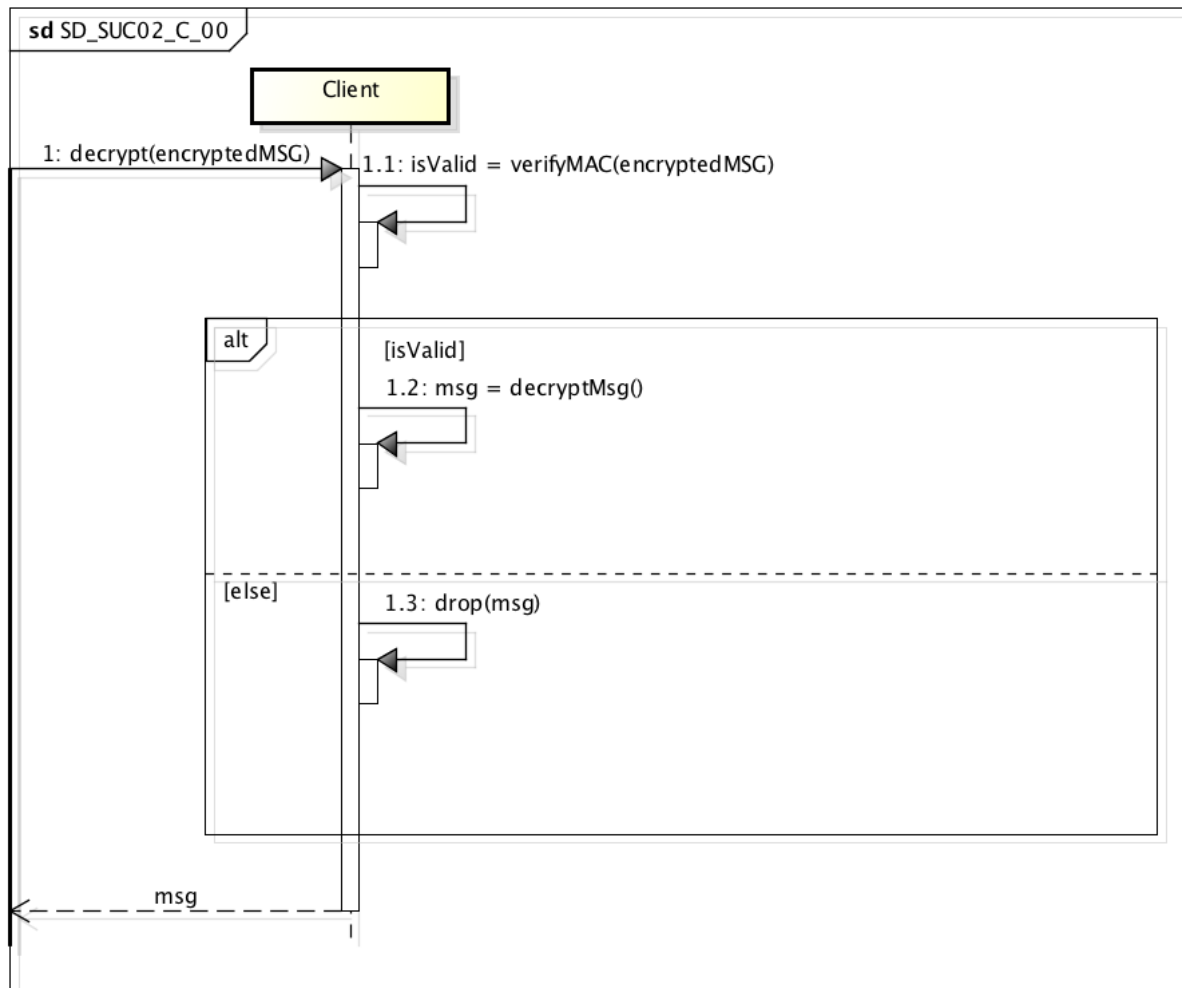
4.2.2.4 SUC02_C_00: MAC prüfen und entschlüsseln

Title [MAC prüfen und entschlüsseln]

Primary Actor: [Client]

Scope: Client

(Story): Die MSG's, welche C empfangen und erfolgreich deren MAC verifiziert hat, sind mit extrem hoher Wahrscheinlichkeit für N bestimmt. Durch die Verifikation des MACs ist bekannt zu welcher Session der Schlüssel gehört, und die Nachricht wird entschlüsselt. Kann der MAC nicht verifiziert werden, dann wird die Nachricht verworfen.



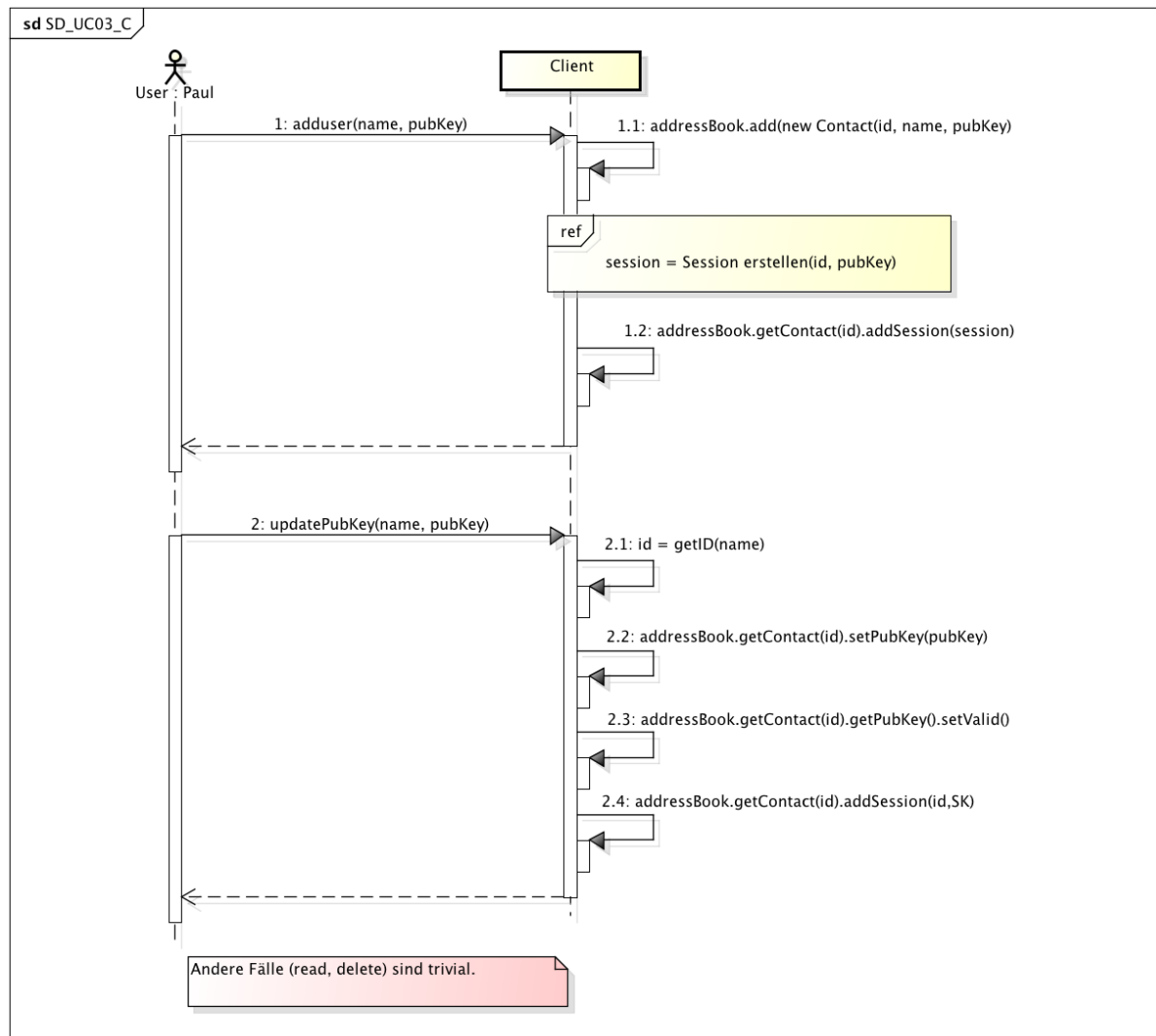
4.2.3 UC03_C: Kontakt verwalten

Title [<<CRUD>> Kontakt verwalten]

Primary Actor: [Chatteilnehmer]

Scope: Client

(Story): N kann E's seinem Adressbuch hinzufügen, dazu benötigt er nur den Namen und den Pubkey von E, dadurch wird SUC03_C_00 ausgeführt. N kann Einträge im Adressbuch bearbeiten, z.B. einen Schlüssel ersetzen. N kann Einträge löschen.



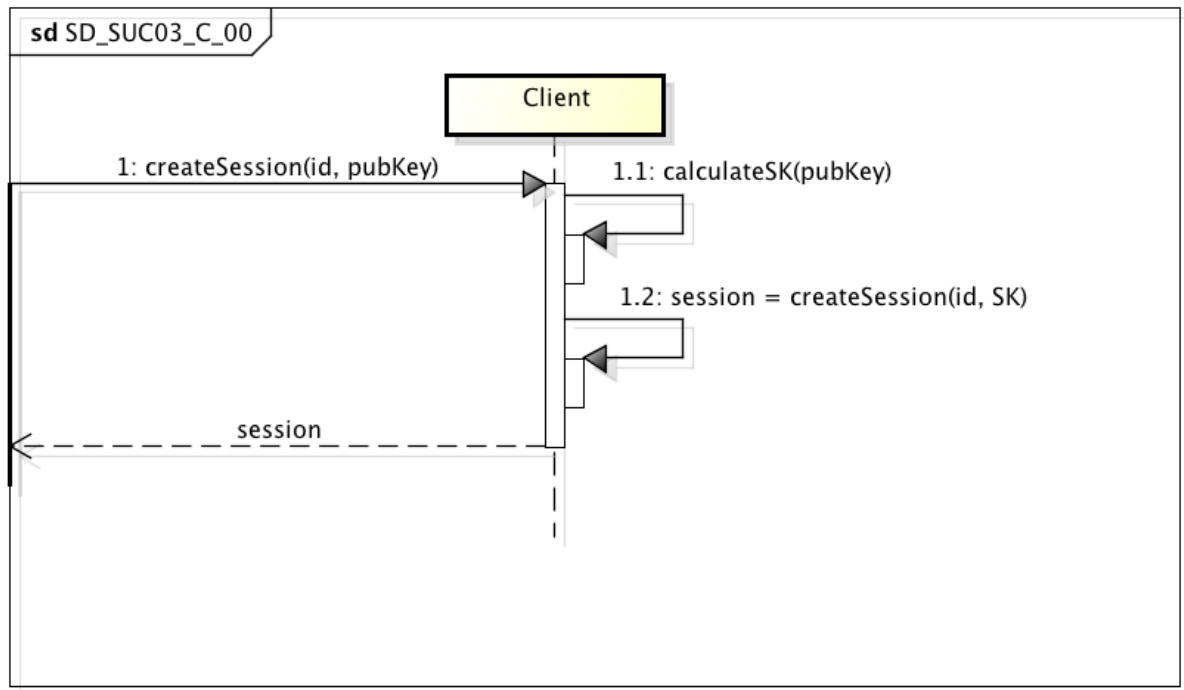
4.2.3.1 SUC03_C_00: Session erstellen

Title [Session erstellen]

Primary Actor: [Client]

Scope: Client

(Story): Nach dem setzen des PubKeys von E wird der gemeinsame SK ausgerechnet und der Session, die für E erstellt wurde, zugewiesen.



4.2.4 UC04_C: Gruppen verwalten

Title [<<CRUD>> Gruppen verwalten]

Primary Actor: [Chatteilnehmer]

Scope: Client

(Story): [Im Rahmen des SE02 Projekts ist die Umsetzung des Gruppen-Features nicht vorgesehen]

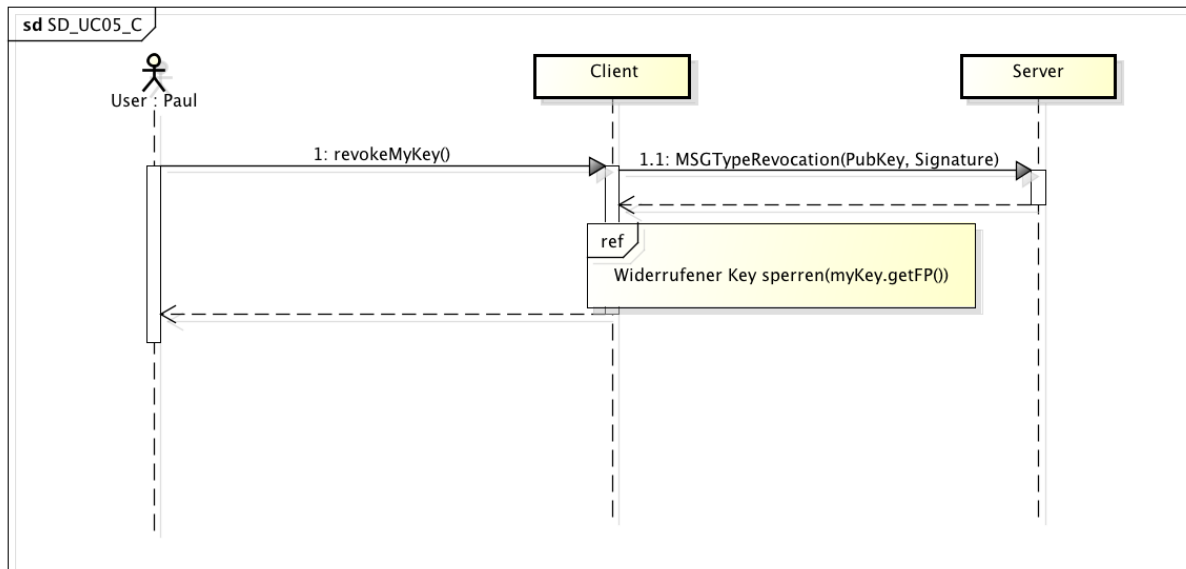
4.2.5 UC05_C: Sperren von Publickeys

Title [Sperren von Publickeys]

Primary Actor: [Chatteilnehmer]

Scope: Client / Server

(Story): Der Nutzer merkt, dass sein PrivateKey kompromittiert wurde und er ihn widerrufen muss. Also sendet er eine Revozierungsnachricht an S. Sobald er die Empfangsbestätigung vom Server hat, werden die eigenen Sessions als ungültig markiert, bis der Schlüssel ersetzt wurde.



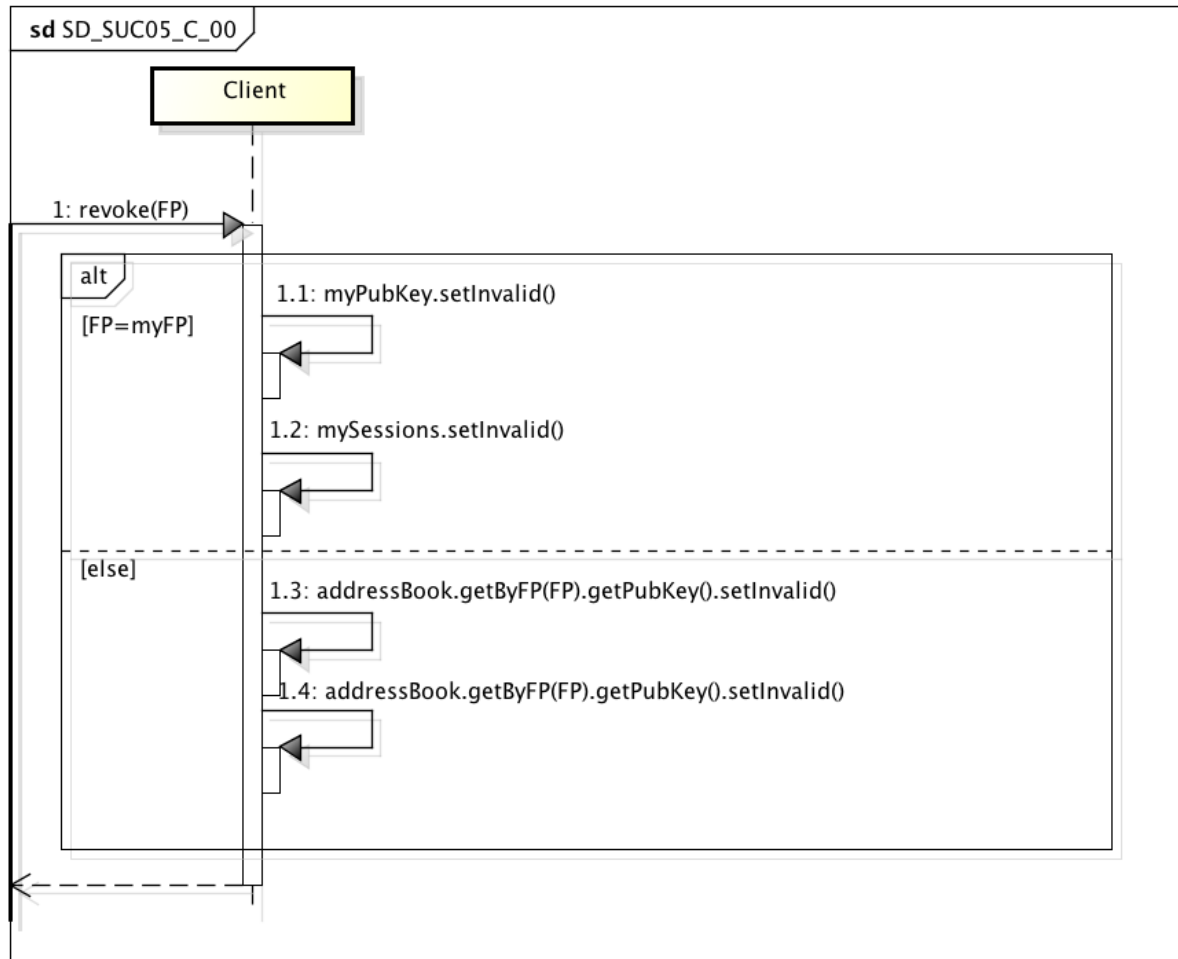
4.2.5.1 SUC05_C_00: Widerrufenes Zertifikat sperren

Title [Widerrufenes Zertifikat sperren]

Primary Actor: [Client]

Scope: Client

(Story): Wenn eine Revozierungsnachricht empfangen wird, dann muss sowohl der Publickey wie auch der gemeinsame Sessionkey mit diesem Partner ungültig werden.



4.3 User/User Usecases

4.3.1 UC00_N: Konfiguration editieren

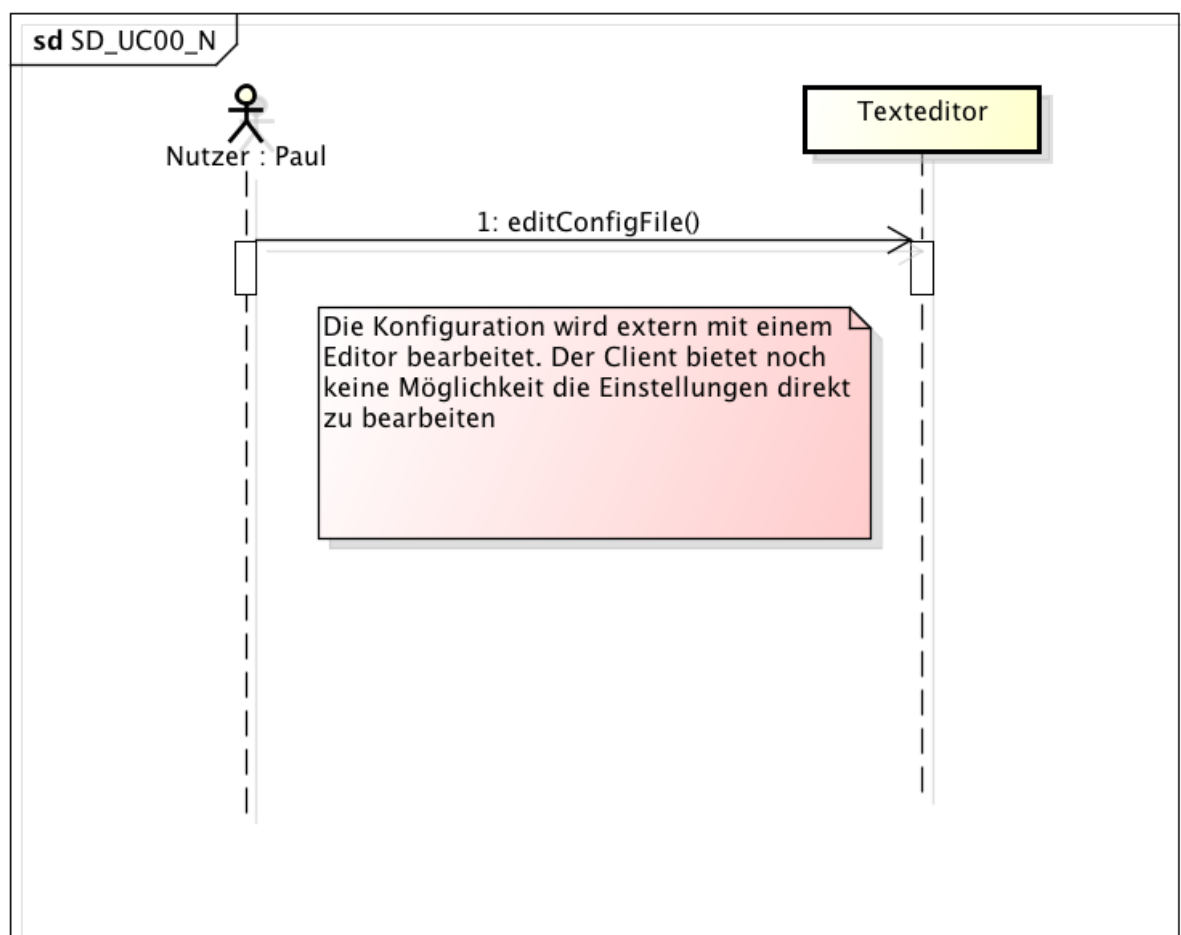
Title [Konfiguration editieren]

Primary Actor: [Chatteilnehmer]

Scope: Nutzer

(Story): N kann:

- Verbindungsdaten [IP, Pubkey] lesen und schreiben.
- Hash der Blacklist lesen und schreiben.
- Hash der letzten MSG lesen und schreiben.
- Einen Pfad zu einem PublicKey / PrivateKey angeben, welche im C geladen werden sollen.
- Andere Configdaten lesen und schreiben.



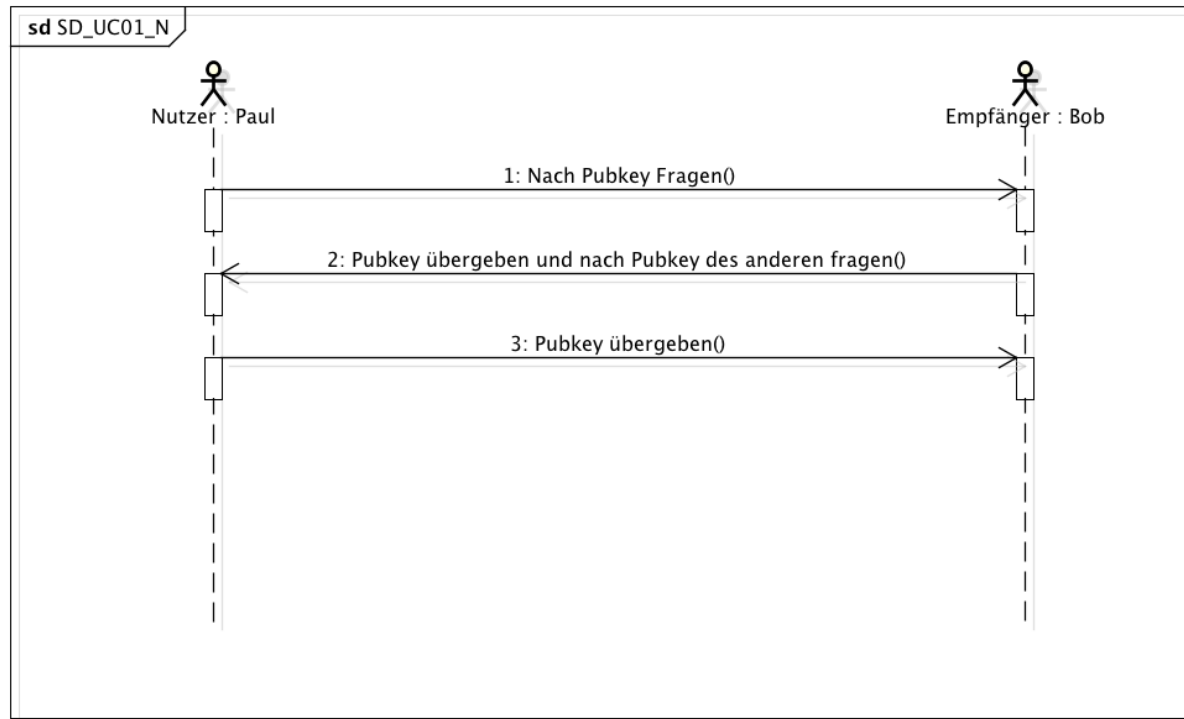
4.3.2 UC01_N: Publickeys austauschen

Title [Publickeys austauschen]

Primary Actor: [Chatteilnehmer / Chatteilnehmer]

Scope: Nutzer & Nutzer

(Story): Nutzer müssen die Schlüssel manuell austauschen. [Im Rahmen des SE02 Projekts ist keine Funktionalität vorgesehen, welche es ermöglicht, über den Server die Schlüssel auszutauschen.]



4.4 Admin/Server und Client/Server Usecases

4.4.1 UC00_S: Server starten

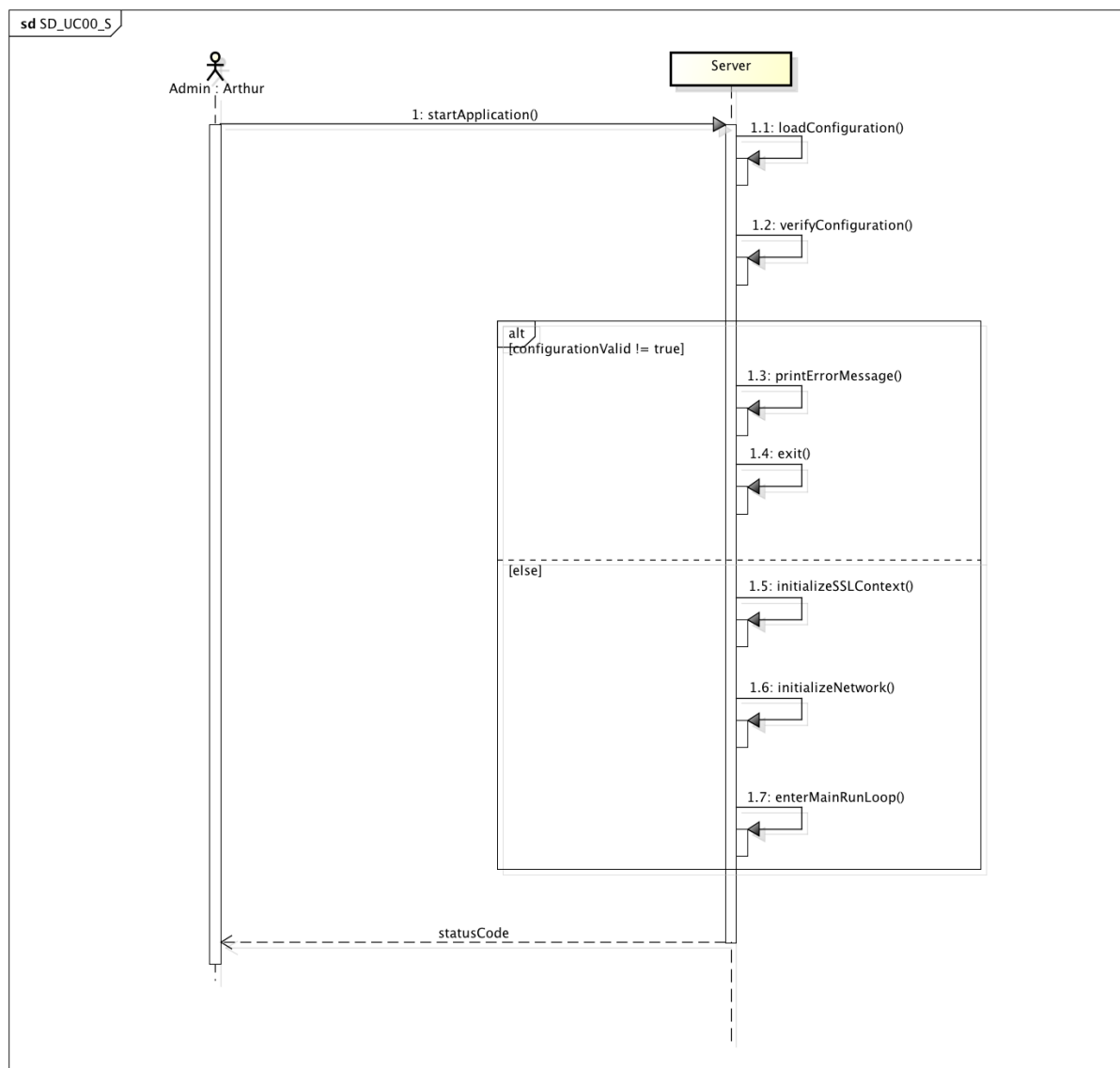
Title [Server starten]

Primary Actor: [Server Admin]

Scope: Server

(Story):

- Der Server Administrator verwendet das mitgelieferte Startskript um den Server zu starten.
- Der Server verifiziert die syntaktische Korrektheit der Konfiguration
- Der Server lädt die Konfiguration
- Der Server lädt sein TLS Zertifikat
- Der Server hört auf den konfigurierten Adressen und Ports
- Der Server betritt den Main-Run-Loop



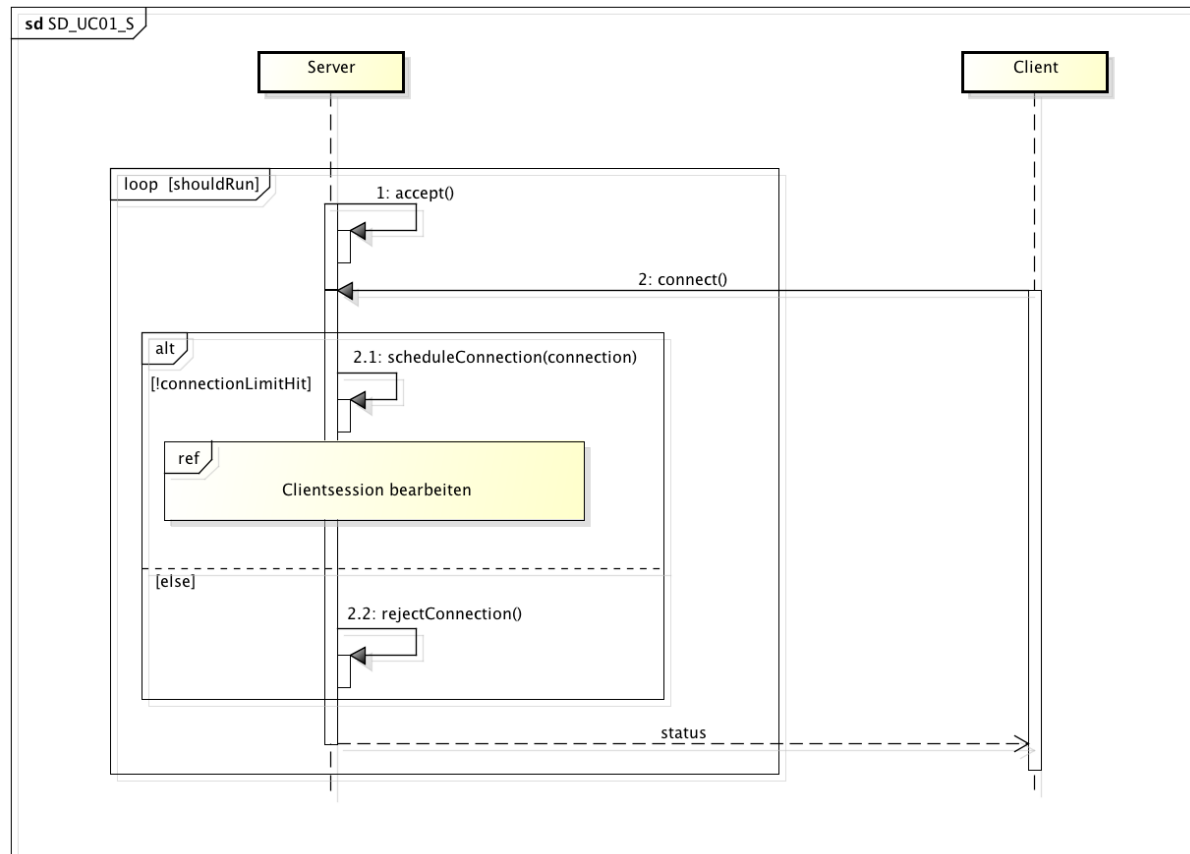
4.4.2 UC01_S: Verbindung annehmen

Title [Verbindung annehmen]

Primary Actor: [Server]

Scope: Server

(Story): Der Server befindet sich nach dem Start im Main-Run-Loop. Er wartet auf eingehende Verbindungen von Clients. Sobald eine Verbindung eingeht wird diese akzeptiert und im Connection-Handling-Runloop scheduled. [In einer weiteren Ausbaustufe wird geprüft ob die Verbindung legitim ist].



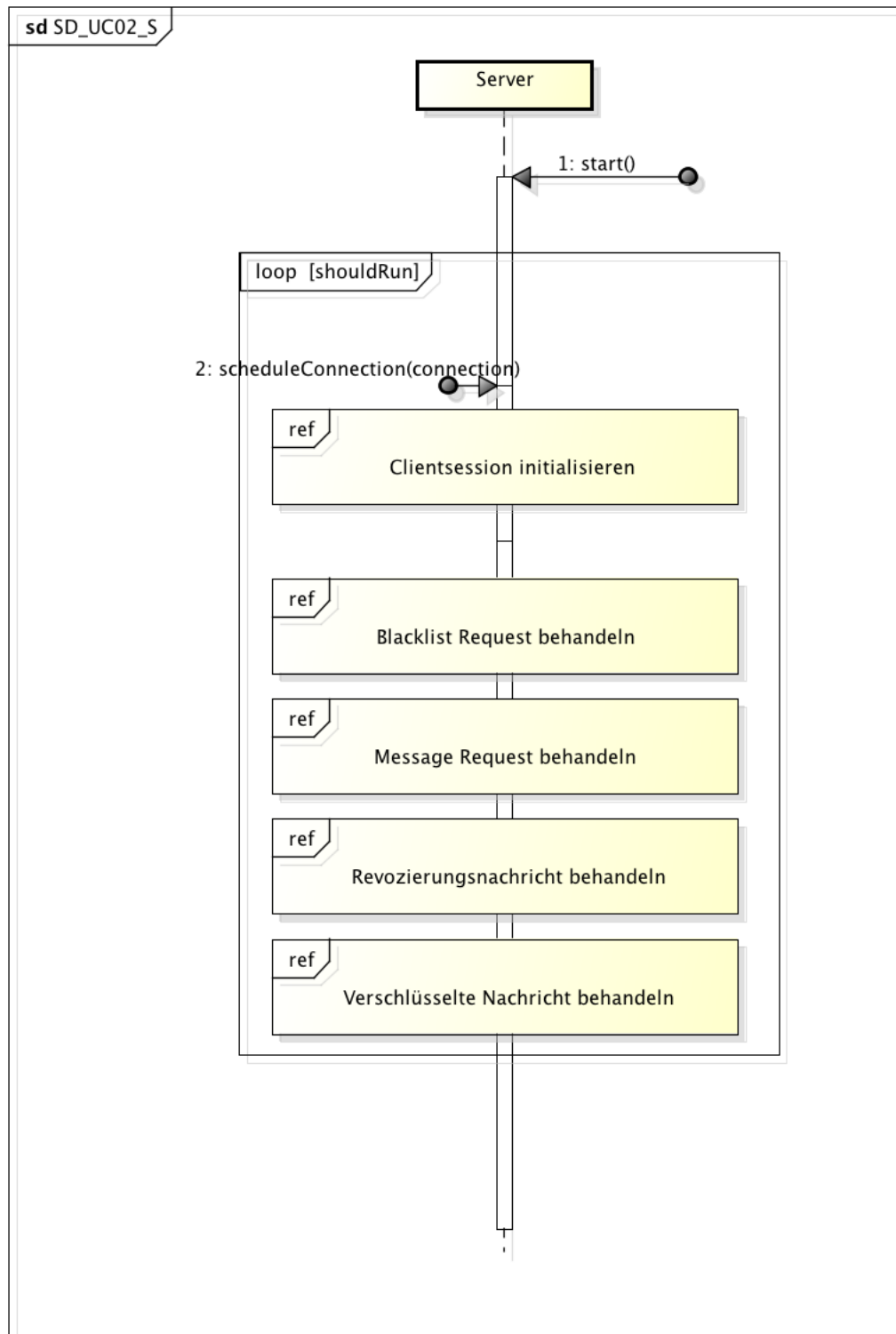
4.4.3 UC02_S: Clientsession bearbeiten

Title [Clientsession bearbeiten]

Primary Actor: [Server Admin]

Scope: Server

(Story): Im Connection-Handling-Runloop werden die Sub-Usecases behandelt solange eine Session mit einem Client besteht.



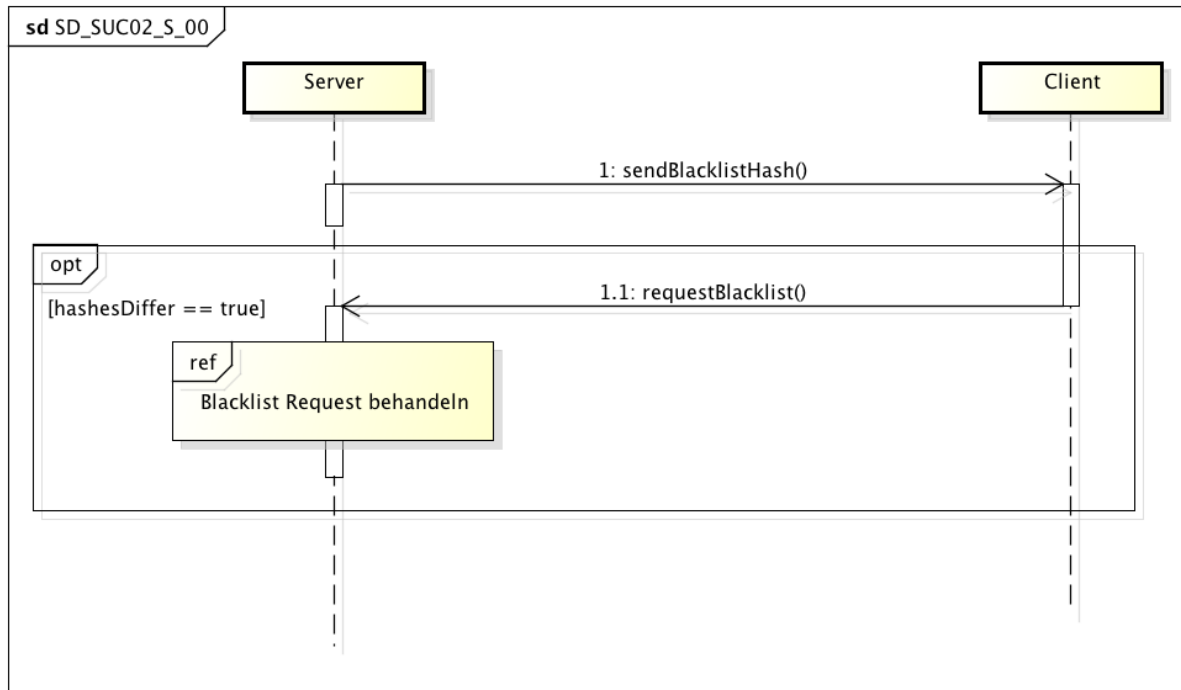
4.4.3.1 SUC02_S_00: Clientsession initialisieren

Title [Clientsession initialisieren]

Primary Actor: [Server, Client]

Scope: Server

(Story): Der Server sendet dem Client seinen aktuellen Blacklist Hash. Möglicherweise sendet der Client daraufhin einen Blacklist-Request, den der Server dann mit SUC02_S_01 bearbeitet.



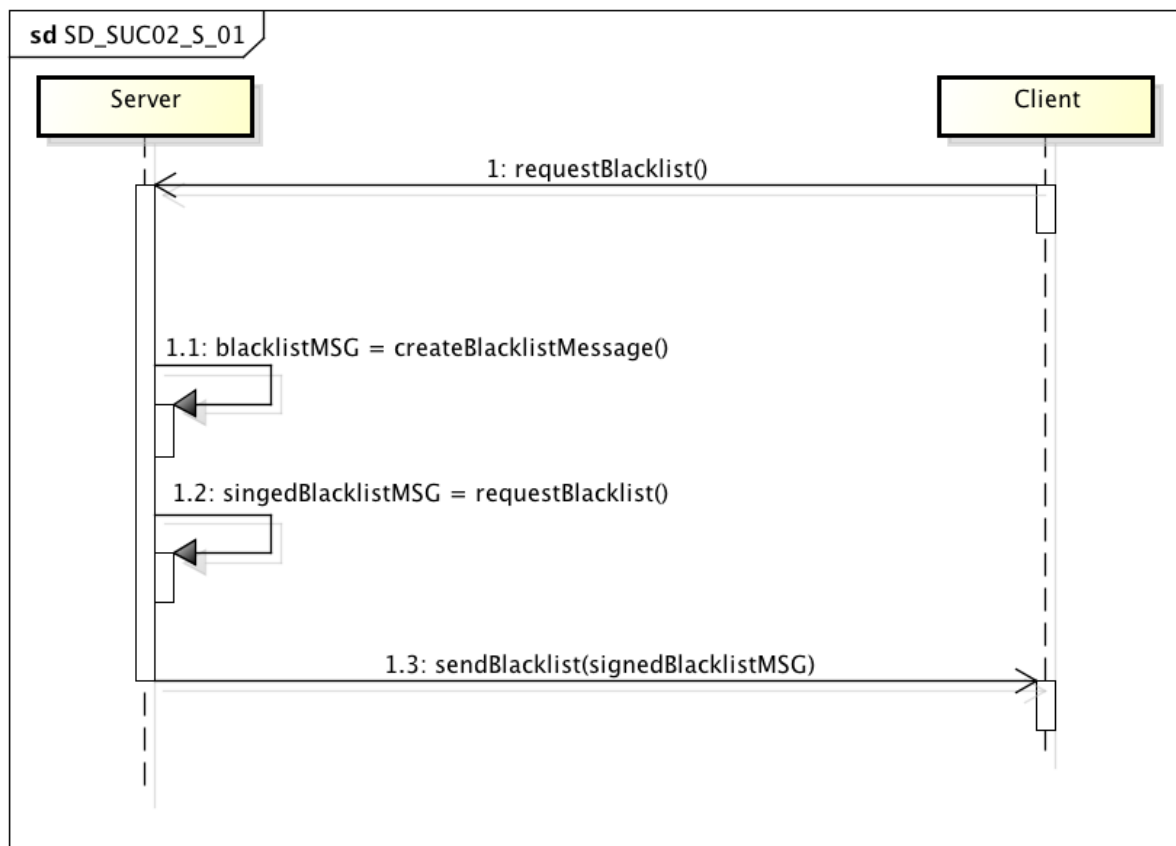
4.4.3.2 SUC02_S_01: Blacklist Request behandeln

Title [Blacklist Request behandeln]

Primary Actor: [Server, Client]

Scope: Server

(Story): Der Server empfängt vom Client eine Blacklist-Request-Message. Der Server konstruiert jetzt eine Blacklist-Response-Message mit der signierten Blacklist als Inhalt und sendet diese an den Client.



4.4.3.3 SUC02_S_02: Message Request behandeln

Title [Message Request behandeln]

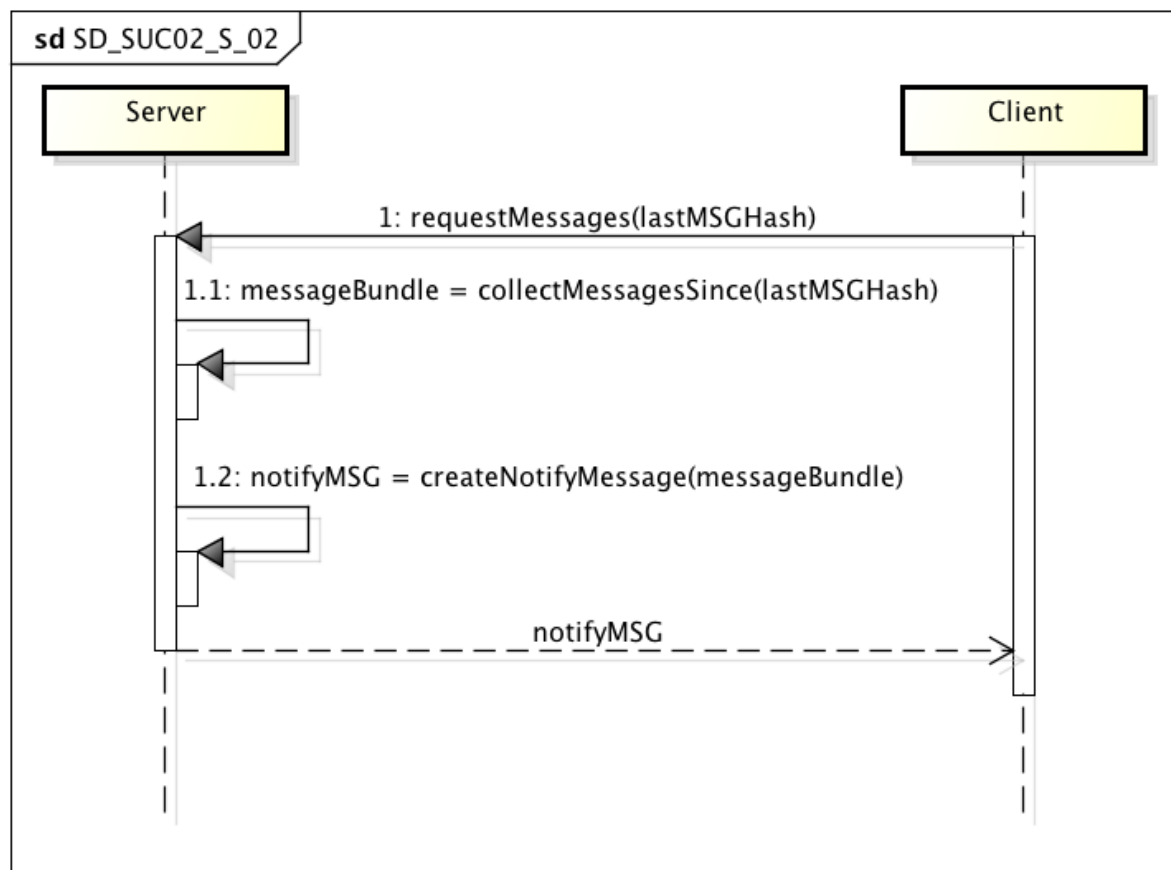
Primary Actor: [Server, Client]

Scope: Server

(Story): Der Server empfängt von einem verbundenen Client eine Message-Request-Message. Er liest den enthaltenen Hash und sucht nach der Nachricht mit dem selben Hash.

Main Success Scenario: Der Server findet den Hash der Nachricht in seinem Index. Er konstruiert ein (möglicherweise leeres) Bundle der Nachrichten, welche nach der Nachricht eingegangen sind von welcher er den Hash vom Client geschickt bekommen hat. Er sendet dieses Bundle mittels einer Notify-Message an den Client.

Extension Scenario: Der Server findet den Hash der Nachricht nicht in seinem Index. Er konstruiert ein Bundle aller Nachrichten und sendet dieses, mittels einer Notify-Nachricht, an den Client.



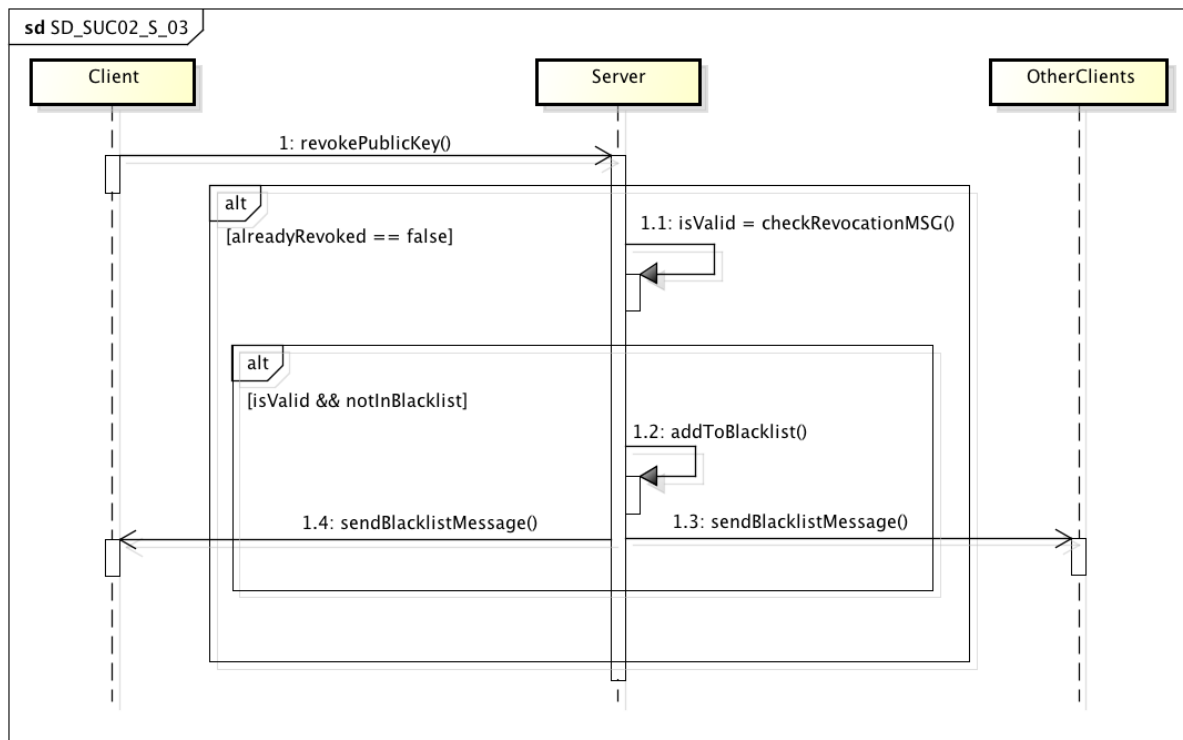
4.4.3.4 SUC02_S_03: Revozierungsnachricht behandeln

Title [Revozierungsnachricht behandeln]

Primary Actor: [Server, Client]

Scope: Server

(Story): Der Server empfängt eine KeyRevocationMessage. Er überprüft die Unterschrift über den revozierten Public-Key. Ist die Überprüfung erfolgreich und der Key noch nicht in der Blacklist eingetragen, trägt er ihn in seine Blacklist ein, andernfalls verwirft er die Nachricht. Nach einer erfolgten Eintragung, sendet er an alle verbundenen Client eine Blacklist-Message mit der neuen Blacklist.



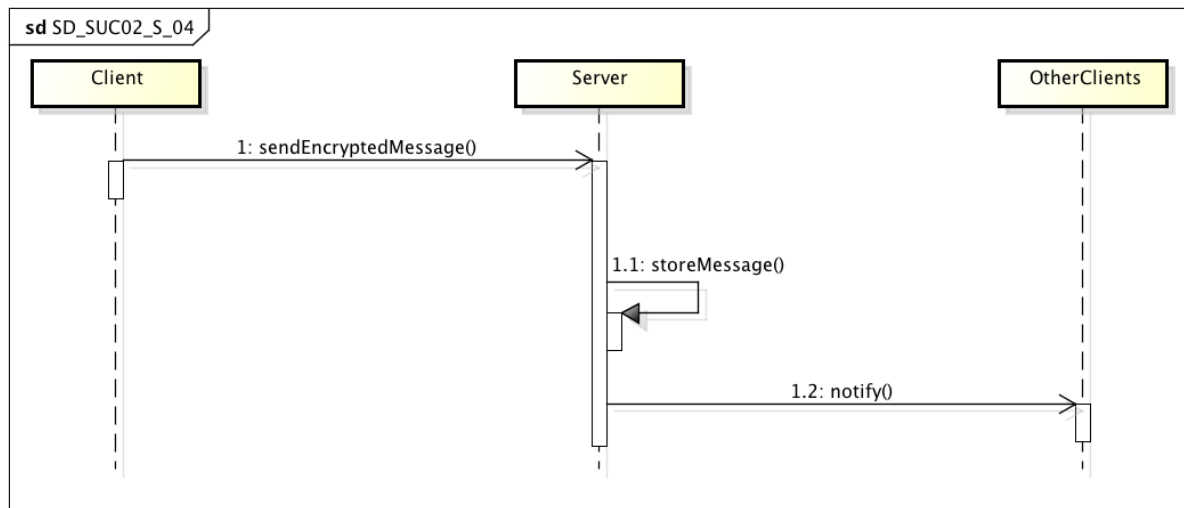
4.4.3.5 SUC02_S_04: Verschlüsselte Nachricht behandeln

Title [Verschlüsselte Nachricht behandeln]

Primary Actor: [Server, Client]

Scope: Server

(Story): Der Server empfängt eine verschlüsselte Nachricht vom Client. Er sendet die empfangene Nachricht an alle verbundenen Clients, ausser dem Client von dem die Nachricht kam, mittels einer Notify-Message.



4.5 Abusecases

4.5.1 AC00: DDOS-Attack on Server

Title [DDOS-Attack on Server]

Primary Actor: [Hostile Entity]

Scope: Server

(Story): Der Server soll nur authentisierten Teilnehmern Zugang gewähren

Solution: [Wird im Rahmen des SE02 Projekts nicht abgedeckt.]

4.5.2 AC01: Senden von Nachrichten mit defektem MAC

Title [Senden von Nachrichten mit defektem MAC]

Primary Actor: [Chatteilnehmer / Angreifer]

Scope: Server und Client

(Story): Wenn ein Angreifer oder ein Nutzer eine Nachricht mit defektem oder ungültigem MAC sendet, kann dieser nicht vertraut werden.

Solution: [Nachrichten mit invalidem MAC werden verworfen]

4.5.3 AC02: Manipulieren von Blacklist

Title [Manipulieren von Blacklist]

Primary Actor: [Server]

Scope: Server und Client

(Story): Hat ein Angreifer den Server unter Kontrolle, so kann er problemlos die Blacklist manipulieren. Er kann einen gemeldeten Blacklisteintrag ignorieren oder sogar im Nachhinein einzelne Einträge aus der Blacklist entfernen.

Solution1: Das Ignorieren eines Blacklisteintrags lässt sich nur beheben, wenn ein verteiltes System eingesetzt wird, womit sich dann mittels Mehrheitsentscheid eine Manipulation feststellen lässt.

Solution2: Ein Nachträgliches ändern der Blacklist liesse sich verhindern, indem die Blacklist jeweils einen neuen Hash bekommt und der alte Hash in die Berechnung einfließt. Der neueste Eintrag liesse sich jedoch dennoch fälschen.

5. Systemoperationen

5.1 Contract C01: Schlüsselaustausch

Operation: Publickeys austauschen

Querverweise: UseCase UC01_N

Vorbedingungen:

- Beide Gesprächspartner(GP) verfügen über einen Public-Key.

Nachbedingungen:

- Gesprächspartner ist verifiziert.
- Der Public-Key des GP (als Grundlage für die Berechnung des SessionKeys) ist vorhanden.

5.2 Contract C02: Kontakt hinzufügen

Operation: <<CRUD>> Kontakte

Querverweise: UseCase UC03_C

Vorbedingungen:

- Public-Key des GP ist vorhanden.

Nachbedingungen:

- SessionKey mit Gesprächspartner wurde berechnet.
- Session mit Gesprächspartner ist vorbereitet.

5.3 Contract C03: SendMSG

Operation: SendMSG()

Querverweise: UseCase UC01

Vorbedingungen:

- Session ist vorhanden.

Nachbedingungen:

- Nachricht ist auf Server für Empfänger bereit.

5.4 Contract C04: ReadMSG

Operation: ReadMSG()

Querverweise: UseCase UC02

Vorbedingungen:

- Session mit Gesprächspartner existiert.
- Nachricht liegt auf Server bereit.

Nachbedingungen:

- Nachricht ist verschlüsselt lokal gespeichert.
- Nachricht kann entschlüsselt betrachtet werden.

6. Requirements

6.1 Funktionale Anforderungen

6.1.1 Netzwerkbasiert

Das System unterstützt das senden und empfangen von Text-Nachrichten über eine TCP/IP basierte TLS-Verbindung.

6.1.2 Asynchrone Kommunikation

Das System muss den asynchronen Austausch von Nachrichten beherrschen. Empfänger und Absender einer Nachricht müssen nicht zur selben Zeit Online sein.

6.1.3 Symmetrische Verschlüsselung

Die Text-Nachrichten welche zwischen den Clients ausgetauscht werden, müssen mit einem symmetrischen Verschlüsselungsverfahren verschlüsselt werden. Um den symmetrischen Schlüssel zu erzeugen, muss jeder Client im Besitz seines eigenen Private-Keys und des Public-Keys der Person sein, mit welcher er kommunizieren will. Das verwendete Verfahren muss mindestens eine Sicherheit von 128-Bit gewähren.

6.2 Nichtfunktionale Anforderungen

6.2.1 Performance

Client: Der Client muss auf Mobil-Computern welche mit Ubuntu 14.04 betrieben werden lauffähig sein. Das Verschlüsseln und Signieren sowie das Entschlüsseln und Prüfen der Signatur einer Nachricht darf auf einem Gerät mit 2-CPU-Kernen und 1GB RAM nicht länger als 2 Sekunden dauern.

Server: Der Server muss auf einem Embedded System (analog Raspberry Pi) 50 Nachrichten pro Sekunde empfangen und ablegen können. Ein Rate-Limiting ist in Rahmen des SE2Projekts nicht vorgesehen und sollte vorerst mittels einer Firewall (z.B. iptables) durchgeführt werden.

6.2.2 Geheimhaltung

Client: Der Client darf die empfangenen Nachrichten nur in verschlüsselter Form im persistenten Speicher des Systems halten. Des Weiteren muss der Client mindestens beim Start eine Identifikation des Benutzers verlangen.

Server: Es darf dem Server nicht möglich sein, eine beliebige Daten-Nachricht effizient zu entschlüsseln. Der Nachrichtenspeicher des Servers sollte in einem verschlüsselten und flüchtigen Dateisystem (z.B. encfs in RAM-Disk) liegen.

6.2.3 Wahrung der Privatsphäre

Client: Der Client darf die Identität seiner Kommunikationspartner nicht in der Kommunikation mit dem Server preisgeben.

Server: Der Server darf nicht in der Lage sein, ausser den IP-Adressen, Daten über Identität (Name, Betreff, Kommunikationspartner) seiner Kommunikationspartner zu erlangen. Eine Ausnahme stellen Revozierungs-Nachrichten dar, in welchen ein Client seinen Public-Key-Fingerprint dem Server gegenüber preisgeben muss.

Gesamtsystem: Einem Man-in-the-Middle darf es nicht möglich sein, die Identität der Kommunizierenden Parteien fest zu stellen. Einem Angreifer welcher Root-Rechte auf dem Server erlangt, darf es nicht möglich sein die Identität der Kommunikationspartner festzustellen. Eine Ausnahme bildet das lesen von Black-List-Control-Messages (siehe oben).

6.2.4 Kryptografische Stärke

Die kryptografische Stärke im Gesamtsystem das 128-Bit nicht unterschreiten. Zu diesem Zwecke müssen die von den Clients verwendeten ECDH-Schlüssel-Paare mindestens eine Länge von 256-Bit aufweisen. Zur Verschlüsselung der Nachrichten durch den Client muss mindestens ein 128-Bit AES Schlüssel ausgehandelt werden. Als Message-Authentication-Code muss ein Verfahren gewählt werden, welches mindestens 128-Bit Sicherheit gewährt.

6.2.5 Qualität

6.2.5.1 Offenheit

Der gesamte Quellcode für das System wird veröffentlicht um so dem Benutzer zu erlauben die Implementation auf Schwachstellen zu untersuchen. Dem Benutzer ist es damit auch möglich, Anpassungen für eigene Anforderungen vor zu nehmen.

6.2.5.2 Benutzbarkeit

Um die Initiale Vertrauensstellung zu etablieren, müssen sich die Benutzer des Systems entweder persönlich treffen um ihre Schlüssel zu verifizieren oder andere Out-Of-Band-Verfahren zum Austausch der Schlüssel verwenden. Zu einem späteren Zeitpunkt soll ein Schlüsselwechsel über den Server möglich sein, dies wird jedoch im Rahmen des SE2-Projekts nicht umgesetzt.

6.2.5.3 Zuverlässigkeit

Client: Der Client darf aufgrund ungültiger Nutzereingaben nicht zusammenbrechen. Auch beim Empfang defekter Nachrichten, sei es infolge von Netzwerkfehlern oder eines Angriffs (Malicious-Server), darf der Client nicht zusammenbrechen.

Server: Der Server muss ungültige Nachrichten verwerfen und darf nicht auf Grund solcher zusammenbrechen. Des Weiteren darf der Server nicht zusammenbrechen wenn der Nachrichten-Puffer voll ist. Der Server soll keine Speicher-Leaks aufweisen.

6.2.5.4 Portabilität

Client: Der Client ist in Python 3 geschrieben und soll auf allen von Python 3 unterstützten Plattformen lauffähig sein.

Server: Der Server ist in C++ geschrieben und soll auf POSIX (teil-)konformen Betriebssystemen lauffähig sein.

Gesamtsystem: Im Rahmen des SE2-Projekts wird die Lauffähigkeit auf dem HSR-Buildserver mit Ubuntu 14.04 garantiert.

6.3 Constraints

- Im Rahmen des SE2-Projekts wird auf die Gruppenchat-Funktionalität verzichtet.
- Mobile-Clients für iOS und Android werden als spätere Ausbaustufe erachtet.
- In der Version welche im SE2-Projekt erarbeitet wird, werden nur Textnachrichten mit einer Länge von bis zu 512 Byte unterstützt, damit es nicht zu IP-Fragmentierung kommt.

7. Protokollspezifikation

7.1 Äusseres Nachrichtenformat

Alle Nachrichten verfügen über die gleiche Struktur:

Length	Type	Res
MSGData		

7.1.1 Beschreibung der Felder

Feld	Typ	Länge in Byte	Bedeutung
Length	Integer	2	Länge der Nachricht
Type	Integer	1	Typ der Nachricht
Res	-	1	Reserviert
Data	Protobuf	Length - 4	Protobuf Nachrichtendaten

Das Feld **Res** ist für spätere Erweiterungen reserviert und momentan immer auf **0xFF** gesetzt. Im **Data** Feld werden die spezifischen Protobuf Daten der jeweiligen Nachricht transportiert. Im Falle der Encrypted-Message besteht das **Data** Feld aus den verschlüsselten Protobuf Daten gefolgt von einem MAC.

7.2 Nachrichten-Typen

7.2.1 MSGTypeEncrypted - Verschlüsselte Nachricht.

Feld	Typ	Länge in Byte	Bedeutung
SubType	Byte[]	1	Subtyp der Nachricht
Session	Byte[]	16	Session ID
MSGData	Byte[]	512	Nachrichtendaten

Das **Session** Feld beinhaltet eine 128-Bit Random-UUID um die assoziierte Session zu identifizieren. Es wird im Rahmen des SE2-Projekts nicht verwendet, ist jedoch für zukünftige Erweiterungen reserviert.

7.2.1.1 Subtypen:

Subtyp	Wert	Bedeutung	Verwendung
Data	0	Daten-Nachricht	Text-Nachrichten
Control	1	Kontroll-Nachricht	Session-Control

Der Subtyp **Control** wird im Rahmen des SE2-Projekts nicht verwendet, ist jedoch für zukünftige Erweiterungen reserviert.

7.2.2 MSGTypeBlacklist - Blacklist Nachricht

Feld	Typ	Länge in Byte	Bedeutung	Optional
SubType	Byte[]	1	Subtyp der Nachricht	
Count	Integer	2	Anzahl Einträge	X
Blacklist	Byte[]	Count * 64	Blacklist Einträge	X
Hash	Byte[]	64	Hash der Blacklist	X
MAC	Byte[]	64	MAC	X

7.2.2.1 Subtypen:

Subtyp	Wert	Bedeutung	Verwendung
Hash	0	Blacklist Hash	Benachrichtigung durch den Server
Request	1	Blacklist Request	Anforderung durch den Client
Blacklist	2	Blacklist	Austausch der Blacklist

Der Subtyp **Hash** erzwingt das optionale Feld **Hash**. Eine Nachricht dieses Subtyps wird nur vom Server beim Verbindungsaufbau versendet.

Der Subtyp **Request** erzwingt das Fehlen der optionalen Felder und wird nur vom Client gesendet.

Der Subtyp **Blacklist** erzwingt die Felder **Count**, **Blacklist** und **MAC** und wird nur vom Server versendet.

7.2.3 MSGTypeRevocation - Revozierungs Nachricht

Feld	Typ	Länge in Byte	Bedeutung
PubKey	Byte[]	65	Public Key
Signatur	Byte[]	???	Signatur über den PubKey

Das Feld **PubKey** beinhaltet den revozierten Public Key. Im Feld Signatur wird die **Signatur** des revozierenden Clients über den Public Key übertragen.

7.2.4 MSGTypeNotification - Benachrichtigungs Nachricht

Feld	Typ	Länge in Byte	Bedeutung
Count	Byte[]	1	Anzahl Nachrichten im Bundle
Bundle	Byte[]	-	Nachrichten Bundle

7.2.5 MSGTypeMessageRequest - Nachrichten Anforderungs Nachricht

Feld	Typ	Länge in Byte	Bedeutung
Hash	Byte[]	64	Hash der letzten Nachricht