Bachelorarbeit

Architektur- und Sicherheitsanalyse von Tresorit und Tresorit DRM

Paul Rösler

Datum: 29. September 2015

Erstkorrekteur: Prof. Dr. Jörg Schwenk
Zweitkorrekteur: M. Sc. Christian Mainka

Betreuer: M. Sc. Christain Mainka und
B. Sc. Martin Grothe

Ruhr-Universität Bochum, Deutschland

Lehrstuhl für Netz- und Datensicherheit
Prof. Dr. Jörg Schwenk
Homepage: www.nds.rub.de

Erklärung

Ich erkläre, dass ich keine Arbeit in gleicher oder ähnlicher Fassung bereits für eine andere Prüfung an der Ruhr-Universität Bochum oder einer anderen Hochschule eingereicht habe.

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen, die anderen Quellen dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen kenntlich gemacht. Dies gilt sinngemäß auch für Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen.

Ich versichere hiermit auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Ich erkläre mich damit einverstanden, dass die digitale Version dieser Arbeit zwecks Plagiatsprüfung verwendet wird.

Ort, Datum	Unterschrift

Acknowledgements

The author would like to thank Szilveszter Szebeni, Zsolt Csóka-Veres and the Tresorit Team for providing a modified Client Version of Tresorit and supporting the thesis by answering additional questions regarding the system.
Additional thanks goes to Guiseppe Ateniese and Duane Wilson for providing information on their research regarding Tresorit.
Schließlich möchte der Autor Christian Mainka, Martin Grothe, Petra und Nadine Grünhagen, sowie Bertram, Petra und Karl Rösler für die Unterstützung während der Anfertigung der Arbeit danken.

Abstract

Tresorit ist eines der bekanntesten Ende-zu-Ende verschlüsselten Cloud Speicher Systeme. Die Entwickler geben vor, durch die zusätzliche Absicherung mittels Microsoft Rights Management Services, einen umfangreicheren Schutz als vergleichbare Systeme zu bieten. Die Untersuchung der Sicherheit dieses Systems und speziell der Kombination der Absicherungstechniken war Ziel dieser Arbeit.

Es konnte gezeigt werden, dass die zusätzliche Absicherung durch Microsoft Rights Management Services keinen erweiterten Zugriffsschutz vor Tresorit bot. Daraus folgte, dass das Vertrauen in Tresorit unabdingbar ist und die Schutzmaßnahmen nur gegenüber anderen Angreifern wirksam sind.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Abkürzungen

ACL	Access Control List; Zugriffssteuerungsliste
Base64	Kodierung von Binärdaten in Buchstaben, Zahlen, "+" und "-"
bzgl.	bezüglich
bzw.	beziehungsweise
CA	Certification Authority; Zertifizierungsstelle
CSV	Comma-separated Values; durch Komma getrennte Datensätze
DER	Formatierung für X.509 Zertifikate
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure; also durch TLS abgesichertes HTTP
JSON	JavaScript Object Notation
PKI	Public Key Infrastructure
TLS/SSL	Transport Layer Security / Secure Sockets Layer
X.509	Internationale Fernmeldeunion Standard für Zertifikate einer PKI
XML	Extensible Markup Language

1. Einführung

Die Entwicklung der Datenhaltung im Unternehmensumfeld warf und wirft Fragen und Probleme bezüglich der Datensicherheit auf. Bei der Verlagerung vom dezentralen Speichern auf dem Personal Computer hin zur internen zentralen Sammlung der Daten auf Servern wurde die Autorisierung mittels Authentifizierung sichergestellt. Die Entziehung der Zugriffsrechte konnte nicht ohne Weiteres durchgesetzt werden, da nach dem einmaligen Besitz der Daten keine Einschränkung des Zugriffs mehr möglich war. Aufgrund dessen erfolgte die zusätzliche Absicherung durch die Kopplung der Berechtigungen an die einzelnen Daten. Für die ortsunabhängige Erreichbarkeit und zur Kostenersparnis fand eine weitere Zentralisierung statt: Die Speicherung auf öffentlich erreichbaren Cloud-Speichern. Um die Absicherung des Zugriffs zu gewährleisten, wurden und werden rechtliche Vereinbarungen zwischen den Besitzern und Verarbeitern der Daten geschlossen. Nach wie vor ist der Verarbeiter in diesem Fall in der Lage auf die zentral gespeicherten Daten zuzugreifen.

Wenn die Autorisierung durch Verschlüsselung ergänzt beziehungsweise erzwungen wird, kann der zentrale, unautorisierte Zugriff verhindert werden. Die zusätzliche Absicherung durch die Kopplung der Berechtigungen an die Daten vervollständigt das Erzwingen der Autorisierung. Ein System, welches sowohl die nutzerseitige Verschlüsselung als auch die Kopplung der Berechtigungen an die Daten bietet, ist Tresorit. Die Kopplung der Berechtigungen an die Daten ist eine Erweiterung des Systems. Diese wird Tresorit DRM genannt.

Es wird im Folgenden analysiert, wie die Kombination der Autorisierungstechniken bei der Entwicklung von Tresorit und Tresorit DRM umgesetzt wurde. Dazu werden die Systemlandschaft und die Protokolle zwischen den interagierenden Entitäten untersucht. Anschließend werden mangelnde Absicherungen erläutert und die Wirksamkeit der implementierten Sicherheitsmaßnahmen überprüft.

Ziel dieser Arbeit ist das Erstellen einer Übersicht der Systemarchitektur und damit die Beschreibung aller Protokolle im System. Insbesondere soll damit die Integration von Tresorit DRM herausgestellt werden. Dies soll einer Sicherheitsevaluation und als Grundlage für weitere Untersuchungen des Systems dienen.

Zur Unterstützung dieser Arbeit stellten die Entwickler von Tresorit eine modifizierte Client-Version zur Verfügung, die ein- und ausgehenden Netzwerkverkehr mitschneidet[1]. In Kombination mit Beobachtungen

[1] Die Bereitstellung der genutzten modifizierten Client-Version von Tresorit wurde mit einem Sperrvermerk beschränkt, der die Weitergabe der Software untersagt, die Verarbeitung und Veröffentlichung der damit gewonnenen Erkenntnisse aber ausdrück-

aus Software für die Überwachung der Speicher-, Registry- und Netzwerkaktivitäten, sollen diese Mitschnitte Aufschluss auf das Verhalten der Client-Software von Tresorit liefern. Mit Hilfe des beobachteten Verhaltens sollen die Protokolle im System ergründet werden. Aus diesen Protokollen soll dann erschlossen werden, wie Tresorit DRM die Integration von Microsoft Rigths Management Services erreicht.

Abzugrenzen ist diese Methode von Reverse Engineering. Es wurde bereits eine Analyse der Schlüsselzertifizierung in Tresorit mittels Reverse Engineering durchgeführt (siehe Abschnitt 6.3).

lich erlaubt.

2. Hintergrund

2.1. Definitionen

In diesem Abschnitt sollen grundlegende Konzepte und Systeme für das Verständnis dieser Arbeit erläutert werden.

2.1.1. Information Rights Management

Information Rights Management (IRM) ist die Verwaltung von Berechtigungen von Daten. Die Verwaltung bezieht sich auf die Vergabe, Entziehung und Durchsetzung von Berechtigungen. Berechtigungen können Lese- und Schreibzugriff, sowie das Weiterleiten von Berechtigungen sein. Außerdem können systemspezifische Berechtigungen, wie das Recht, Daten zu drucken oder Inhalte zu kopieren, dazugehören. Dabei sind die Berechtigungen an die Daten gekoppelt, sodass der Besitz der Daten nicht zum vollständigen Zugriff führt. IRM wird größtenteils im professionellen Umfeld, also in Unternehmen genutzt, um den Zugriff auf Daten zu steuern und damit Vertraulichkeit zu erreichen. Synonym dazu wird der Begriff *Enterprise Rights Management* (ERM) genutzt.

Der Begriff *Digital Rights Management* (DRM) meint das Pendant zu IRM zur Durchsetzung des Urheberrechts der Erzeuger und Anbieter von Daten gegenüber Konsumenten. Da diese Bedeutung von DRM für diese Arbeit nicht relevant ist und Tresorit DRM die Verwaltung von Berechtigungen nach der Definition von IRM ist, werden in dieser Arbeit IRM, ERM und DRM synonym verwendet.

2.1.1.1. Einführung in Microsoft Rights Management Services

Microsoft Rights Management Services (RMS) ist eine Umsetzung des Information Rights Managements. Dabei fungiert eine zentrale Entität als vertrauenswürdige Institution zur Verwaltung der Berechtigungen. Die Berechtigungen werden dezentral, zusammen mit den geschützten Daten gespeichert. Spezielle Software, wie Microsoft Office, sorgt beim Zugriff auf die Daten für die Durchsetzung der Berechtigungen.

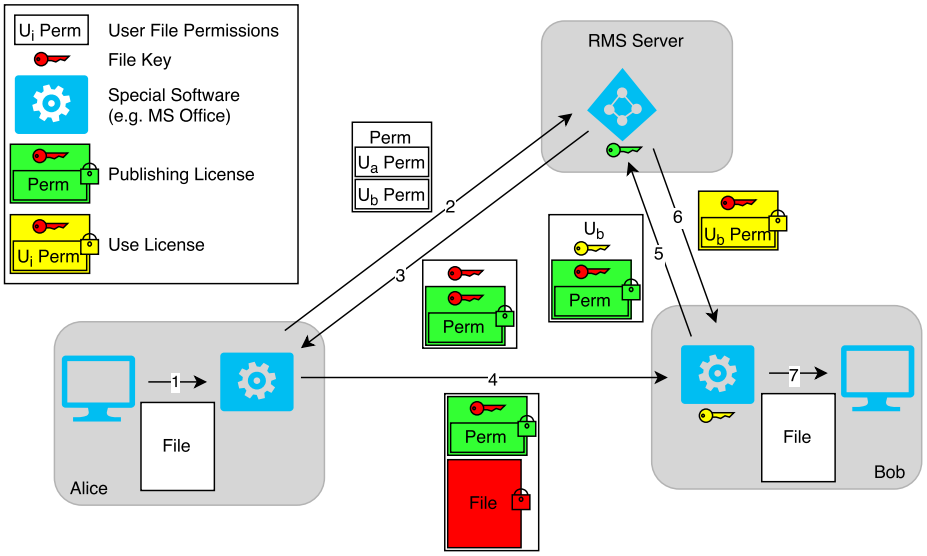

Abbildung 2.1.: Schematische Darstellung von Microsoft RMS[1]

Die Berechtigungen einer zu schützenden Datei werden an den zentralen RMS Server gesendet (2 in Abbildung 2.1), der diese zusammen mit einem Datei-Schlüssel mit dem eigenen öffentlichen Schlüssel verschlüsselt. Dieses Chiffrat wird Publishing License genannt. Der Server sendet die Publishing License und den unverschlüsselten Datei-Schlüssel zurück zum Ersteller der Datei (3). Dieser verschlüsselt die Datei mit dem Datei-Schlüssel und hängt die Publishing License an die verschlüsselte Datei. Die so geschützte Datei kann dann an andere Nutzer gesendet werden (4).

Sobald ein anderer Nutzer diese Datei mit einer geeigneten Software öffnet, wird die Publishing License an den RMS Server gesendet (5). Der Server extrahiert daraus die Berechtigungen und den Datei-Schlüssel. Wenn der Nutzer zugriffsberechtigt ist, sendet der Server den Datei-Schlüssel und die Berechtigungen des anfragenden Nutzers an die öffnende Software (6). Diese entschlüsselt die Datei mit dem Datei-Schlüssel und setzt die Berechtigungen des Nutzers durch (7).

2.1.2. Ende-zu-Ende Verschlüsselung in Cloud Storage Systemen

Ein *Cloud Storage System* besteht aus Storage Servern und der Client-Software. Die Client-Software kann auf mehreren Geräten eines Nutzers installiert sein. Sie sorgt dafür, dass die gewählten Daten auf allen Geräten des Nutzers synchronisiert sind. Dazu sind die aktuellen Daten immer auf den Storage Servern gespeichert. Ein Nutzer kann einem anderen Nutzer des Cloud Storage Systems Zugriffsberechtigungen zu seinen Daten erteilen und entziehen. Die Synchronisation findet dann zwischen allen Geräten aller berechtigten Nutzer statt.

[1] Aus Erkenntnissen dieser Arbeit in Verbindung mit `https://technet.microsoft.com/de-de/library/JJ585026.aspx`.

Ende-zu-Ende Verschlüsselung in Cloud Storage Systemen bedeutet, dass die geschützten Daten von keinem gelesen werden können, außer den berechtigten Nutzern. Dazu werden die Daten vor der Speicherung auf dem Storage Server nutzerseitig verschlüsselt. Die verwendeten Schlüssel werden nur zwischen den berechtigten Nutzern kommuniziert und gelangen somit weder zum Server noch zu anderen Nutzern.

Ein Ansatz zur vertrauenswürdigen Verteilung der Nutzer-Schlüssel ist der Aufbau einer Public-Key-Infrastruktur. Unter Vertrauen zum Aussteller bzw. Service Provider werden die öffentlichen Schlüssel der Nutzer zertifiziert und zentral für andere Nutzer bereitgestellt. So können symmetrische Schlüssel, mit denen die zu schützenden Daten verschlüsselt sind, mit den öffentlichen Schlüsseln der berechtigten Nutzer verschlüsselt werden.

2.2. Angreifermodell

Nach einer Voruntersuchung der Dokumentation (siehe 2.3.1) wurden die Entitäten im untersuchten System und deren Relationen zueinander identifiziert. Mögliche Angriffsvektoren wurden aus der daraus erlangten Konzeptarchitektur abgeleitet.

Physikalisch von einander getrennte Entitäten waren die Tresorit Server, die Microsoft RMS Server, das Gerät des Nutzers und das Gerät eines möglichen Kontakts des Nutzers. Zusätzlich zum Hauptgerät des Nutzers konnte dieser weitere Geräte verwenden, auf denen die Tresorit Client-Software installiert und auf denen er mit seiner Nutzerkennung angemeldet war. Die Tresorit Client-Software konnte auf verschiedenen Betriebssystemen installiert werden. Während der Voruntersuchung konnte festgestellt werden, dass die Client-Software für Windows mit den meisten Funktionen ausgestattet war. Aus diesem Grund wurde nur die Tresorit Version 2.1.501.344 unter Windows 7 betrachtet. Auf einem Nutzergerät konnte zusätzlich zur Tresorit Client-Software Tresorit DRM installiert sein. Zur Nutzung der DRM-Funktionalitäten war des Weiteren Microsoft Office installiert.[2] Außerdem konnte ein Webbrowser zur Administration der Nutzerdaten und zum Öffnen von Encrypted Links [1] und ein Mail Client zur Kommunikation mit Kontakten installiert sein. Der Tresorit Server wurde aufgeteilt in den Server, der die Kommunikation mit der Client-Software verarbeitet, und den Webserver, der das Webinterface zur Administration der eigenen Daten und zum Download via Encrypted Link zur Verfügung stellte.

Für eine Analyse getrennt betrachtete Entitäten der Systemarchitektur könnten sein: Tresorit Client-Software auf verschiedenen Geräten, Tresorit DRM, Tresorit Server, Tresorit Webserver, Microsoft Office und Microsoft RMS Server. Neben den physikalisch erreichbaren Entitäten könnten die Protokolle zwischen diesen untersucht werden. Die identifizierten Entitäten und Protokolle der skizzierten Systemarchitektur sind mit den zugeordneten Aufgaben und möglichen Angriffsszenarien in Tabelle 2.2 aufgelistet.

[2] Tresorit DRM und Microsoft Office nicht notwendig für Nutzung von Tresorit, aber Bestandteil dieser Analyse

Entität/Protokoll	**Aufgabe**	**Angriffsvektor**
Tresorit Client	Schlüsselgenerierung bei Installation	Key freshness/randomness, Kryptanalyse
	Speicherung der Schlüssel	Auslesen , Modifizieren
	Verschlüsselung der gespeicherten Produktivdaten[3]	*Auslesen*, *Modifizieren*, *Kryptanalyse*
	Verifizierung der Protokollnachrichten	Angriff trotz Absicherung
	Agreement Module	Hinzufügen weiterer Nutzer
Tresorit Protokolle	TLS Kanal zwischen Client und Server	Auslesen , Modifizieren
	Registrierung	Auslesen, Modifizieren
	Schlüsselzertifizierung	Ausstellen gefälschter Zertifikate
	Login → Schlüsselverteilung	Kryptanalyse gegen $roaming.profile$
	Passwortänderung	Auslesen, Modifizieren, Relay, Replay, Typing Attack, Kryptanalyse
	Verschlüsselung und Signierung der Daten → Root Reencryption und Lazy-Reencryption	Kryptanalyse: Auslesen, Modifizieren
	Kontaktaufnahme → ICE Protokoll[4] zur Vereinbarung eines Langzeitschlüssels	Auslesen, Modifizieren
	ACL Update	Blockieren
Tresorit Server	Verifizierung der Protokollnachrichten	Angriffe trotz Absicherung
Tresorit DRM/ Microsoft RMS mit Protokollen	Installation → Authentifizierung	Key freshness, Kryptanalyse , Webseitenangriffe [5] auf SSO Protokoll
	Share und Revoke DRM geschützter Dateien/Tresors	Auslesen, Modifizieren, Blockieren
	Öffnen DRM geschützter Dateien/Tresors → Lokales Zwischenspeichern der Daten → Kommunikation mit Office	Auslesen, Modifizieren, Blockieren
Tresorit Webinterface	Nutzerverwaltung	Webseitenangriffe
	Aufruf Encrypted Link	Auslesen Browserhistorie
	Anzeige Encrypted Link	Webseitenangriffe
Microsoft Office	Erzwingen der Rechte → Speichern, Screenshot, Copy/Paste, Drucken	Umgehen der Beschränkungen
Tresorit App	Siehe Tresorit Client	
Tresorit App Protokolle	Siehe Tresorit Protokolle → Besonderheiten der Zertifizierung und Kommunikation	
Microsoft RMS Server	Verifizierung der Protokollnachrichten	Angriffe trotz Absicherung

Tabelle 2.2.: Entitäten und Protokolle im System mit Aufgaben und möglichen Angriffsszenarien

[3] Noch nicht implementiert
[4] Proprietäres Kontaktaufnahmeprotokoll von Tresorit
[5] XSS, CSRF, SQLi, Cookie Diebstahl, ...

Der Fokus dieser Arbeit lag auf der Beschreibung der Protokolle, der Analyse der Integration von Tresorit DRM in Tresorit und der Sicherheitsevaluierung der analysierten Protokolle und Entitäten. Daher wurden das Tresorit Webinterface, Microsoft Office, Tresorit App und Microsoft RMS Server nicht untersucht. Zur Darstellung eines vollständigen Angreifermodells wurden diese Entitäten und Protokolle trotzdem aufgelistet.

Entgegen der ersten Annahme, dass die Netzwerkprotokolle gezielt gegen Angriffe wie Auslesen, Modifizieren, Relay, Replay und Typing Attack [2] geschützt sind, wurde festgestellt, dass die Nachrichten insgesamt mittels TLS abgesichert waren. Gezielte weitere Sicherungsvorkehrungen wurden nur vereinzelt getroffen.

2.2.1. Sicherheitsannahme

In der wissenschaftlichen Ausarbeitung des ursprünglichen Protokolls von Tresorit (*Tresorium*) [3] wird der Operator und der Service Provider als "honest but curious" definiert. Der Service Provider ist der Administrator bzw. Betreiber des Storage Servers, auf den Daten von Tresorit geladen werden. Operators sind weitere Nutzer desselben Servers mit physikalischem oder virtuellem Zugang.

Es wird angenommen, dass diese beiden Gruppen von Angreifern "ehrlich aber neugierig" sind. Da keine weiteren Definitionen der Angreifer im System Tresorit existieren, wird diese Annahme auf die Service Provider aller Tresorit Server ausgeweitet.

Im Whitepaper zu Tresorit DRM [4] wird eine Aussage zur Sicherheit von Tresorit DRM geschützten Daten gemacht:
"Tresorit hat keinen Zugriff auf Content Keys, die auf den Microsoft RMS Servern gespeichert sind. Daher kann Tresorit, selbst wenn es Zugriff auf DRM geschützte Dateien des Nutzers hätte, diese Dateien nicht [Micrsoft RMS] entschlüsseln und somit auf den Inhalt dieser Dateien zugreifen." [4] (übersetzt)
Diese Aussage lässt darauf schließen, dass nur eine der beiden Schutzmaßnahmen - entweder Tresorit oder Microsoft RMS (Tresorit DRM) Verschlüsselung - notwendig ist, damit die Vertraulichkeit unter der Sicherheitsannahme gewährleistet ist. Es konnte gezeigt werden, dass dies nicht zutrifft (siehe Abschnitt 5.2)

2.3. Methodik

Zur Untersuchung der genannten Entitäten und Relationen in der Architektur wurde ein Vorgehen in drei Schritten gewählt: Zuerst wurde die Dokumentation und relevante Literatur analysiert, um die erwartete Systemarchitektur zu beschreiben. Danach wurden die realen Entitäten und Relationen in einer Testumgebung beobachtet und beschrieben. Die dabei gefundenen Schwachstellen wurden letztlich evaluiert.

2.3.1. Beschreibung der Dokumentation

Zur Untersuchung der Dokumentation standen zunächst die zwei wissenschaftlichen Veröffentlichung *Tresorium* [3] und *I-TGDH* [5] zur Speicher- und Schlüsselverwaltung und zur Berechnung des Gruppenschlüssels zur Verfügung. Außerdem wurden durch Tresorit veröffentlichte Produktbeschreibungen bezüglich *Tresorit* allgemein [6], *Tresorit DRM* [4], *Encrypted Link* [1] und Registrations- und Anmeldeprotokollen[7] verwendet. Des Weiteren wurden Beschreibungen im *Online Security Support Forum von Tresorit* [8] berücksichtigt. Letztlich wurden auch die Erkenntnisse einer wissenschaftlichen *Untersuchung zur Schlüsselzertifizierung* [9] von Tresorit durch Wissenschaftler der Johns Hopkins University, Baltimore, Maryland, USA genutzt.

Die Verfahren, Protokolle und Zusammenhänge zu den zu untersuchenden Entitäten und Relationen wurden jeweils zusammengetragen. Die Beschreibungen in den genannten Dokumenten unterschieden und widersprachen sich teilweise aufgrund der Aktualität.

2.3.2. Beschreibung der Implementierung

Der Quellcode von Tresorit war nicht einsehbar, daher musste die Analyse der Implementierung durch Beobachtung der Software durchgeführt werden. Hierzu wurden zwei Methoden gewählt: Beobachten der Daten auf dem Client und Beobachtung der Netzwerkprotokolle. Ziel dessen war die vollständige Erfassung der Protokolle von Tresorit.

Da die Netzwerkverbindung zwischen der Software und den Servern mittels TLS abgesichert war und ein Unterbrechen des TLS-Kanals mit gefälschten CA-Zertifikaten erkannt wurde und zum Abbruch der Kommunikation führte, konnte die Untersuchung nicht ohne Modifizierung der Software durchgeführt werden. Die Tresorit Entwickler stellten für diese Arbeit eine modifizierte Client-Software zur Verfügung, die ausgehenden und eingehenden Netzwerkverkehr mitschnitt. Die Version des modifizierten Clients entsprach *Tresorit 2.1.501.344*. Diese modifizierte Version der Tresorit Client-Software wurde für die gesamte Analyse verwendet.

Zur Analyse der Implementierung wurde eine virtuelle Maschine mit *Oracle VM VirtualBox 4.3.30* erstellt. Auf dieser wurden nach einander *Windows 7 Professional N 64 Bit Service Pack 1*, *Microsoft Office 365 (MSO 15.0.4737.1002)*, *Firefox 39.0*, *Microsoft .NET Framework 4 Client Profile*, alle *Windows Updates bis zum 27.07.2015 um 08:00 Uhr*, *Oracle VM VirtualBox Guest Additions 4.3.30*, *Process Monitor 3.20*, *Wireshark 1.12.6* und *WinPcap 4.1.3* installiert. Außerdem wurde die Testdatei *Testfile.docx* mit der installierten Microsoft Word Version erstellt. Diese Datei wurde zur Beobachtung der Protokolle genutzt. Eine schemenhafte Darstellung des Versuchsaufbaus ist in Abbildung 2.2 gegeben.

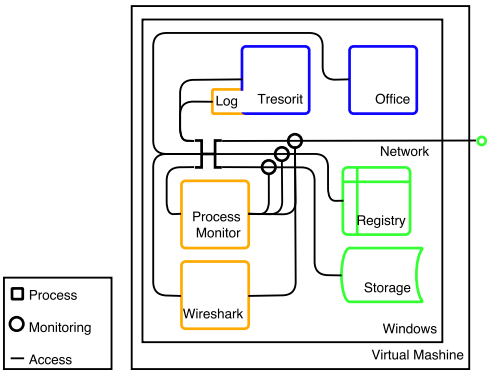

Abbildung 2.2.: Versuchsaufbau zur Beobachtung der Implementierung

In Tabelle 2.4 sind alle Schritte je virtueller Maschine nach der Instanziierung (X100) aufgelistet. Für den Nutzer Alice wurden während der Analyse zwei Maschinen erstellt, für Nutzer Bob eine Maschine.

Alice Device 1	**Alice Device 2**	**Bob Device 1**
Installieren der modifizierten Tresorit Software (A102)	*Duplizieren der Maschine A102*	*Duplizieren der Maschine A102*
Registrieren des Nutzers Alice (A103)		
	Login des Nutzers Alice (A202)	
	Synchronisieren des Standardtresors (A203)	
Kopieren der Testdatei in Standardtresor und Upload = Synchronisierung (A104)		
	Download der Datei (A204)	
		Registrieren des Nutzers Bob
Kontakt Bob hinzufügen und Manager-Berechtigung für Standardtresor erteilen = Share (A105)		
		Kontakt Alice und Berechtigung für Tresor annehmen (B102)
		Tresor von Alice synchronisieren (B102a)
Empfangsbestätigung von Bob bzw. System bzgl. Share erhalten (A105a)		
Bob die Zugriffsberechtigung auf Standardtresor entziehen = Revoke (A106)		
		Berechtigungsentzug erhalten (B103)
Nutzer-Passwort ändern (A107)		
Tresorit DRM Berechtigung beantragen und erhalten		*Tresorit DRM Berechtigung beantragen und erhalten*

Alice Device 1	Alice Device 2	Bob Device 1
Tresorit DRM aktivieren und installieren (A108)		*Tresorit DRM aktivieren und installieren*
Erstellen eines Tresorit DRM geschützten Tresors (A109)		
Kopieren der Testdatei in DRM geschützten Tresor und Upload = Synchronisierung (A109a)		
Kontakt Bob Manager-Berechtigung für DRM geschütztem Tresor erteilen (A109b)		
		Berechtigung zu DRM geschütztem Tresor annehmen (B104)
		Testdatei mit Microsoft Word öffnen (B105)
Bob die Zugriffsberechtigung für DRM geschütztem Tresor entziehen = Revoke (A110)		
		Berechtigungsentzug erhalten (B106)
		Testdatei mit Microsoft Word versuchen zu öffnen (B107)

Tabelle 2.4.: Arbeitsschritte während der Analyse der Implementierung, kursiv gekennzeichnete Schritte wurden ohne Beobachtung durchgeführt, Kürzel in Klammern geben den Namen der erstellten Sicherungskopie an

2.3.2.1. Clientdatenanalyse

Während jedes einzelnen Arbeitsschritts wurden die Datei- und Registryzugriffe sowie die Netzwerkverbindungen mittels Process Monitor aufgezeichnet. Außerdem wurde der Systemzustand nach jedem Arbeitsschritt mit einem Sicherungspunkt in VirtualBox gespeichert. Die mit Process Monitor erfassten Netzwerkverbindungsdaten enthielten lediglich Metadaten.

Die Auswahl der betrachteten Dateien und Registry-Keys fand mittels Filterung nach Schreibvorgängen statt: Alle Daten, die während der beobachteten Aktionen erzeugt oder beschrieben wurden, waren Bestandteil der Analyse. Nur beschriebene Dateien konnten Daten enthalten, die von Tresorit und Tresorit DRM erzeugt oder durch diese verarbeitet wurden. Zusätzlich wurde die genannte Testdatei in Betracht gezogen (siehe Abschnitt 2.4.2).

Zuerst sollten die vollständigen virtuellen Festplatten nach jedem Arbeitsschritt kopiert und jeweils miteinander verglichen werden. Aufgrund der großen Datenmengen bei gleichzeitig ungenaueren Ergebnissen,

verglichen zur Analyse mittels Process Monitor, wurde diese Herangehensweise nicht weiter verfolgt.

2.3.2.2. Netzwerkdatenanalyse

Die Netzwerkkommunikation wurde mit Wireshark, Process Monitor und den internen Mitschnitten der modifizierten Tresorit Version beobachtet. Mit Hilfe der Metadaten in Kombination mit den Daten der Dateizugriffe konnten die Netzwerkdaten aus den drei Quellen zusammengeführt werden. Die Daten aus Process Monitor konnten zur Einordnung und Sortierung der Abläufe genutzt werden, während die Mitschnitte der modifizierten Tresorit Version mit den Klartextdaten der Netzwerkkommunikation Aufschluss auf die Inhalte der Netzwerkprotokolle lieferten.

2.4. Protokollanalyse

2.4.1. Rohdaten

Nach der Beobachtung der Aktionen lagen die Daten von Process Monitor, Wireshark und der modifizierten Client Version von Tresorit vor. Die mitgeschnittenen Wireshark Daten wurden nur für die Verifizierung der anderen Daten genutzt und flossen daher nicht direkt mit in die Analyse ein. Die Daten aus Process Monitor konnten im CSV-Format exportiert werden. Die Log-Daten des modifizierten Tresorit Clients lagen im JSON-Format vor. Diese Log-Daten enthielten die Produktiv-POST-, GET- und Antwort-Daten sowie einen Zeitstempel und den HTTP-Statuscode der HTTPS Requests, die durch den Tresorit Client ausgelöst wurden.

Zur einheitlichen Analyse wurden die Daten aller Aktionen aus Process Monitor in einer Tabelle gesammelt. Diese Daten wurden anschließend gefiltert (siehe Abschnitt 2.4.2). Nach der Filterung wurden die Log-Daten des modifizierten Clients in die Tabelle eingefügt.

Da die Mitschnitte von Process Monitor nur die IP-Adresse, den Zeitpunkt und die Datengröße einer Netzwerknachricht enthielten, musste ein Abgleich mit den Log-Daten stattfinden. Es konnte ein Unterschied der Zeitstempel der beiden Quellen von bis zu 30 Sekunden festgestellt werden. Als Grund dafür wurde das Stoppen und Weiterlaufen der virtuellen Maschinen vermutet. Um die Daten korrekt einzuordnen, musste deshalb auf IP-Adresse, Zeitstempel, Größe und Dateizugriffe auf den lokalen Speicher zurückgegriffen werden. Bei der Zusammenführung der Mitschnitte blieben Stellen, an denen diese Parameter nicht eindeutig den Zugriffszeitpunkt bestimmten.

Die Log-Daten des modifizierten Clients gaben bei Uploads und Downloads keinen Aufschluss auf die übertragene Datei. Durch die Ausgabe der Produktivdaten im JSON-Format konnte nicht ausgeschlossen werden, dass weitere Daten in den Nachrichten übertragen wurden. Bei den Nachrichten GetMessages() (siehe Abschnitt 3.7) und GetPublishingLicense() (siehe Abschnitt 4.5) wurde daher davon ausgegangen, dass die vorliegenden Daten nicht vollständig waren.

2.4.2. Filterung der Events

Die mit Process Monitor gesammelten Daten wurden nacheinander mit folgenden Kriterien gefiltert[6]:

Filter	Begründung
Prozesse: tresorit.exe, von tresorit.exe aufgerufen, WINWORD.EXE	Es stellten sich keine weiteren Abhängigkeiten zu anderen Prozessen heraus. Lediglich die Erkennung von Speicherzugriffen zur Synchronisierung wurde durch andere Prozesse durchgeführt. Da der tatsächliche Dateizugriff durch Tresorit bei Up- und Download entscheidend war, wurden die Prozesse zur Speicherzugriffserkennung nicht betrachtet.
Operationen: ReadFile, WriteFile, RegQueryValue, RegSetValue, TCP Send, TCP Receive	Da weitere Operationen nur zur Vor- oder Nachbereitung des Zugriffs bzw. des Verbindungsaufbaus dienten, wurde der Fokus auf die genannten Operationen gelegt[7].
Pfade: Während der Beobachtung mindestens einmal beschriebene Datei- bzw. Registry- Pfade	Weil nur diese Dateien Inhalt der Protokolle enthalten konnten, wurden andere Daten nicht betrachtet.
Mehrfach ausgeführte selbe Operation des selben Prozesses auf selben Pfad	Da aus den Metadaten der Zugriffe nicht ablesbar war, welcher Inhalt einer Datei beschrieben oder gelesen wurde, konnte die Zugriffsinformation auf die Art des Zugriffs beschränkt werden.
Mehrfachübertragung aufgrund von Übertragungsfehlern (manuell)	Wegen der Blockierung des Netzwerkverkehrs zur Verzögerung der Beobachtungen und der beschränkten Verbindungsgeschwindigkeit in der Testumgebung wurden einige Anfragen mehrfach versandt (erkennbar an identischem Inhalt der Anfrage und der Antwort).
Log Daten (manuell)	Daten und Netzwerkverbindungen, die zum Logging dienten, wurden herausgefiltert.
Temporäre Daten (manuell)	Temporäre Daten, die erkennbar keinen weiteren Einfluss auf den Protokollfluss hatten, wurden aus Gründen der Übersichtlichkeit ausgelassen.
Tresorit Gamification (manuell)	Zum Erlangen von zusätzlichen Nutzungsmöglichkeiten bot Tresorit bei bestimmtem Nutzungsverhalten kostenlose Zusatzfunktionen. Die betreffende Datenübertragung wurde nicht betrachtet.

Tabelle 2.6.: Kriterien zur Filterung der beobachteten Daten

Beispielhaft kann die Filterung an Abbildung 2.3 nachvollzogen werden. Ziel des Zugriffs war vermutlich das Beschreiben der Datei *C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit\Temp\s5mw2rcwv31oy uwkcx5ioz8x5gvk3y*. Bis auf den Schreibzugriff wurden alle anderen Zugriffe dieses Ausschnitts herausgefiltert.

Während der Beobachtung der Netzwerkdaten konnten, von den jeweiligen Aktionen unabhängig, regelmäßige Nachrichten erkannt werden. Diese sind separat im Abschnitt *Regelmäßiges Polling* beschrieben.

[6] Die Daten können vorgefiltert in der Datei *ProcMonEventsFiltered.ods* beiliegend auf der CD oder im SVN des NDS eingesehen werden.

[7] Initial wurden alle Zugriffe betrachtet, um keine Aktionen zu übersehen. Wenn dieser Filter ausgelassen wurde, ist dies gekennzeichnet.

Time of ...	Process N...	Operation	Path
13:43:01...	Tresorit.exe	CreateFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit
13:43:01...	Tresorit.exe	QueryNetworkOpenInfor...	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit
13:43:01...	Tresorit.exe	CloseFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit
13:43:01...	Tresorit.exe	CreateFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit
13:43:01...	Tresorit.exe	QueryNetworkOpenInfor...	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit
13:43:01...	Tresorit.exe	CloseFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit
13:43:01...	Tresorit.exe	CreateFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit
13:43:01...	Tresorit.exe	SetBasicInformationFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit
13:43:01...	Tresorit.exe	CloseFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit
13:43:01...	Tresorit.exe	CreateFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit\Temp
13:43:01...	Tresorit.exe	QueryNetworkOpenInfor...	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit\Temp
13:43:01...	Tresorit.exe	CloseFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit\Temp
13:43:01...	Tresorit.exe	CreateFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit\Temp\s5mw2rcwv31oyuwkcx5ioz8x5gvk3y
13:43:01...	Tresorit.exe	NotifyChangeDirectory	C:\Users\Tresorit\My Tresors\Tresor von Franz
13:43:01...	Tresorit.exe	WriteFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit\Temp\s5mw2rcwv31oyuwkcx5ioz8x5gvk3y
13:43:01...	Tresorit.exe	NotifyChangeDirectory	C:\Users\Tresorit\My Tresors\Tresor von Franz
13:43:01...	Tresorit.exe	ReadFile	C:
13:43:01...	Tresorit.exe	WriteFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit\Temp\s5mw2rcwv31oyuwkcx5ioz8x5gvk3y
13:43:01...	Tresorit.exe	WriteFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit\Temp\s5mw2rcwv31oyuwkcx5ioz8x5gvk3y
13:43:01...	Tresorit.exe	CloseFile	C:\Users\Tresorit\My Tresors\Tresor von Franz\.tresorit\Temp\s5mw2rcwv31oyuwkcx5ioz8x5gvk3y
13:43:01...	Tresorit.exe	NotifyChangeDirectory	C:\Users\Tresorit\My Tresors\Tresor von Franz

Abbildung 2.3.: Ungefilterter Ausschnitt der Process Monitor Daten des Downloadprotokolls

2.4.3. Definition und Erläuterung der Protokollelemente

Aus der Syntax der beobachteten Nachrichten konnte gefolgert werden, dass ein Container eine Speichereinheit in den Protokollen von Tresorit ist. Eine Speichereinheit wurde definiert als Einheit, die eindeutig referenziert werden kann und zu der weitere Elemente (speziell Dateien) zugeordnet werden können. Zur Verwaltung der Berechtigungen zu Dateien auf Containern dienen Tresors. Diese beinhalten die Berechtigungen zu Containern. Im Folgenden sollen die Relationen zwischen Nutzer, Tresor und Container erörtert werden. Eine ausführliche Beschreibung der beobachteten Nachrichten ist im Anhang (A.1) einsehbar.
Der Syntax der Nachricht GetTresorMembers() konnte entnommen werden, dass einem Nutzer-Container Paar genau ein Tresor zugeordnet wird. Also:

$$\exists t : \ (u, c) \mapsto t \ \forall \ u \in U, c \in C, t \in T$$

U ist die Menge der Nutzer, C die Menge der Container und T die Menge der Tresors.

Gleichzeitig ging aus der Syntax hervor, dass ein Tresor für verschiedene Nutzer in verschiedenen Containern gespeichert sein konnte. Also:

$$t \mapsto \{(u_i, c_j)\} \ \forall \ t \in T, u \in U, c \in C, i, j \in \mathbb{N}$$

Logisch konnte weiterhin gefolgert werden, dass zu einem Container nur ein Tresor zugeordnet werden konnte: Da ein Tresor mit Hilfe des Nutzerzertifikats und der Container-ID angesprochen wurde, konnte für einen Nutzer nur ein Tresor in einem Container liegen. Falls zwei unterschiedliche Nutzer $u, v \in U$ verschiedene Tresors $s, t \in T$ auf einem Container $c \in C$ nutzen, gilt $(u, c) \mapsto t \wedge (v, c) \mapsto s$. Wenn dann ein Nutzer u einem Nutzer v die Zugangsberechtigung zu seinem Tresor s erteilt, würde gelten

$(u, c) \mapsto s \wedge (v, c) \mapsto t \wedge (v, c) \mapsto s$. Dies ist ein Widerspruch, wenn $t \neq s$. Also gilt:

$$c \mapsto t, t \mapsto \{(c_j)\} \; \forall \; c_j \in C, t \in T, j \in \mathbb{N}$$

Die Zuordnung mehrerer Container zu einem Tresor wurde auf die damit ermöglichte Skalierbarkeit des Systems zurückgeführt. Zumal keine Zuordnung mehrerer unterschiedlicher Container zu einem Tresor beobachtet werden konnte und diese für die Protokolle keine Auswirkung hätte, wird im Folgenden angenommen, dass die Container-Tresor-Zuordnung eine eins-zu-eins Relation ist:

$$c \mapsto t, t \mapsto c \; \forall c \in C, t \in T$$

Tresorit bietet bei der Erteilung von Berechtigungen ohne und mit Microsoft RMS die Möglichkeit zwischen Owner, Manager, Editor und Reader zu unterscheiden. Diese Möglichkeit war für die verwendeten Test-Nutzer nicht freigeschaltet. Aufgrund dessen konnte nur eine Unterscheidung zwischen Zugriff und Zugriffsverweigerung stattfinden. Diese Einschränkung wurde in die Sequenzdiagramme übernommen. Die Vereinfachungen sollen zur Übersichtlichkeit der Protokoll-Sequenzdiagramme beitragen und folgen aus der Tatsache, dass mangels Beobachtung keine weiterführenden Aussagen getroffen werden konnten. Die ursprünglichen Elemente und deren Attribute der Protokollarchitektur von Tresorit und die Vereinfachungen sind in Tabelle 2.8 aufgeführt.

Element	**Erläuterung**	**Vereinfachung**
$User$	Nutzer-Objekt; es wurden zwei Nutzer erstellt und verwendet	A, B
⌞ $name$	Vollständiger Name des registrierten Nutzers	$name_{user}$
⌞ $mail$	Gewählte E-Mail Adresse des Nutzers; wird in den Protokollen als Identifikationsmerkmal verwendet	$mail_{user}$
⌞ pwd	Gewähltes Passwort des Nutzers	pwd_{user}
⌞ $salt$	Salt zur Randomisierung des gewählten Passworts	$salt_{pwd_{user}}$
⌞ $saltedPwd$	Authentifizierungsgeheimnis gewonnen aus gewähltem Nutzer-Passwort (siehe Abschnitt 3.1)	$authSecret_{user}$
⌞ $clientResponse$	Antwort auf Challenge-Response-Protokoll zur Authentifizierung (siehe Abschnitt 3.1)	$clientResponse_{user}$
⌞ RMS	Attribut das angibt, ob der Nutzer zur Microsoft RMS-Nutzung zugelassen ist	$on_{RMS_{user}}$, $off_{RMS_{user}}$
⌞ $mail$	Zugewiesene E-Mail Adresse zur Nutzung von Microsoft RMS	$mail_{RMS_{user}}$
⌞ pwd	Zugewiesenes Passwort zur Nutzung von Microsoft RMS	$pwd_{RMS_{user}}$
⌞ $roaming.profile$	Synchronisierte Datei, die Nutzerinformationen enthält; ist verschlüsselt lokal gespeichert, daher nicht explizit mit enc(...) angegeben (siehe Abschnitt 3.1)	rpF_{user}

Element	Erläuterung	Vereinfachung
∟ $device.profile$	Lokale Datei, die Nutzerinformationen gerätabhängig enthält; ist verschlüsselt lokal gespeichert, daher nicht explizit mit enc(...) angegeben (siehe Abschnitt 3.1)	$dpF_{device_{user}}$
∟ $securelogin$	Datei, die zur Nutzung des automatischen Logins lokal gespeichert ist (siehe Abschnitt 3.1)	$securelogin_{user}$
∟ $cert$	Zertifikat bzw. Zertifikat-Request (siehe Abschnitt 3.2)	$cert_{user}$, $cert_{user}^{request}$
∟ $user$	Hauptzertifikat des Nutzers	$cert_{user}$
∟ $device$	Neben gerätunabhängigen Zertifikaten existieren Zertifikate für die Geräte	
∟ ssl	SSL-Zertifikat für jedes Gerät	$ssl_{device_{cert_{user}}}$
∟ $encrypt$	Verschlüsselungs-Zertifikat für jedes Gerät	$enc_{device_{cert_{user}}}$
∟ $sign$	Signatur-Zertifikat für jedes Gerät	$sign_{device_{cert_{user}}}$
∟ $anonym$	Anonymes Zertifikat zur Kontaktaufnahme mit anderen Nutzern	$anon_{cert_{user}}$
∟ $agree$	Zertifikat zur Kontaktaufnahme mit anderen Nutzern	$agree_{cert_{user}}$
∟ $pairID$	ID des Zertifikatpaares aus Agreement- und Anonymous-Zertifikat; bei Sharing und Revocation wird die des neuen Nutzers in Group Key File angegeben	$pairID_{cert_{user}}$
∟ $intermediate$	Zwischenzertifikat zwischen Tresorit Root CA Zertifikat und ausgestelltem Zertifikat (siehe Abschnitt 3.2); wurden jeweils zu jedem übertragenen Zertifikat mitgesandt	entfällt
∟ $containers = \{(container)\}$	Alle Container, auf die der Nutzer zugreifen kann, dazu zählt auch der Container des Standard Tresors	$tresors_{user}$: $[\{tresor\}]$
∟ $accountInfo$	Systemberechtigungen des Nutzers, dazu zählt auch die RMS-Nutzungsberechtigung	$info_{user}$
$Tresor$	Jeder Nutzer besitzt einen Standard Tresor, außerdem wurde je ein Tresor zur Beobachtung der Berechtigungserteilung ohne und mit Microsoft RMS erstellt	ST_{user}, ShT, $RShT$
∟ $name$	Name des Tresors	$name_{tresor}$
∟ $permissions = \{(user, permission)\}$	Menge aller Nutzer mit Zugriffsrecht zum Tresor und deren Berechtigungen; vereinfacht nur zugriffsberechtigte Nutzer	$perm_{tresor}$: $[\{user\}]$, $user_{perm_{tresor}}$
∟ $version$	Versionsnummer der Menge der Berechtigungen; wird bei Änderung inkrementiert	$ver_{perm_{tresor}}$
∟ $members = \{(user, container)\}$	Menge aller Nutzer mit Zugriffsrecht zum Tresor mit der jeweiligen Container-ID zum enthaltenden Container	entfällt
∟ RMS	Attribut das angibt, ob für die Berechtigungen des Tresors Microsoft RMS aktiviert ist	$on_{RMS_{tresor}}$, $off_{RMS_{tresor}}$

Element	Erläuterung	Vereinfachung
⌞ ID	Microsoft RMS ID des Tresors	$ID_{RMS_{tresor}}$
⌞ $permissions = \{(user, rmsPermission)\}$	Menge aller Nutzer mit Zugriffsrecht zum Tresor und deren MS RMS Berechtigungen; vereinfacht nur zugriffsberechtigte Nutzer; kann sich von $perm_{tresor}$ unterscheiden (siehe Abschnitt 4.7)	$perm_{RMS_{tresor}} : [\{user\}]$
⌞ $invitations = \{(invitation)\}$	Menge der Einladungen zur Erteilung von Zugriffsrechten; im Protokoll werden explizit nur aktive Einladungen angefragt, daher gilt dies vereinfacht implizit	$invitations_{tresor} : [\{user\}]$
⌞ $fragment$	Daten, die bei der Erstellung eines Tresors gesetzt werden	$fragment_{tresor}$
$Container$	Speichereinheit, die zur Verwaltung der Daten eines Tresors dient	
⌞ ID	Ein Tresor wird vom Nutzer mit der Container-ID und dem Nutzer-Zertifikat referenziert	ID_{tresor}
⌞ $files = \{(file)\}$	Die Dateien in einem Container werden mit der Nachricht GetContainerChanges() erfragt; in der Antwort sind für den Nutzer individuell alle geänderten Dateien gelistet; in den Sequenzdiagrammen sind daher auch nur die modifizierten Dateien gelistet	$files_{tresor} : [\{file\}]$
⌞ $version$	Versionsnummer der Dateien des Containers; wird bei Dateiänderungen inkrementiert	ver_{tresor}
⌞ $Root\ Dir\ File$	Zentrale Datei, die Informationen zum Container enthält; Inhalt konnte nicht ausgelesen werden, da kein direkter Zugriff möglich war	rdF_{tresor}
⌞ $Group\ Key\ File$	Datei, die Informationen zum Gruppenschlüssel enthält; Inhalt nicht genauer bekannt, da Base64-encodierte Daten auch dekodiert nicht lesbar waren	gkF_{tresor}
$File$	Dateien auf einem Container; mindestens rdF	$file, rdF$
⌞ $name$	Dateiname; wird vor Upload verschlüsselt	$enc(name_{file})$
⌞ $size$	Dateigröße in Byte	$size_{file}$
⌞ $version$	Versionsnummer der Datei; wird bei Dateiänderung inkrementiert	ver_{file}
⌞ $changedBy$	Nutzer, der die Datei als letztes modifiziert hat; wird mit dessen E-Mail Adresse referenziert	$changedBy_{file}$
⌞ $lastAction$	Letzte Aktion die mit der Datei durchgeführt wurde	$lastAction_{file}$, $\text{"}Create\text{"}_{file}$, $\text{"}Update\text{"}_{file}$
⌞ PL	Microsoft RMS Publishing License, enthält Berechtigungsinformationen zur zugehörigen Datei	PL_{file}
$nonce$	Zufallszahl zur Randomisierung einer Funktion	$nonce_{user/server}$

Element	**Erläuterung**	**Vereinfachung**
$invitation$	Einladungen zur Erteilung von Zugriffsrechten; Beispiel: $user1$ lädt $user2$ ein	$user2_{invitation_{user1}}$
∟ $validity$	Gültigkeitsdauer einer Einladung	$val_{user2_{invitation_{user1}}}$
$permission$	Berechtigungsstufe eines Nutzers; nur Manager und Owner konnten beobachtet werden; es existieren außerdem Editor und Reader	entfällt
$authSession$	Session zur Authentifizierung, wenn Nutzer noch kein Zertifikat besitzt	$authSession$
∟ ID	ID der Authentifizierungssession	$ID_{authSession}$
∟ $validity$	Gültigkeitsdauer der Authentifizierungssession	$val_{authSession}$
∟ key	Verschlüsselter Schlüssel für die Authentifizierung	$enc(key_{authSession})$
$session$	Session nach Anmeldung mit Zertifikat	$session$
∟ ID	ID einer Session	$ID_{session}$
∟ $validity$	Gültigkeitsdauer einer Session	$val_{session}$
∟ $token$	Token zur Authentifizierung nach Ablauf einer Session	$token_{session}$
∟ $validity$	Gültigkeitsdauer des Session-Tokens	$val_{token_{session}}$
$tenantHost$, $rmsServerHost$, $adrAddinUrl$	Siehe Abschnitt 4.2	$tenantHost$, $rmsServerHost$, $adrAddinUrl$
$start.pdf$	Durch Tresorit automatisch erstellte Datei im Standard Tresor jedes Nutzers; enthält Hinweise zur Nutzung; Name der Datei ist abhängig von der Systemsprache	$start.pdf$
$docu.docx$	Mit Microsoft Word erstellte Testdatei, während der Durchführung der Aktionen verwendeter Name der Datei war Testfile.docx	$docu.docx$
Storage	Festplatte: Schreib- (W:), Lese- (R:) und Löschzugriffe (D:) werden dargestellt	W:, R:, D:
Registry	Windows-Registrierungsdatenbank: wird zur Kommunikation zwischen Tresorit und Office genutzt, Schreib- (W:) und Lesezugriffe (R:) werden dargestellt	W:, R:
login.tresorit.com, register.tresorit.com	Diese Verwaltungsserver wurden zusammengefasst; an register.tresorit.com wird immer nur eine Anfrage (GetMessages()) gesandt	Tresorit Main
storage2.tresorit.com	Speicherserver für Upload, Download und Berechtigungsmanagement	Tresorit Storage
rmsapi.tresorit.com	Server für die Verwaltung von RMS Berechtigungen und Zugangsdaten	Tresorit RMS
share.tresorit.com	Übertragungsserver von Berechtigungseinladungen	Tresorit Share
sts.aadrm.com	Authentifizierungsserver für Microsoft RMS	Microsoft Auth 1

Element	Erläuterung	Vereinfachung
login.microsoftonline.com	Authentifizierungsserver für Microsoft RMS	Microsoft Auth 2
api.aadrm.com	Server für die Ausstellung der Publishing License	Microsoft RMS

Tabelle 2.8.: Element-Definitionen und -Vereinfachungen zur Protokollarchitektur von Tresorit

Die Indizes wurden genutzt, um die Abhängigkeiten der existierenden Objektklassen zu verdeutlichen. Ein Elternelement befindet sich immer im tiefsten Index, das übertragene Kindelement ist das Wort, das nicht im Index steht. Wenn also der Name der Group Key File des Standard Tresors von Alice übertragen wurde, ist dies im Sequenzdiagramm als $name_{gkF_{ST_A}}$ gekennzeichnet. Würde nur die Group Key File dieses Tresors übertragen werden, wäre es gkF_{ST_A}. Wenn ein Element übertragen wird, sind damit nicht auch alle Kindelemente gemeint. Im erläuterten Fall der Übertragung von gkF_{ST_A} wird also nicht automatisch $name_{gkF_{ST_A}}$ und $ver_{gkF_{ST_A}}$ übertragen, sondern lediglich die Group Key File.

Geschweifte "{ }" und eckige "[ ]" Klammern wurden in der Darstellung wie im JSON-Format verwendet. Ein Block zwischen geschweiften Klammern gibt an, dass die Daten darin zu einem Element gehören. Ein Block zwischen eckigen Klammern gibt an, dass sich darin mehrere gleichartige Elemente befinden oder befinden können.

Die Namen der Dateien waren in den Netzwerkprotokollen verschlüsselt und damit zu keinem Zeitpunkt einsehbar. Sie wurden trotzdem zum Zweck der Übersichtlichkeit angegeben. Dazu wurde angenommen, dass die Übertragungen unmittelbar nach einem Lesezugriff oder vor einem Schreibzugriff der jeweiligen Datei stattfanden. Um die Verschlüsselung zu symbolisieren, sind diese Elemente umgeben von $enc(...)$. Da die genutzten Schlüssel unbekannt waren, sind sie in den Sequenzdiagrammen nicht angegeben.

Neben den Versionsangaben von Dateien und Containern waren ETags[8] Bestandteil der Netzwerknachrichten. Diese Redundanz wurde durch die Beschränkung auf Versionsnummern nicht in die Sequenzdiagramme übernommen. Außerdem wurden leere, und für das Protokoll irrelevante Nachrichten-Elemente nicht angegeben. Die vollständigen Nachrichten sind in Anhang A.1 aufgelistet.

Versionsnummern beziehen sich auf die Änderungen innerhalb eines Diagramms. Ausgangspunkt ist die aktuelle Version auf dem Server, davon abhängig sind vorherige und nachfolgende Nummern angepasst.

[8] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.19

3. Tresorit Protokolle

3.1. Kryptografie

Die folgenden Beschreibungen beruhen auf der genannten Dokumentation. Die Aussagen wurden nicht praktisch verifiziert.

Schlüsselableitung

Zur Schlüsselableitung aus dem gewählten Passwort des Nutzers wird die Funktion PBKDF2[1] unter Verwendung des HMAC[2] mit SHA1[3] genutzt [8], [7]. Mit dieser Funktion wird zunächst ein Authentifizierungsgeheimnis erstellt:

$$authSecret_{user} = PBKDF2^{HMAC\ SHA1}(pwd_{user}, salt_{pwd_{user}}, 10000)$$

$$clientResponse_{user} = PBKDF2(nonce_{user} \,||\, nonce_{server} \,||\, mail_{user}, authSecret_{user}, 1)$$

Dieses Geheimnis wird in den Challenge-Response-Protokollen während Registrierung, Login und Passwortänderung zur Authentifizierung genutzt.

Zur Verschlüsselung des $roaming.profile = rpF_{user}$ wird neben AES-256[4] ebenfalls diese Schlüsselableitungsfunktion unter Verwendung von SHA-512[5] genutzt:

$$masterKey = PBKDF2^{HMAC\ SHA-512}(pwd_{user}, salt_{rpF_{user}}, 20000),$$

$$enc.rpF_{user} = enc^{AES-256}_{masterKey}(plain.roaming.profile),$$

$$rpF_{user} = enc.rpF_{user} \,||\, HMAC^{SHA-512}_{masterKey}(enc.rpF_{user}) \,||\, salt_{rpF_{user}}$$

Dabei sind $salt_{pwd_{user}}$ und $salt_{rpF_{user}}$ zwei unabhängige Werte. Die Konkatenation zweier Werte wird durch $||$ dargestellt.

[1] https://www.ietf.org/rfc/rfc2898.txt
[2] http://www.ietf.org/rfc/rfc2104.txt
[3] http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf
[4] http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
[5] http://tools.ietf.org/html/rfc6234

Das $device.profile = dpF_{device_{user}}$ wird mit der gleichen Funktion verschlüsselt. Der genutzte Schlüssel wird mit einem separaten Salt und statt mit 20000, mit einem Funktionsdurchlauf der PBDKF2 gewonnen.

Der $masterKey$ wird auf dem Betriebssystem für den automatischen Login gespeichert. Hierfür diente vermutlich die Datei $securelogin_{user}$.

Ende-zu-Ende Verschlüsselung

In der ursprünglichen Veröffentlichung des Protokolls *Tresorium* war beschrieben, dass Tresorschlüssel mittels Invitation-oriented Tree-based-Group-Diffie-Hellman berechnet werden [3] [5]. Dieses Konzept widersprach aktuelleren Beschreibungen des Systems [8]:

Dateien und Verzeichnisse werden mit AES-256 im OpenPGP CFB Modus[6] verschlüsselt. Dazu wird bei jeder Änderung einer Datei ein neuer Initialisierungsvektor verwendet. Die Integrität der Chiffrate ist jeweils mit einem HMAC-SHA-256 geschützt.

Der Tresorschlüssel ist für jeden berechtigten Nutzer mit dessen öffentlichen Agreement-Schlüssel (siehe Abschnitt 3.2) verschlüsselt. Alle daraus entstehenden Chiffrate sind in der Group Key File gespeichert. Wenn einem Nutzer die Zugriffsberechtigung entzogen wird, wird ein neuer Tresorschlüssel erzeugt, der dann mit den Agreement-Schlüsseln aller übrigen Nutzer erneut verschlüsselt wird. Von da an veränderte Dateien werden so kryptografisch vor ehemaligen Nutzern geschützt.

Protokollabsicherung

Allen Transaktionen wird eine RSA-2048 SHA-512 Signatur angehängt [6]. Damit wird sowohl die Integrität, als auch die Authentizität einer Transaktion sichergestellt.

3.2. Zertifikate

Alle, während der Analyse beobachteten X.509 Zertifikate[7] wurden im Base64 encodierten DER Format übertragen. Um sie anschaulich darzustellen, wurden sie ins binäre DER Format transformiert.

Die Anfrage bei den Entwicklern von Tresorit bezüglich Korrektheit der Abbildung 3.1 ergab, dass das beobachtete Zertifikat des Test-Nutzers *Alice* von einer nicht mehr genutzten CA ausgestellt wurde. Die aktualisierte Zertifikatsstruktur wurde durch die gestrichelten Linien dargestellt. Die Beschreibung im Folgenden bezieht sich auf die vorliegenden beobachteten Zertifikate.

[6] Nach `https://www.ietf.org/rfc/rfc2440.txt`
[7] `http://tools.ietf.org/html/rfc3280`

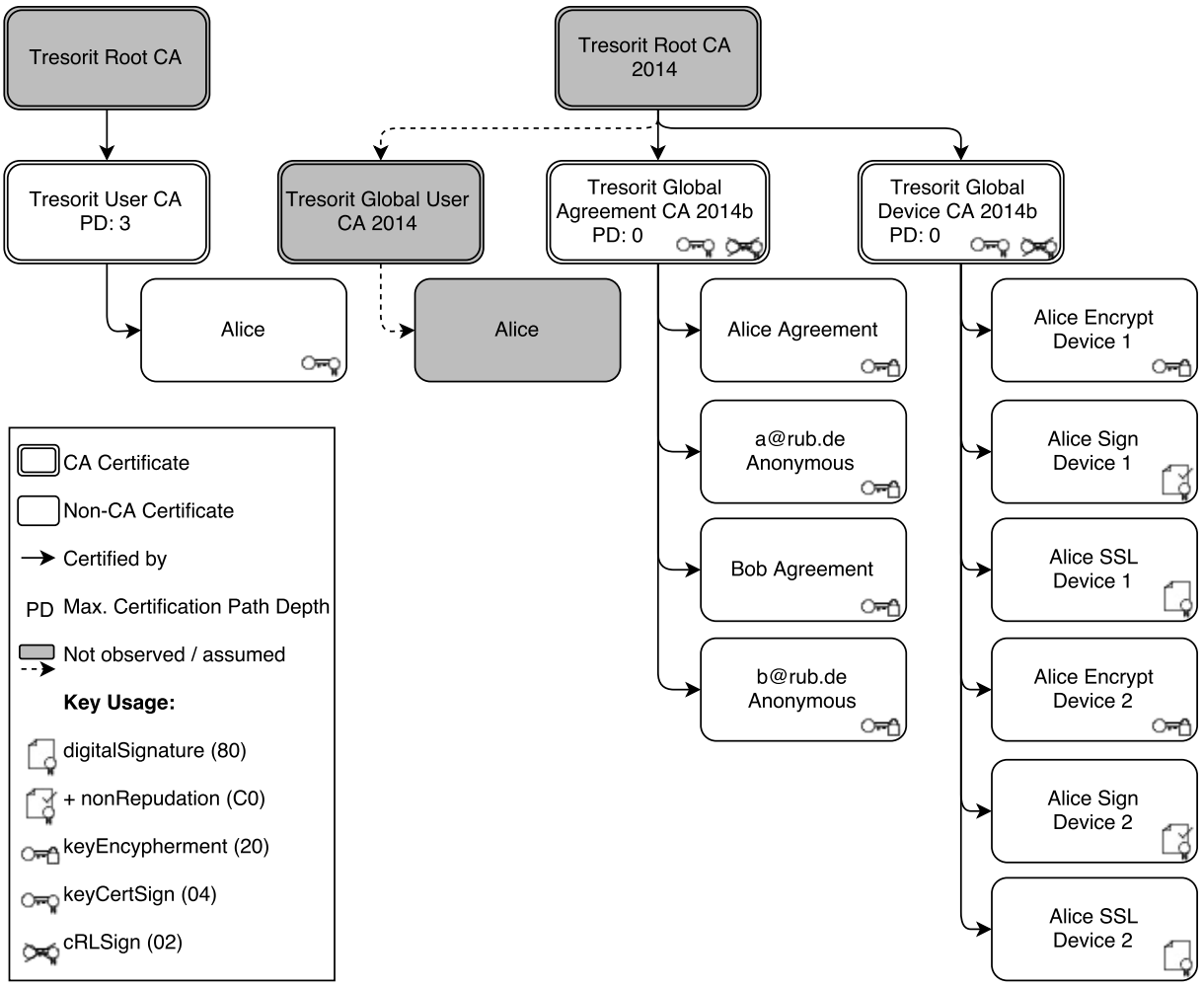

Abbildung 3.1.: Tresorit Zertifikatsstruktur

Die Zertifikate hatten alle eine Gültigkeit von fünf Jahren. Das Nutzerzertifikat *Alice* wurde mit dem Signaturalgorithmus SHA1RSA signiert. Für alle Anderen wurde zur Signatur SHA512RSA verwendet. Alle vorliegenden CA-Zertifikate und die von *Tresorit User CA* und *Tresorit Global Agreement CA 2014b* zertifizierten Zertifikate enthielten 4096 Bit lange RSA Schlüssel. Die RSA Schlüssel der Gerätezertifikate waren 2048 Bit lang.

Die öffentlichen Schlüssel des Anonymous- und des Agreement-Zertifikats eines Nutzers stimmten überein. Im Agreement-Zertifikat waren zusätzlich Vor- und Nachname des Nutzers angegeben. Außerdem waren im Feld *1.3.6.1.4.1.42235.3.6* mehr Informationen eingetragen, die nicht ohne Weiteres lesbar waren. Die Seriennummer und der Fingerabdruck der beiden Zertifikate unterschieden sich ebenfalls. Damit wurde sichergestellt, dass der vollständige Name eines Nutzers bei einer Kontaktaufnahme erst durch dessen Bestätigung an einen anderen Nutzer weitergegeben wurde.

Aufgrund der Mehrfachübertragung der Zertifikate wurden in Abbildung 3.1 nur paarweise verschiedene Zertifikate dargestellt. In der Dokumentation ist beschrieben, dass jeder Nutzer mehrere Agreement-Anonymous-Zertifikatpaare für verschiedene Tresors besitzt. Es lag pro Nutzer aber nur ein solches Zertifikatpaar vor, obwohl mehrere Tresors erstellt wurden [8].

Es ist jeweils der Name und der Zweck des Zertifikats angegeben. Der Zweck wurde mit Hilfe des Attributs Schlüsselverwendung festgestellt. Die Schlüsselverwendung des *Tresorit User CA* Zertifikats war nicht eingeschränkt. Die Zertifizierung ist durch Pfeile markiert.

3.3. Installation

Da die modifizierte Client-Software nicht, wie die frei verfügbare Version, mit einem Setup installiert wurde, wird davon ausgegangen, dass das Protokoll in Diagramm 3.2 von der allgemeinen Installation abweicht. Die Software lag komprimiert in einer Zip-Datei vor und wurde zunächst in das Verzeichnis *C:\Users*Alice*\AppData\Local\Tresorit* entpackt. Die darauf folgenden Aktionen sind in Sequenzdiagramm 3.2 dargestellt.

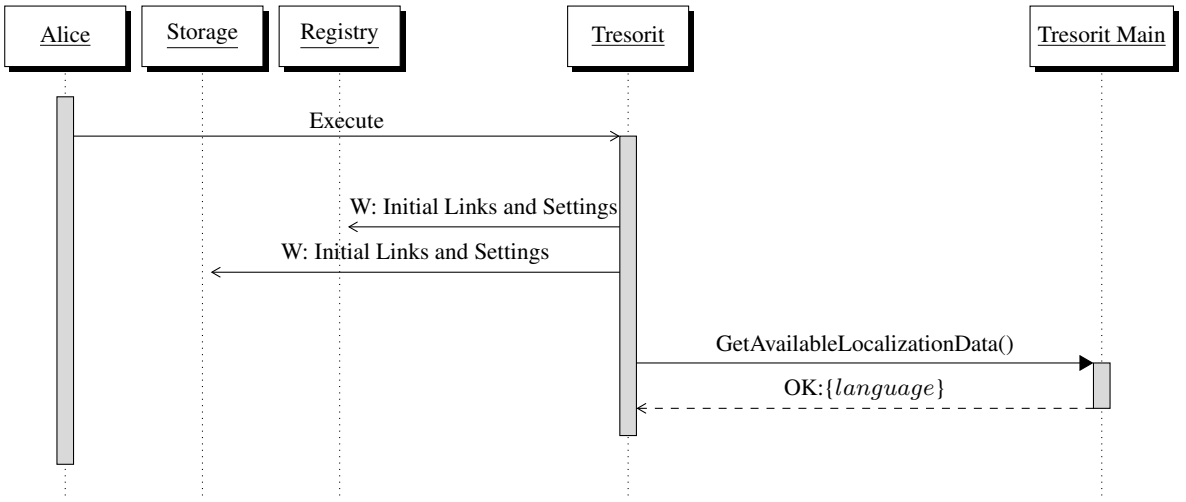

Abbildung 3.2.: Sequenzdiagramm des Installationsprotokolls

3.4. Registrierung

Die Registrierung kann in drei Phasen aufgeteilt werden: Registrierung und Authentisierung bei den Tresorit Servern, Initialisierung des Tresorit Storage Servers und Tresorit RMS Servers und Erstellen des Standard Tresors. Diese drei Phasen verliefen ohne Nutzeraktion nahtlos nacheinander.

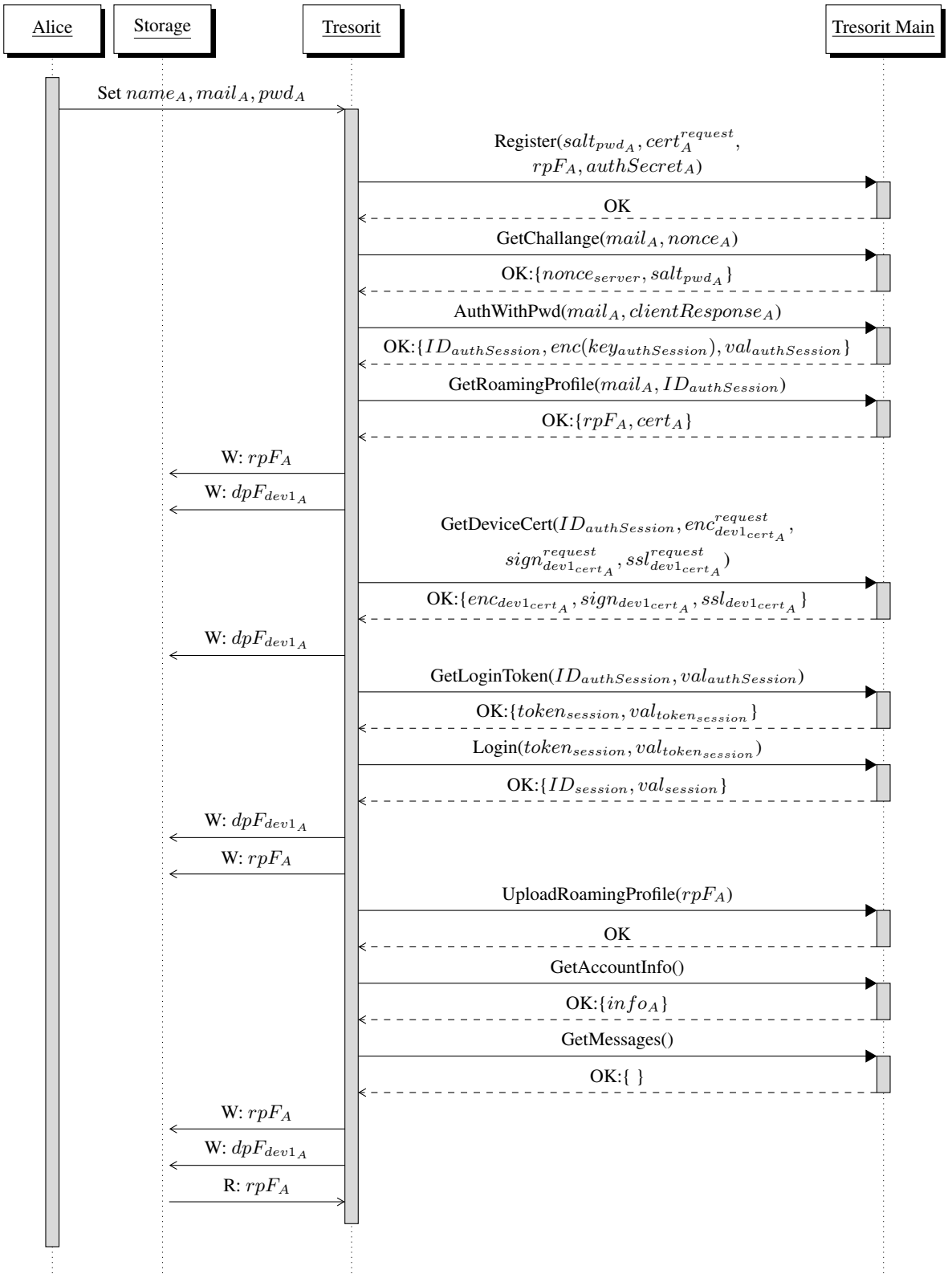

Abbildung 3.3.: Sequenzdiagramm der Registrierungs- und Authentisierungsphase des Registrationsprotokolls

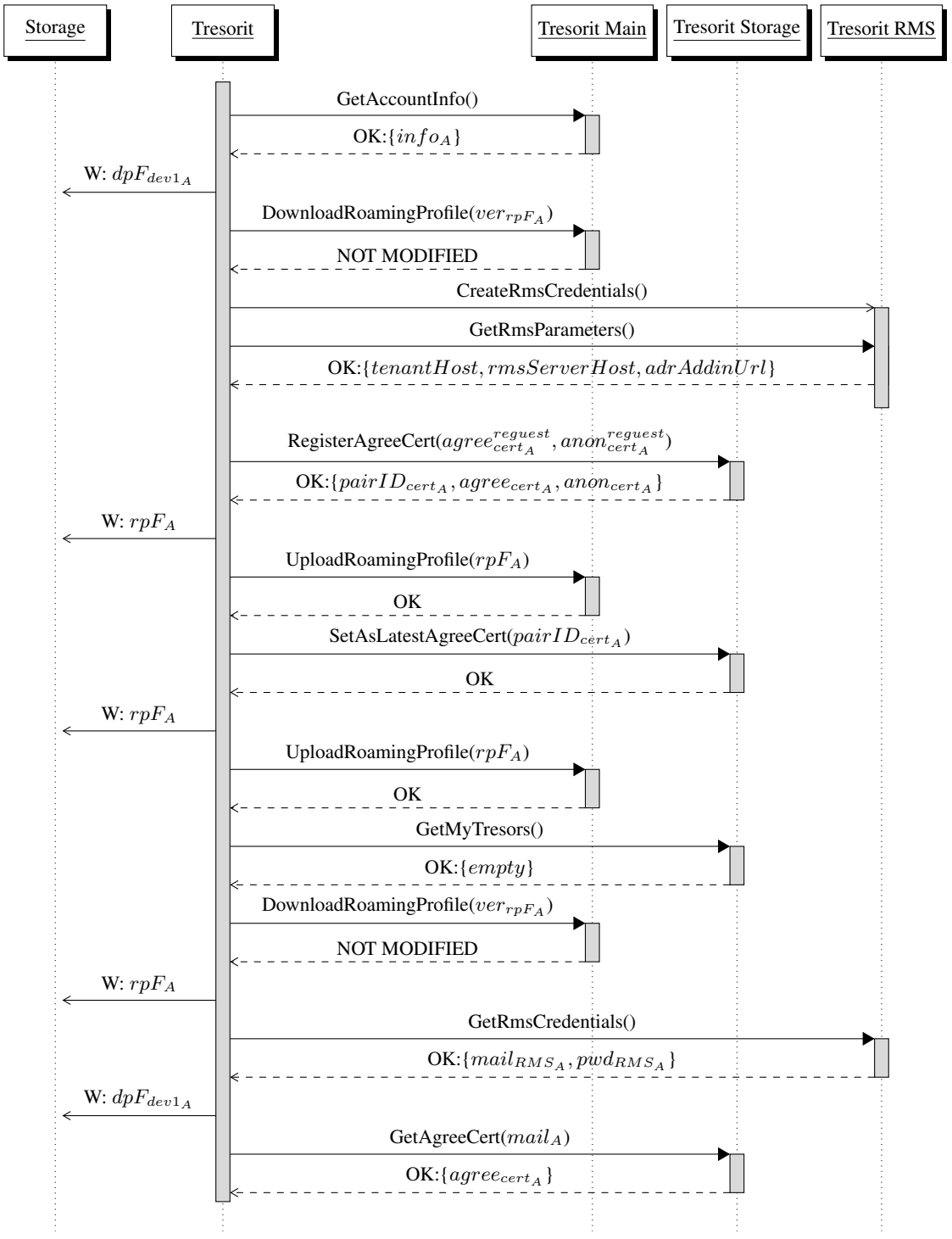

Abbildung 3.4.: Sequenzdiagramm der Tresorit DRM- und Storage-Initialisierungsphase des Registrierungsprotokolls

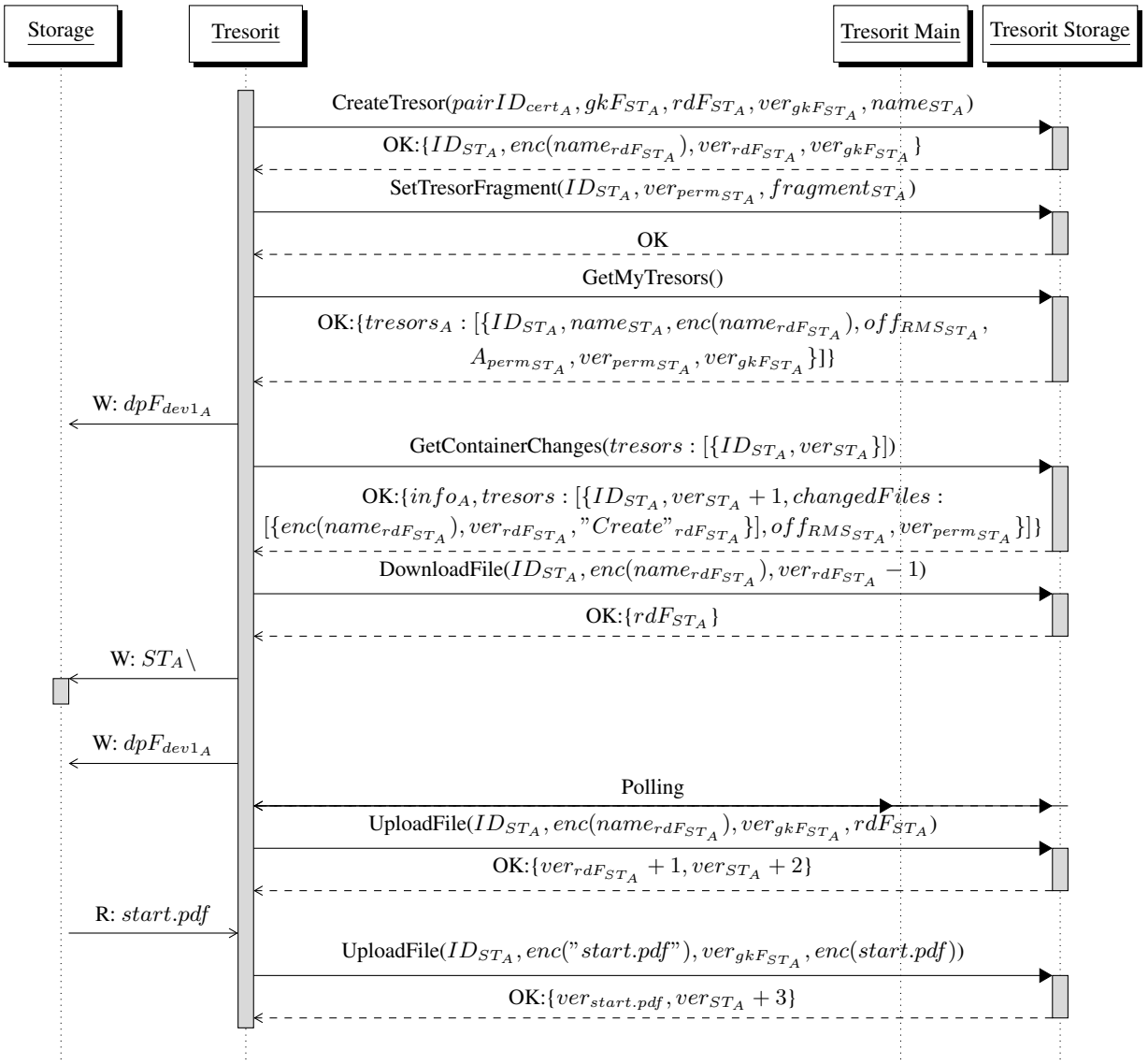

Abbildung 3.5.: Sequenzdiagramm der initialen Tresor-Erstellung während des Registrationsprotokolls

In Phase eins des Registrierungsprotokolls fand die eigentliche Registrierung und Authentifizierung des Nutzers statt. Dazu wurden das Nutzer-Zertifikat und die Geräte-Zertifikate ausgestellt und eine Session aufgebaut. Außerdem wurden weitere Nutzerdaten ausgetauscht.

Im zweiten Schritt wurden Agreement- und Anonymous-Zertifikat zur Kontaktaufnahme mit anderen Nutzern erstellt und festgelegt. Zusätzlich wurden hier, ohne dass der Nutzer die Berechtigung zur Tresorit DRM Nutzung hatte, Microsoft RMS Parameter und die Credentials zur Anmeldung an Microsoft RMS Servern erstellt und an den Nutzer gesendet.

Letztlich wurde der Standard-Tresor des Nutzers erstellt. Dieser enthielt nach der Registrierung eine Ge-

brauchsanleitung, die, in Abhängigkeit von der genutzten Sprache, einen anderen Namen und anderen Inhalt hatte. Diese Datei wurde, den Beobachtungen zufolge, mit der Installationsdatei auf den Client geladen, bei der Registrierung lokal in den Tresor gespeichert und dann synchronisiert.

Das in der Nachricht SetTresorFragment() übermittelte $fragment_{ST_A}$ war ein 936 Zeichen langer Base64-String. Die Base64-dekodierten Daten waren ohne Weiteres nicht lesbar.

3.5. Passwortänderung

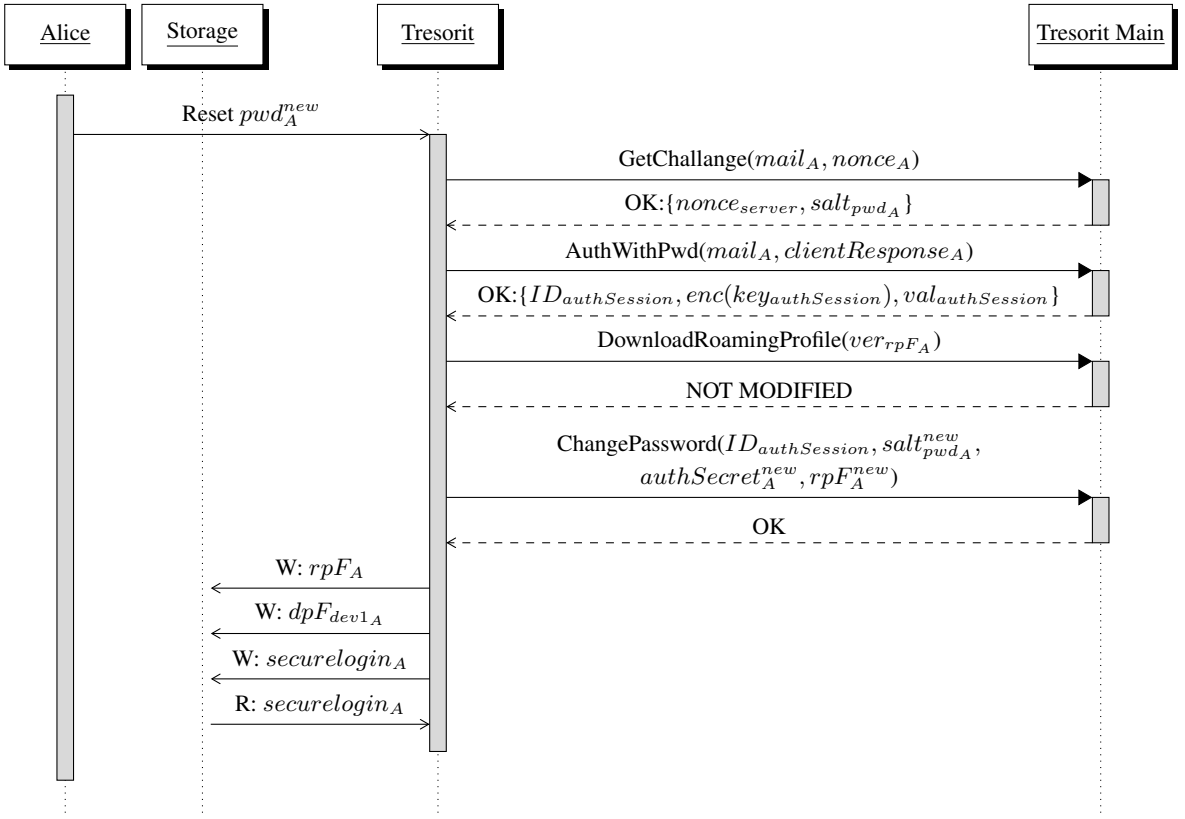

Abbildung 3.6.: Sequenzdiagramm der Passwortänderung

Zum Aufbau der Authentifizierungssession wurde der Salt des alten Passworts benötigt. Dieser wurde danach mit der Übertragung des neuen Passworts geändert. Außerdem wurde das $roaming.profile$ wiederverschlüsselt mitgesendet, da der Schlüssel vom neuen Passwort abgeleitet wurde.

3.6. Login auf einem weiterem Gerät

Das Anmelden des Nutzers auf einem weiteren Gerät erfolgte zunächst in zwei Phasen: Authentifizierung des Nutzers mit der Erstellung der Geräte-Zertifikate und anschließend die Initialisierung mit der Übertra-

gung von Nutzerinformationen, Zertifikaten und Informationen zu Tresors, auf die der Nutzer Zugriff hatte.

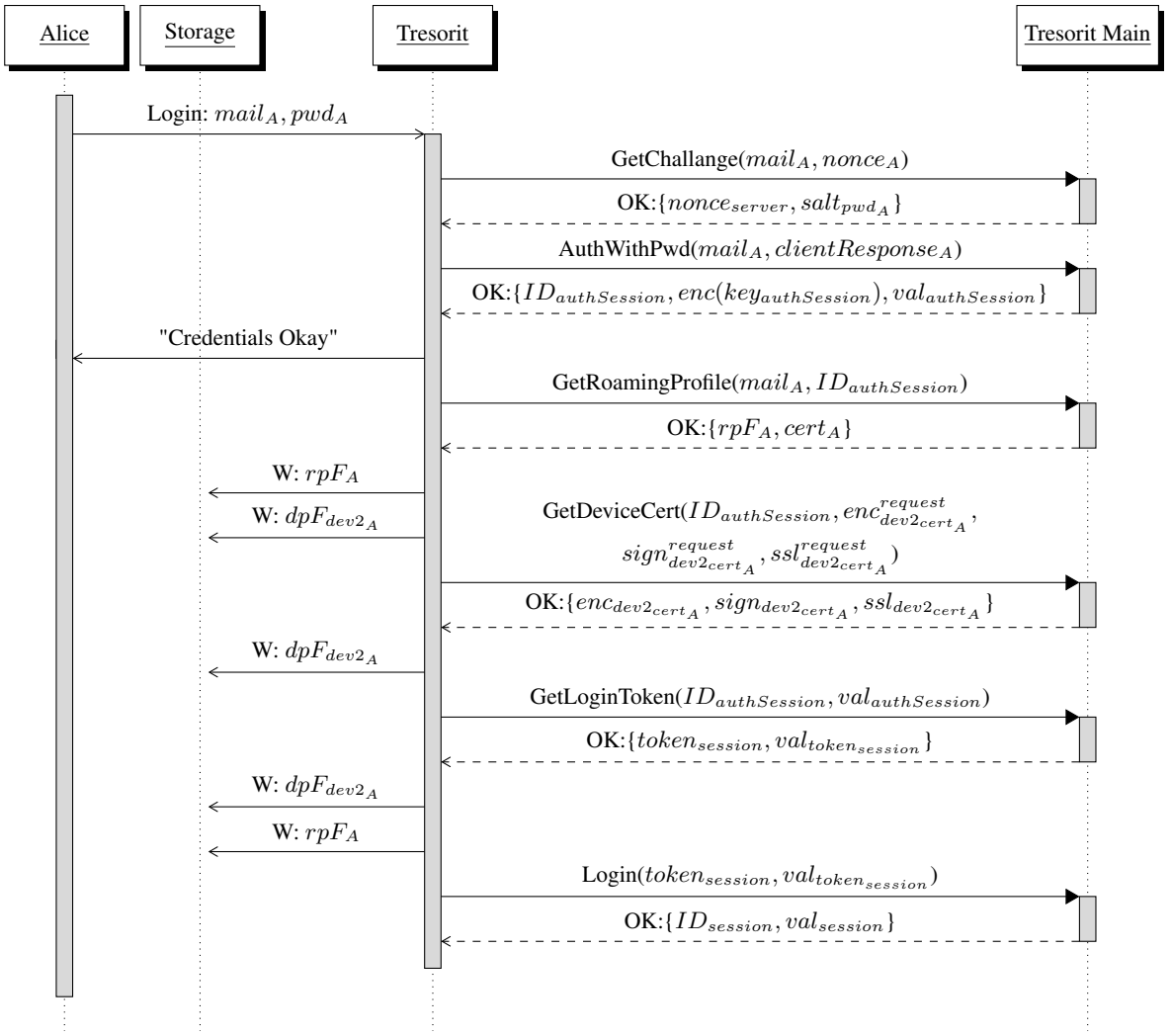

Abbildung 3.7.: Sequenzdiagramm der Authentifizierung während des Logins auf einem weiterem Gerät

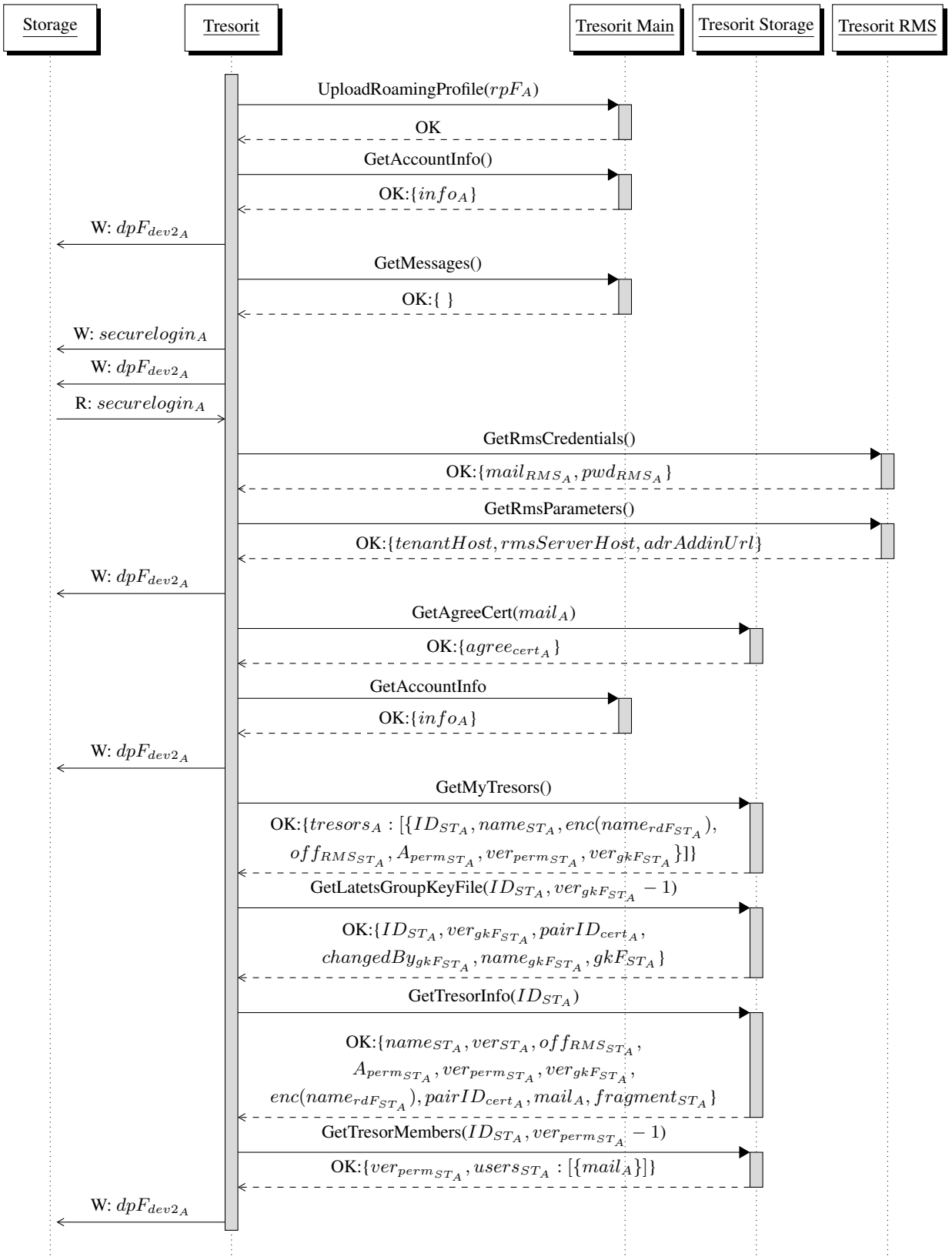

Abbildung 3.8.: Sequenzdiagramm der Initialisierung während des Logins auf einem weiterem Gerät

Das in der Antwort zu GetTresorInfo() enthaltene $fragment_{ST_A}$ war der gleiche Base64-String, der während der Registrierung mittels SetTresorFragment() an den Server übertragen wurde.

In der gleichen Antwort wurde eine Version der Group Key File und der pairID angegeben. Zumal sich die $pairID_{cert_{user}}$ und deren Version bezüglich eines Tresors beim Erteilen und Entziehen von Berechtigungen änderte (siehe 3.13) und dies auch bei der Group Key File der Fall war, konnte diese Redundanz nicht nachvollzogen werden. Die Version der $pairID_{cert_{user}}$ wurde daher nicht im Diagramm 3.8 angegeben.

Das Synchronisieren des Standard-Tresors des Nutzers mit dem lokalen Speicher des weiteren Geräts war nicht automatisch aktiviert. Hierfür musste eine Nutzerinteraktion stattfinden (siehe Abschnitt 3.8).

3.7. Regelmäßiges Polling

Die Regelmäßigkeit der Nachrichten wurde anhand der Häufigkeit der Übertragung unabhängig von Nutzeraktionen festgestellt. Eine feste zeitliche Periode der Anfragen wurde nicht beobachtet. Eine weitere Überprüfung der Übertragungen, während keine Aktionen durchgeführt wurden, bestätigte die Annahme des aktionsunabhängigen Pollings. Aufgrund der zeitlichen Nähe zu anderen aktionsabhängigen Anfragen während einer Aktion, wurde davon ausgegangen, dass zusätzlich zur regelmäßigen Abfrage auch aktionsabhängige Anfragen, der in Diagramm 3.9 aufgeführten Nachrichten, stattfanden.

Die Änderungen von Containern und der Download der Root Dir File(s) wurden zu allen Containern, zu denen der Nutzers Zugriff hatte, angefragt.

In der Antwort zur Anfrage GetMessages() konnte zu keinem Zeitpunkt eine Nachricht beobachtet werden. Zumal die Übertragungsgröße der Antwort in Process Monitor, verglichen mit anderen Nachrichten, lang war, wurde angenommen, dass die Nachrichten nicht mitgeloggt wurden.

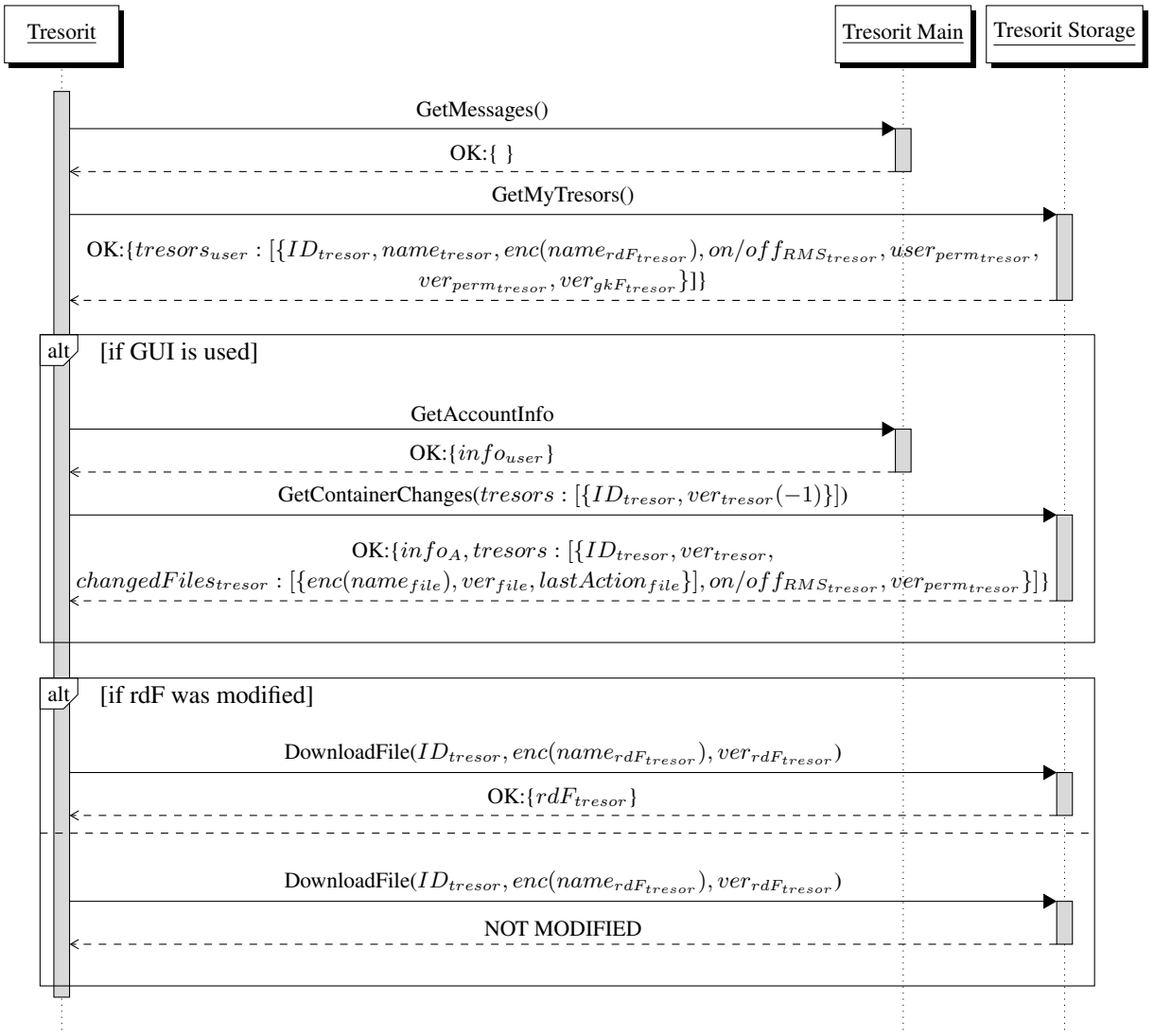

Abbildung 3.9.: Sequenzdiagramm der Nachrichten, die regelmäßig und unabhängig von Aktionen des Nutzers gesendet wurden (Polling)

3.8. Synchronisation auf einem weiterem Gerät

Wenn ein Nutzer den Zugriff auf einen zuvor unbekannten Tresor erhielt, wurden die Daten nicht automatisch mit dem lokalen Speicher synchronisiert. Zur Synchronisation war die Zustimmung des Nutzers erforderlich. In Sequenzdiagramm 3.10 ist die beobachtete Synchronisation nach dem Login auf einem zweiten Gerät des Nutzers dargestellt. Zum Zeitpunkt der Synchronisation waren keine Änderungen am Standardtresor vorgenommen worden.

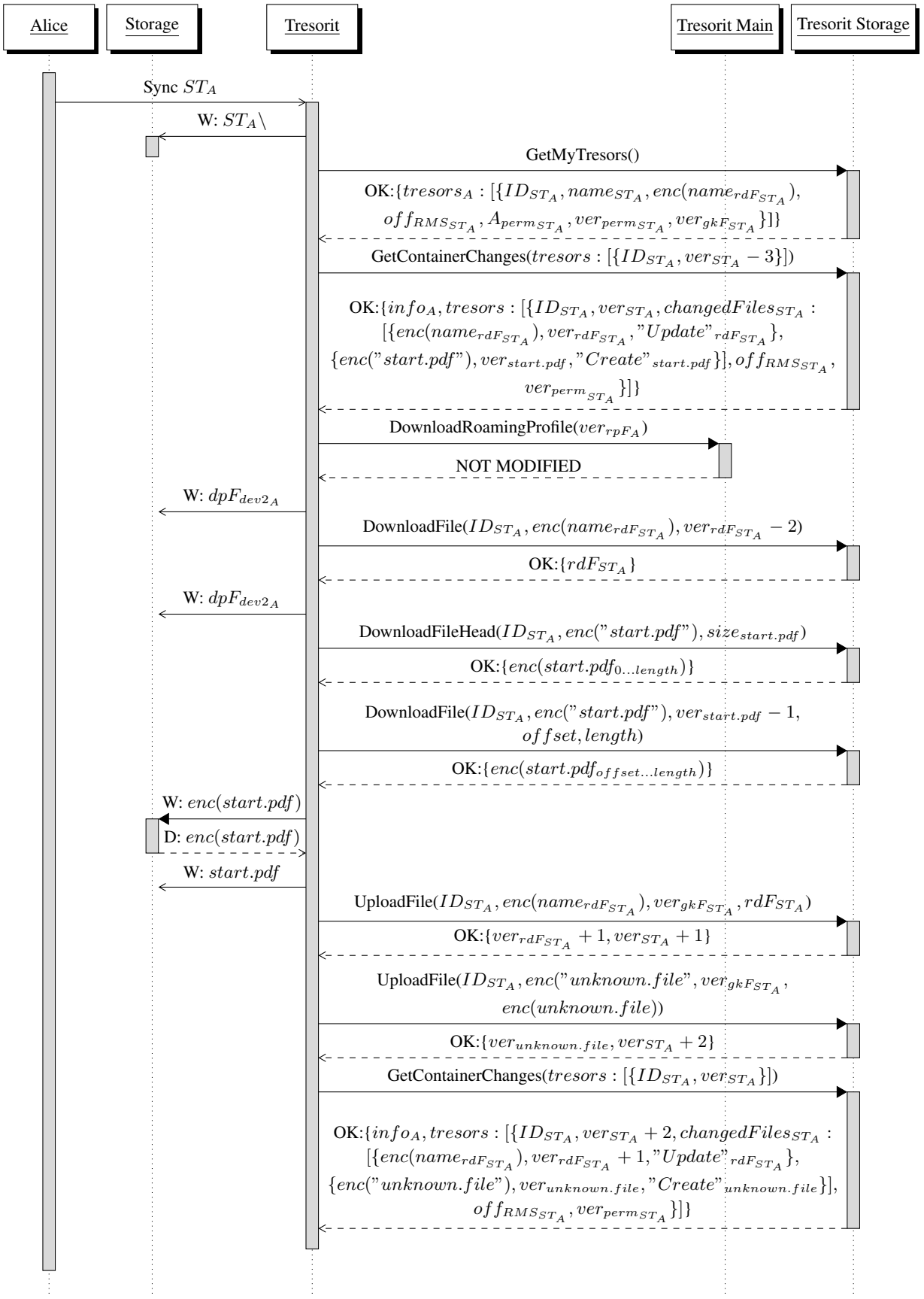

Abbildung 3.10.: Sequenzdiagramm der Synchronisation auf einem weiterem Gerät

Die unbekannte Datei, die am Schluss der Synchronisierung auf den Server geladen wurde, konnte keiner lokalen Datei zugeordnet werden. Die Größe der Datei betrug laut den beobachteten Nachrichten 3 Byte.

3.9. Upload neuer Daten

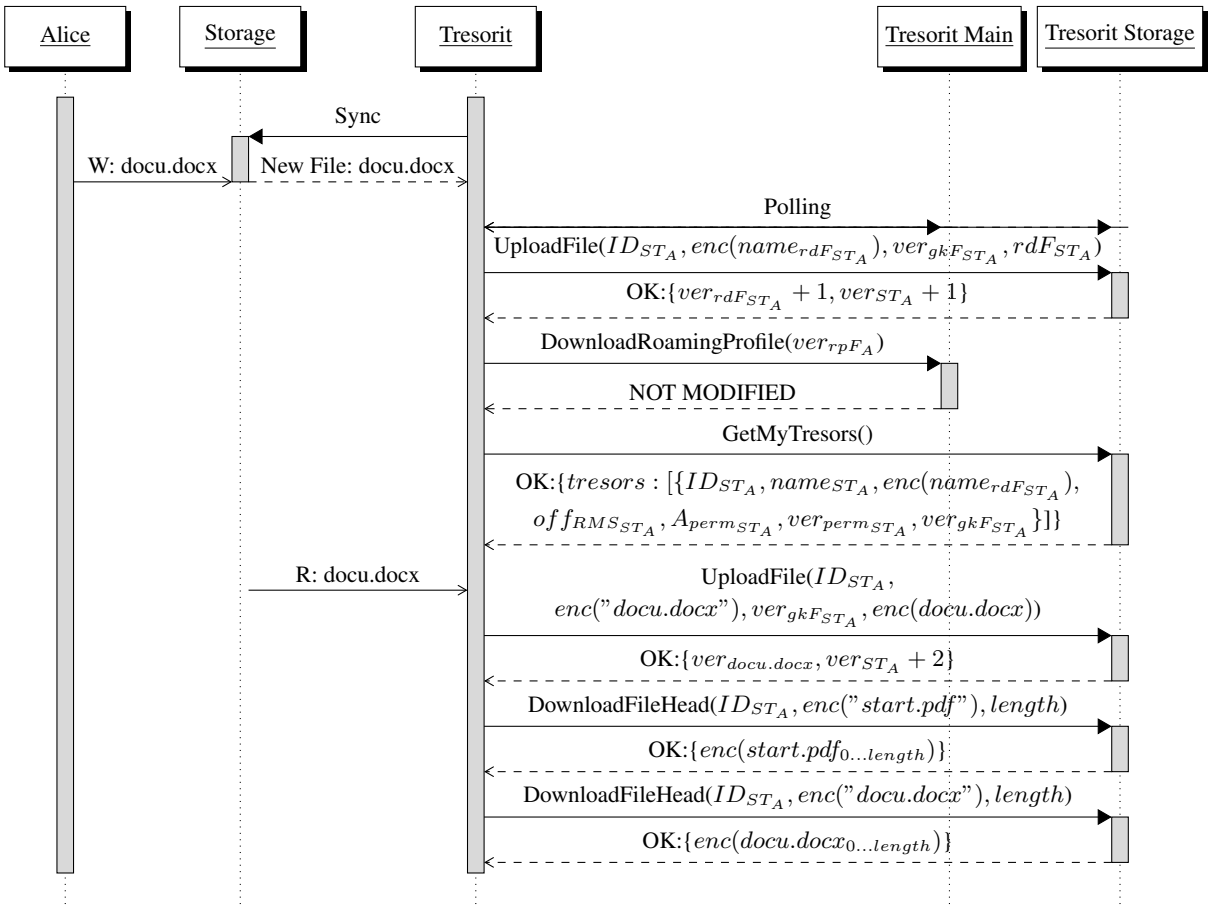

Abbildung 3.11.: Sequenzdiagramm des Uploads

Das Hinzufügen der Datei, welches in Diagramm 3.11 dargestellt ist, wurde durchgeführt, nachdem der Login und das Synchronisieren des Standardtresors auf einem zweiten Gerät des Nutzers *Alice* erfolgte.

3.10. Download neuer Daten

Der Download erfolgte nach dem in Abschnitt 3.9 beschriebenen Upload. Dabei liefen die Geräte des Nutzers nicht gleichzeitig, sodass keine direkte Kommunikation stattfinden konnte.

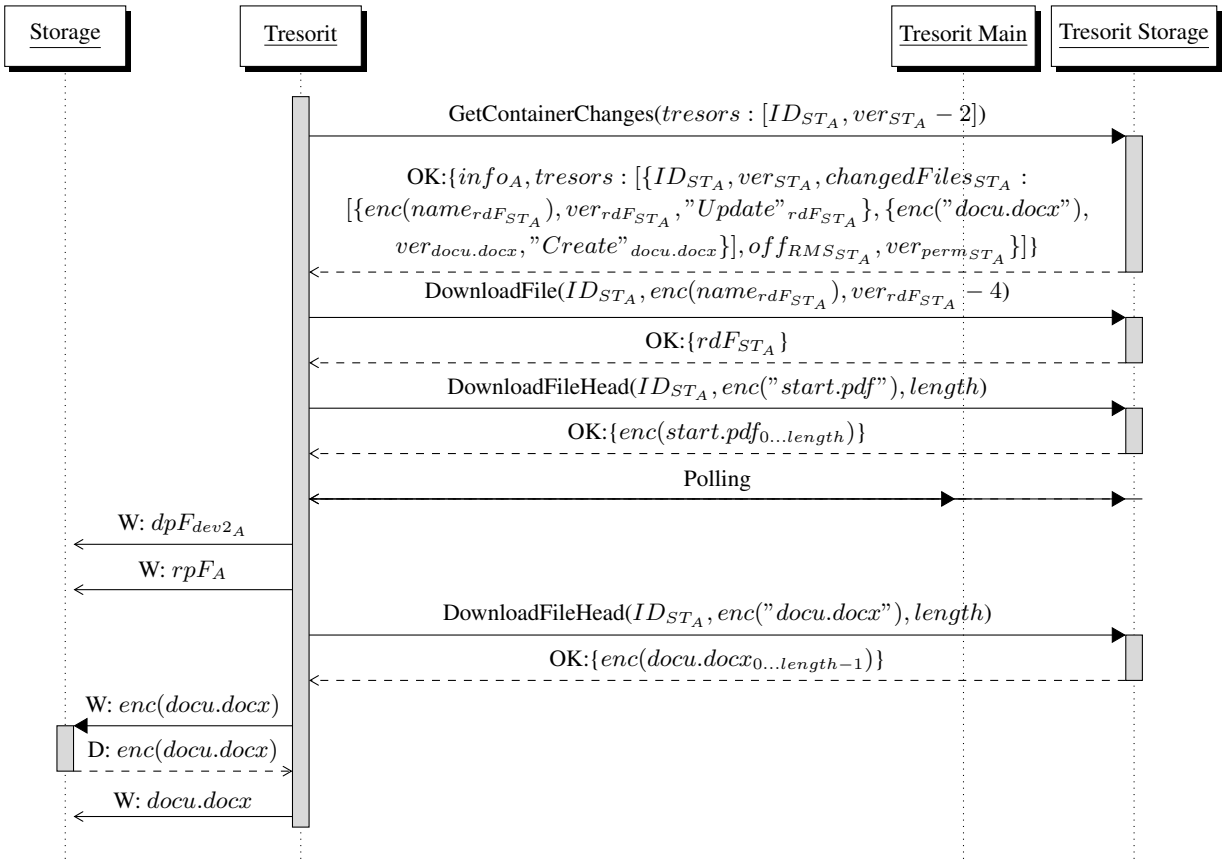

Abbildung 3.12.: Sequenzdiagramm des Downloads

Da die im Up- und Download übertragenen Dateien nicht in die Log-Daten der modifizierten Client-Version geschrieben wurden, wurde bei einem mehrfachen Anfragen von $DownloadFileHead()$ angenommen, dass es sich um einen Übertragungsfehler handelte. Diese Annahme wurde durch die Tatsache gestützt, dass die Parameter der Anfrage unverändert blieben. Ihr widerspricht, dass die Anfrage innerhalb einer Sekunde zweimal hintereinander gestellt wurde, ohne dass beim Client eine Antwort angekommen war. Ein Timeout als Grund für die Mehrfachübertragung wird aufgrund der kurzen Zeit nicht angenommen.

Die zuvor hochgeladene Datei $docu.docx$ wurde, wie in Abbildung 3.12 beschrieben, auf dem zweiten Gerät heruntergeladen.

3.11. Zugriffsberechtigung Erteilen (Share)

Dem Nutzer *Bob* wurden Berechtigungen für den Standardtresor von Nutzer *Alice* erteilt. Da, verglichen zu anderen Tresors, keine Besonderheiten des Standardtresors beobachtet werden konnten, soll das Erteilen und Entziehen von Berechtigungen mit einem allgemeinen Tresor betrachtet werden. Daher wurde der Standardtresor von *Alice* ST_A in den folgenden Diagrammen ShT genannt.

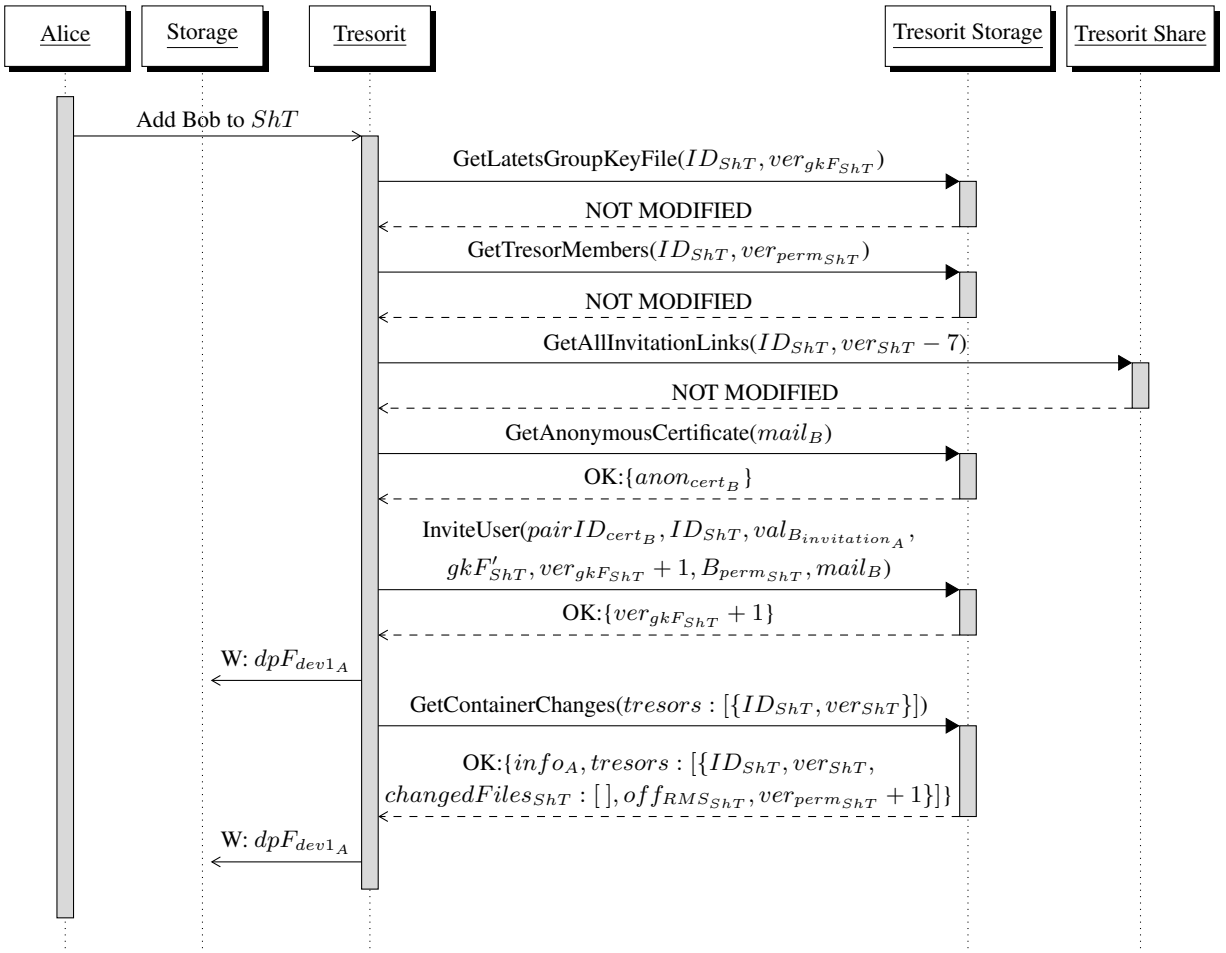

Abbildung 3.13.: Sequenzdiagramm der Erteilung von Berechtigungen

Die ersten drei Nachrichten an den Storage Server wurden in der realen Beobachtung, wie in Sequenzdiagramm 3.13 abgebildet, angefragt, ohne dass eine Änderung der Daten auf dem Server stattfand. Diese Darstellung bildet damit nicht die Allgemeinheit ab.

In der Nachricht InviteUser() wurde dem Storage Server mitgeteilt, welcher Nutzer Zugriff auf den Tresor erhält. Gleichzeitig wurde die Group Key File übertragen, damit der Nutzer *Bob* und alle anderen berechtigten Nutzer die Ende-zu-Ende verschlüsselten Daten entschlüsseln konnten.

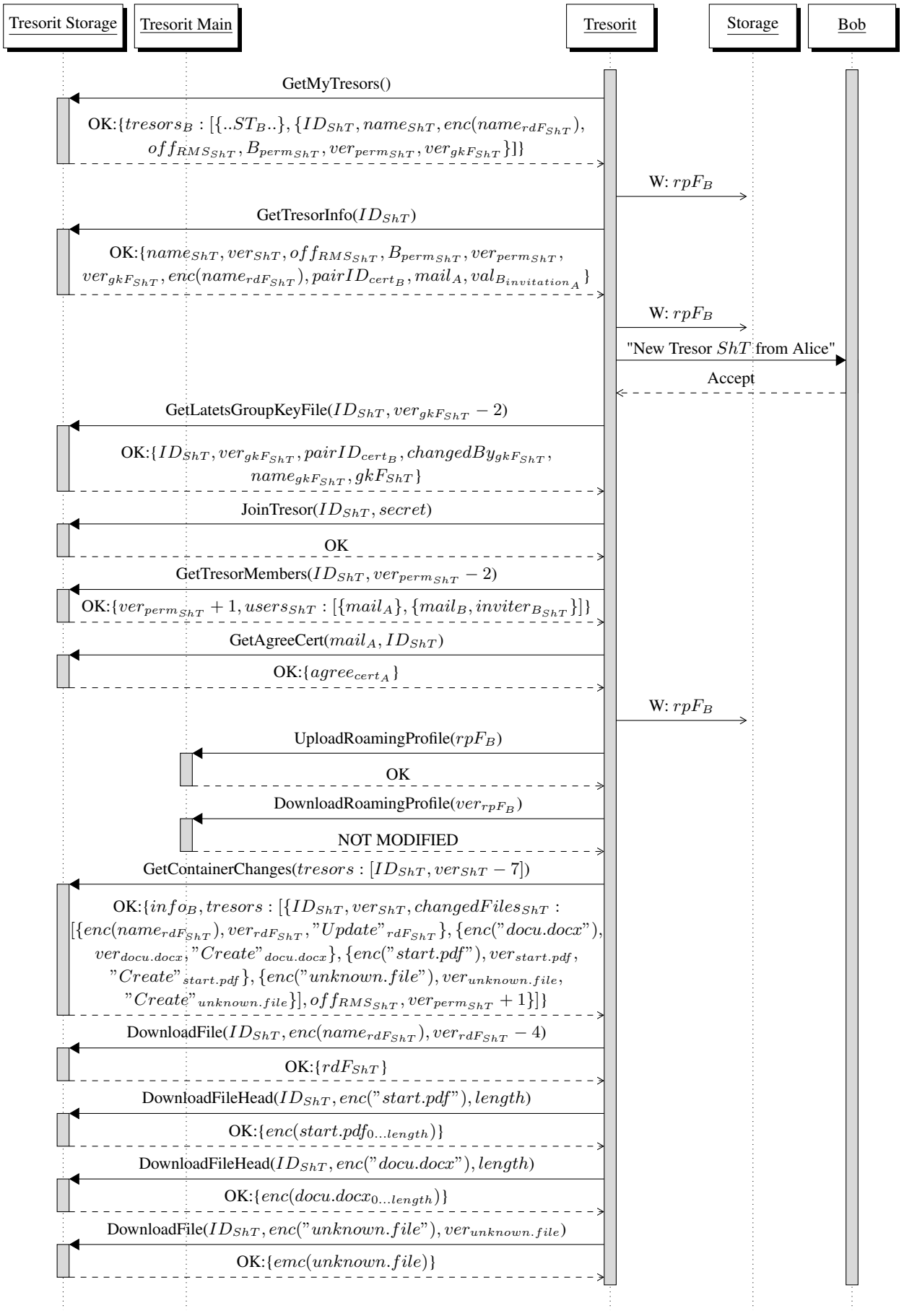

Abbildung 3.14.: Sequenzdiagramm vom Protokoll des Erlangens von Berechtigungen

Da die Informationen zu Bobs Standardtresor für das Empfangen von ShT nicht relevant waren, wurden sie im Sequenzdiagramm 3.14 abgekürzt.

Das in Nachricht JoinTresor() übermittelte Geheimnis $secret$ konnte auf keine Quelle zurückgeführt werden. Da kein weiterer Kanal zwischen *Bob* und *Alice* bzw. den Tresorit Servern hergestellt war und die Maschinen der beiden Nutzer zu keinem Zeitpunkt gleichzeitig liefen, wurde die Group Key File als einzig mögliches Trägermedium vermutet.

Der ursprünglichen Syntax der Nachricht GetTresorMembers() wurde entnommen, dass ein Tresor auf mehreren Containern gespeichert sein kann (siehe Abschnitt 2.4.3). Die Container der beiden Nutzer unterschieden sich in der Beobachtung nicht.

Mit der Anfrage GetContainerChanges() wurden alle Dateien, die sich zu dem Zeitpunkt im Tresor befanden, abgefragt. Da der Tresor ShT keine Verzeichnisse enthielt, konnte keine Aussage zur Behandlung von Verzeichnissen in Tresorit getroffen werden.

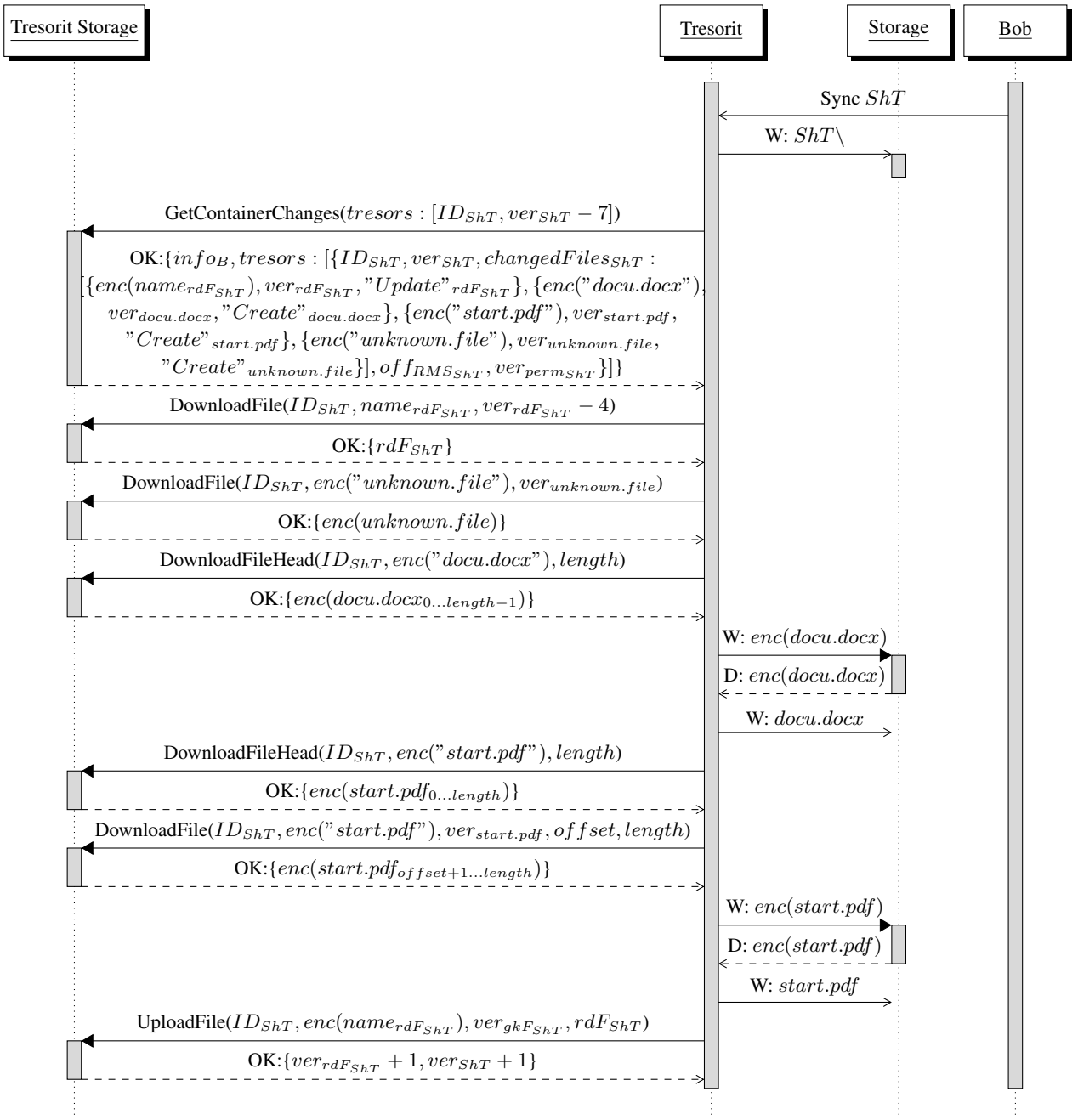

Abbildung 3.15.: Sequenzdiagramm der Synchronisierung der Daten des Shared Tresors

Nachdem die Informationen zu Tresor ShT abgefragt waren, konnte der Nutzer die Synchronisierung dieses Tresors aktivieren. Dazu lief ein ähnliches Protokoll wie in Sequenzdiagramm 3.10 ab. Im Gegensatz zur Synchronisation des eigenen Tresors, wurde nach dem Erhalt von Zugriffsberechtigungen keine neue Datei auf den Storage Server geladen, wie in Sequenzdiagramm 3.15 zu sehen ist.

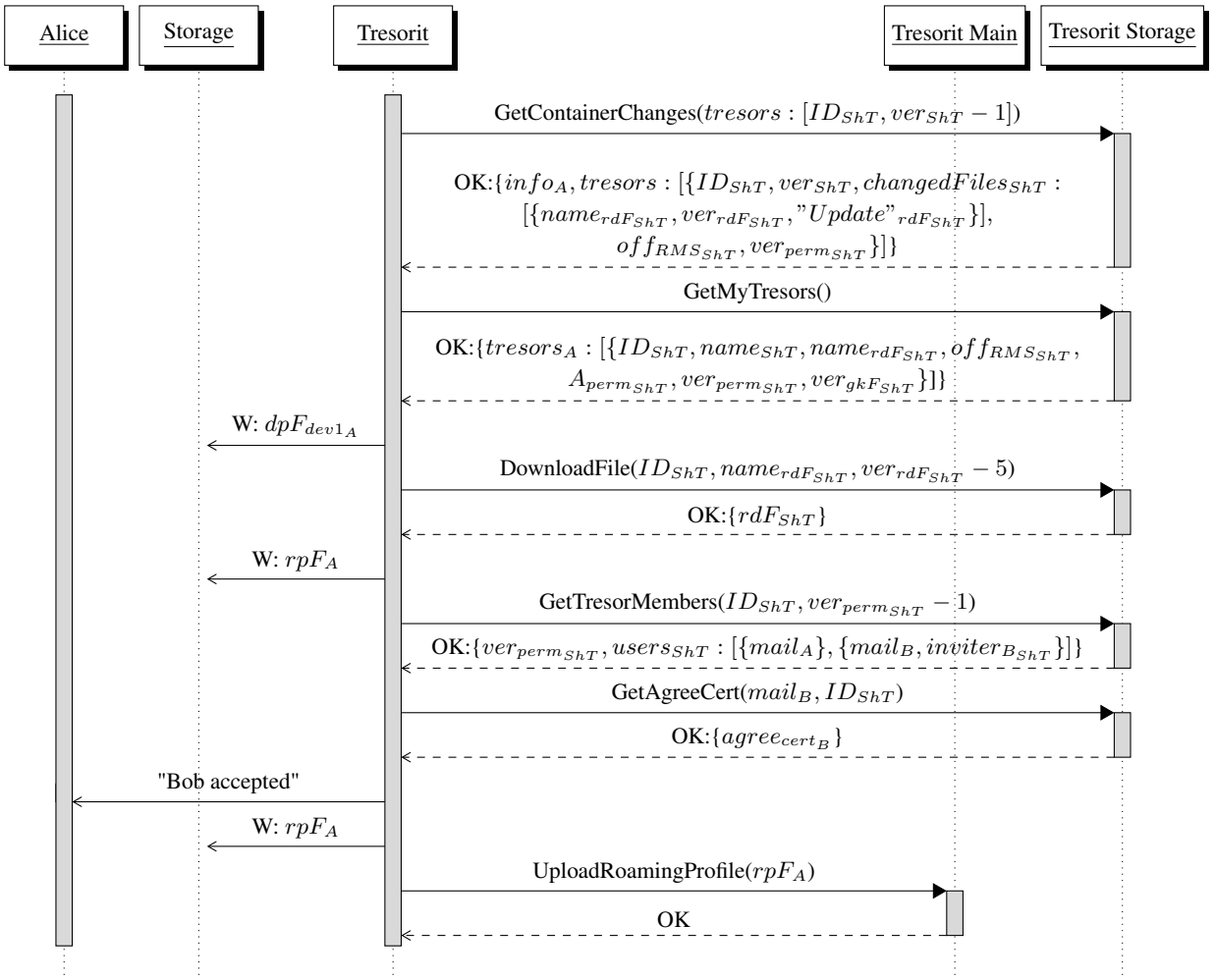

Abbildung 3.16.: Sequenzdiagramm der Empfangsbestätigung des Shared Tresors

Nachdem der Nutzer *Bob* die Einladung zum Zugriff auf den Tresor von *Alice* angenommen hatte, fragte die Client-Software von *Alice* im Polling die geänderten Dateien der Tresors ab. Da die Root Dir File des Tresors von *Bobs* Client bei der Annahme des Tresors geändert wurde, konnte die Client-Software von *Alice* diese Annahme erkennen. Daraufhin wurden unter anderem die zugriffsberechtigten Nutzer abgefragt, zu denen dann *Bob* gehörte. Nachdem die Erteilung der Berechtigung nur mit Kenntnis des Anonymous-Zertifikats von *Bob* erfolgte, war *Alice* nach der Annahme berechtigt das Agreement-Zertifikat zu erhalten.

3.12. Zugriffsberechtigung Entziehen (Revoke)

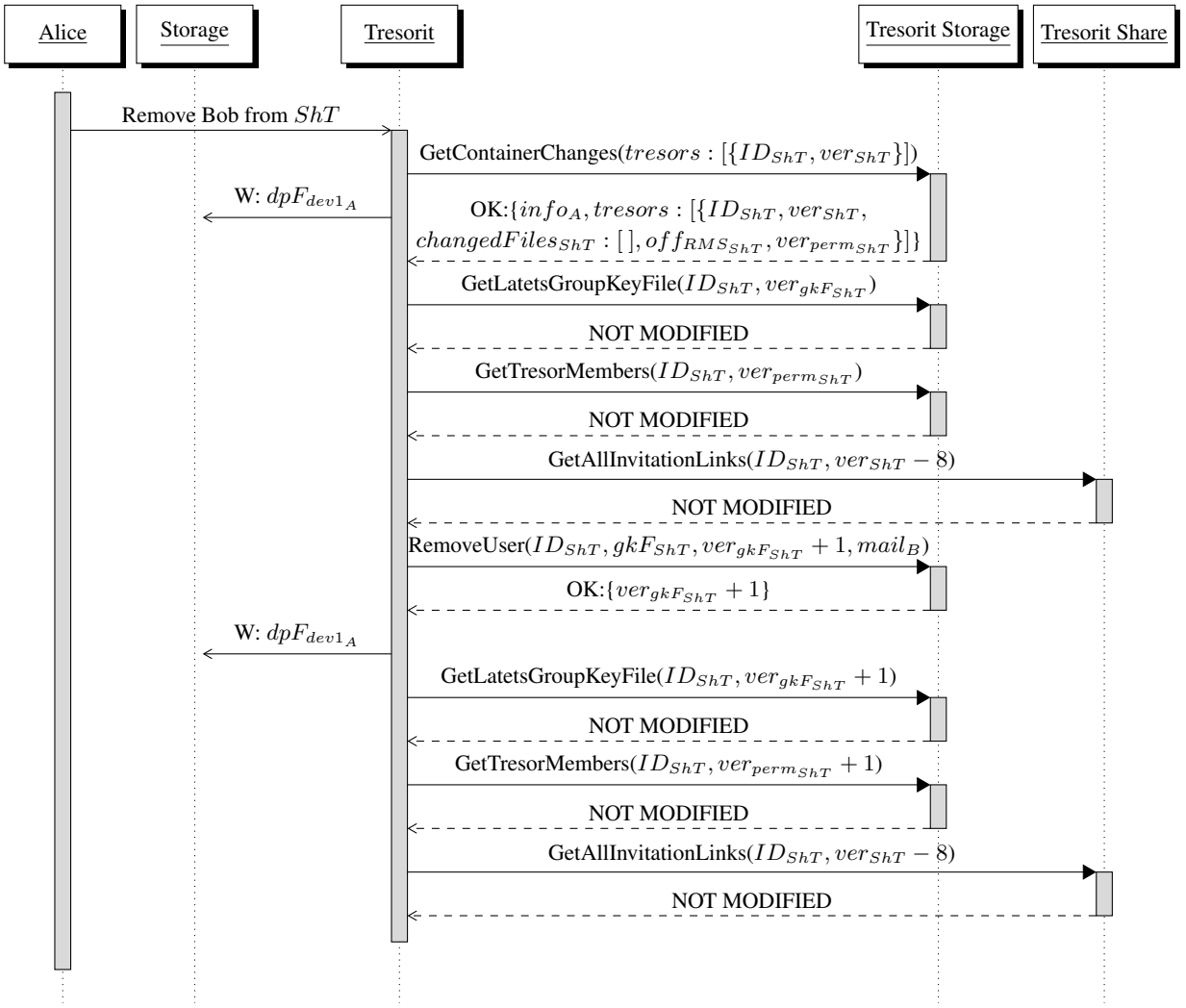

Abbildung 3.17.: Sequenzdiagramm der Entziehung von Zugangsrechten

Zur Absicherung der Entziehung von Berechtigungen gegen das Blockieren von Nachrichten bzw. zur Bestätigung des korrekten Protokollablaufs wurde der Status der Berechtigungen nach der Anfrage RemoveUser() abgefragt. Diese Abfrage wurde mit dem HTTP-Status-Code 304 beantwortet, wodurch der Client-Software bestätigt wurde, dass dem Nutzer *Bob* erfolgreich die Zugangsberechtigung entzogen wurde.

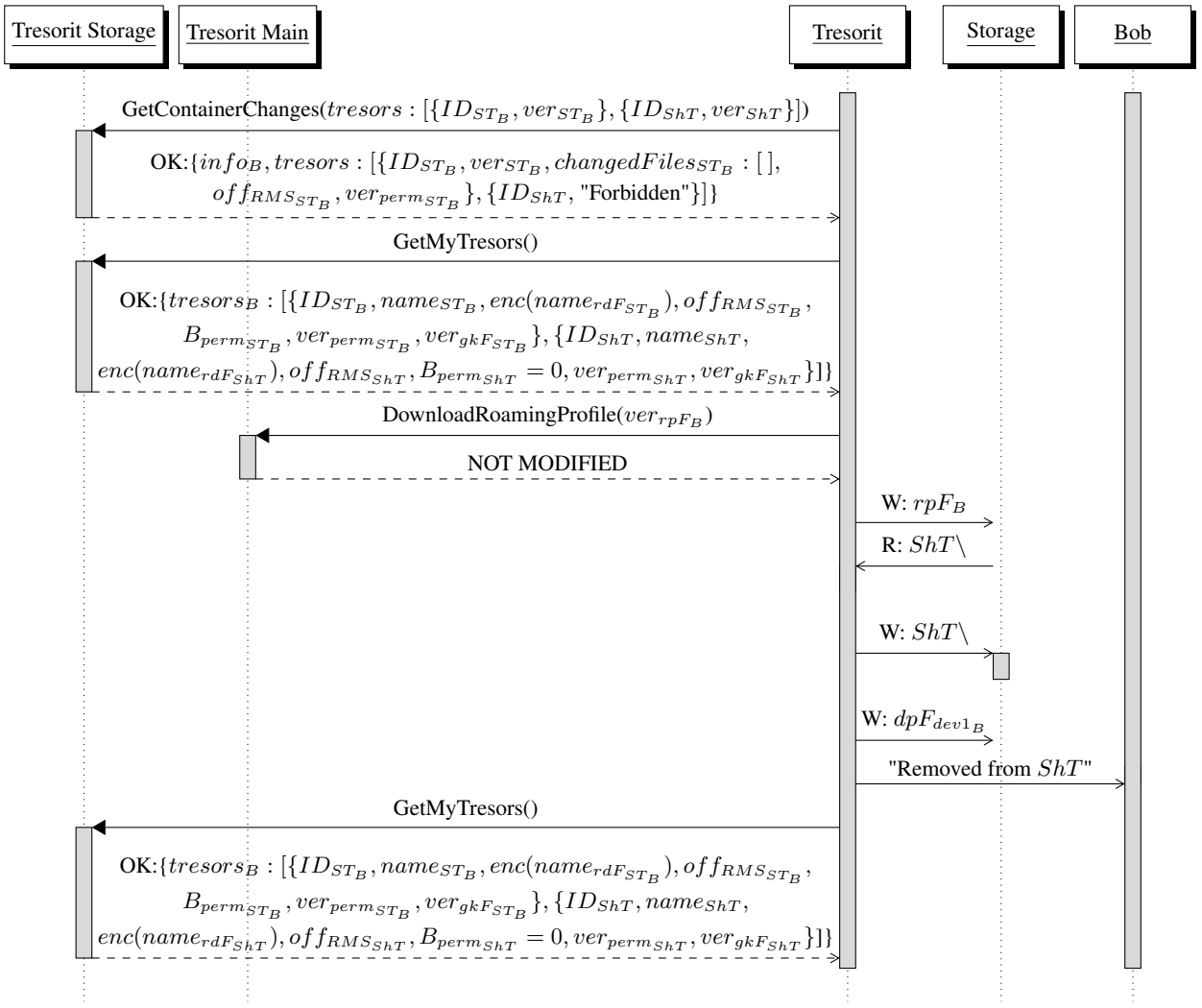

Abbildung 3.18.: Sequenzdiagramm des Empfangs der Entziehung von Zugangsrechten

Nachdem die Client-Software von *Bob* in der Antwort auf die Anfrage GetContainerChanges() im Rahmen des Pollings keine Informationen zu Tresor ShT erhielt, wurden die Tresors von *Bob* abgefragt. In der Antwort dieser zweiten Anfrage war die aktualisierte Berechtigungskategorie von *Bob* zu Tresor ShT aufgeführt. Daraus konnte geschlossen werden, dass *Bob* die Berechtigungen zu diesem Tresor entzogen wurden.

4. Tresorit DRM Integration

4.1. Einführung anhand Dokumentation

Tresorit bot, wie bereits beschrieben, vier Berechtigungskategorien: Owner, Manager, Editor, Reader. Die Berechtigungen bestanden ohne Tresorit DRM aus den Rechten Lese-, Schreib- und Löschzugriffen durchzuführen, sowie dem Recht, Berechtigungen weiterzugeben. Wie in Abbildung 4.1 dargestellt ist, wurden die vier Kategorien durch Tresorit DRM auf dateispezifische Berechtigungen erweitert.

Abbildung 4.1.: Berechtigungskategorien in Tresorit DRM [4]

Diese Erweiterung wurde mit Hilfe von Microsoft RMS realisiert. Die Berechtigungen von Microsoft RMS bezogen sich auf E-Mail Adressen der berechtigten Nutzer. Anstatt diese E-Mail Adressen der berechtigten Nutzer direkt anzugeben, wurden von Tresorit drei Gruppen E-Mail Adressen, für Managers, Editors und Readers erstellt. In diesen E-Mail Gruppen waren dann die tatsächlichen E-Mail Adressen der berechtigten Nutzer hinterlegt. In der Publishing License einer Datei waren nur die drei Gruppen E-Mail Adressen und deren Berechtigungen angegeben [4]. Das hatte zur Folge, dass die Publishing License bei einer Änderung der Berechtigungen nicht verändert wurde. Die Gruppen E-Mail Adressen waren vermutlich Active Directory Gruppen zugeordnet.

4.2. Erstellung der Credentials bei Registrierung und Login

Während der Registrierung wurden die Anmeldedaten auf Anforderung des Clients beim Tresorit RMS Server erstellt (siehe Nachricht CreateRmsCredentials() in Sequenzdiagramm 3.4). Anschließend wurden durch den Client drei Serveradressen angefragt: $tenantHost$, $rmsServerHost$ und $adrAddinUrl$.
Die Adresse $tenantHost$ = "tresorit.onmicrosoft.com" war die Mandanten-Domain von Tresorit in Microsoft RMS. Diese Domain war sowohl die Domain der Microsoft RMS E-Mail Adressen der Nutzer als auch die Domain der Gruppen E-Mail Adresse der Tresorit DRM geschützten Tresors.
Die Adresse $rmsServerHost$ = "9e589980-d0b8-4473-bcf1-3bcd2f73ab76.rms.eu.aadrm.com" war die Adresse des Microsoft RMS Servers. Sie war für beide Testbenutzer gleich.
Der Link $adrAddinUrl$ = "https://installerstorage.blob.core.windows.net/public/rms/tresorit_drm_addin.exe" verwies auf eine ausführbare Datei. Diese wurde während den Beobachtungen nicht aufgerufen. Daher war der Zweck der Adresse mit den vorliegenden Daten nicht erklärbar.

Anschließend wurden $mail_{RMS_{user}}$ und $pwd_{RMS_{user}}$ vom Tresorit RMS Server übertragen. Da diese jeweils keinen erkennbaren Zusammenhang mit der gewählten E-Mail Adresse und dem gewählten Passwort des Nutzers aufwiesen, wurde davon ausgegangen, dass diese Microsoft RMS Credentials durch den Tresorit RMS Server erstellt wurden und am Microsoft RMS Server registriert wurden. Des Weiteren konnte davon ausgegangen werden, dass diese Credentials dauerhaft im Klartext auf dem Tresorit RMS Server gespeichert waren, da während des Logins auf dem zweiten Gerät die gleiche Übertragung stattfand.

4.3. Aktivierung von Tresorit DRM

Zur Nutzung von Tresorit DRM wurden die entsprechenden Berechtigungen bei den Entwicklern angefragt. Nach der Bestätigung der erfolgreichen Berechtigungsvergabe wurde die virtuelle Maschine gestartet. Anschließend wurde zuerst Tresorit DRM aktiviert und danach installiert. Das Protokoll dieses Prozesses wird im Folgenden beschrieben:

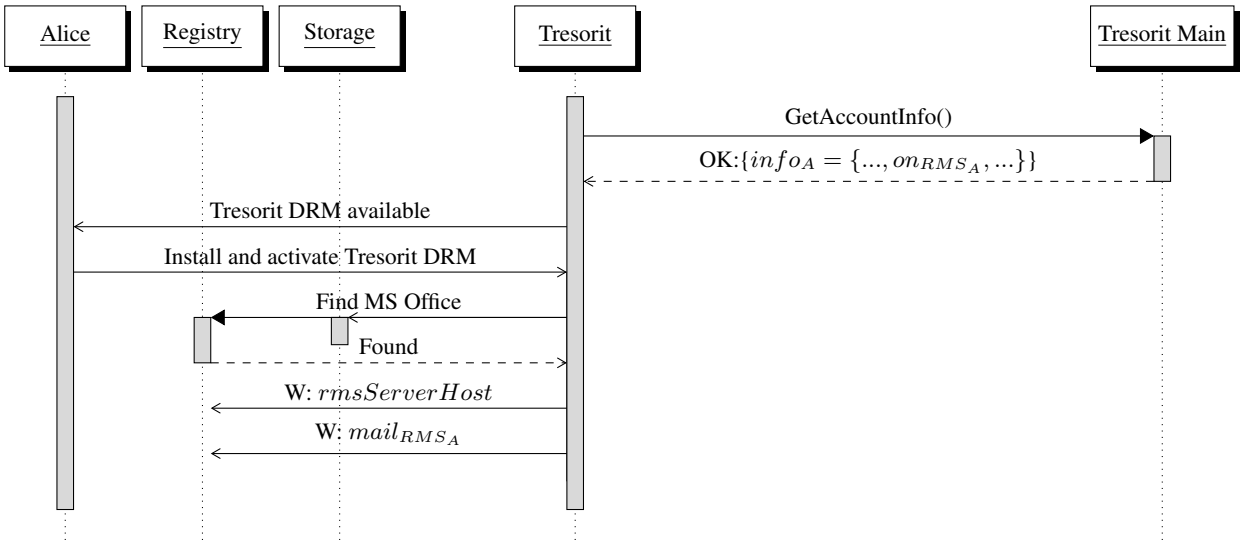

Abbildung 4.2.: Sequenzdiagramm der Aktivierung von Tresorit DRM

Nachdem der Nutzer aktiv zur Integration von Tresorit DRM zustimmte, wurde zuerst nach einer installierten Microsoft Office Version gesucht. Die Suche fand mittels "CreateFile"- und "RegOpenKey"-Zugriffen auf den Speicher und die Registry statt, weshalb die zuvor definierten Filter für die Verfolgung der Suche außer Acht gelassen werden mussten.

Alle Dateipfade, die von Tresorit zur Suche und Identifizierung der installierten Microsoft Office Version durchsucht wurden, existierten nicht und wurden vom System mit einem "PATH NOT FOUND" oder "NAME NOT FOUND" abgelehnt. Obwohl nach Office Version 14.0 und 15.0 gesucht wurde und das installierte Office die Version 15.0 hatte, stimmten die gesuchten Pfade nicht mit dem Tatsächlichen überein.
Nach der erfolglosen Suche nach Dateien, suchte die Software in der Registry nach Microsoft Office Installationen. Das Auslesen des Schlüssels
HKLM\SOFTWARE\Wow6432Node\Microsoft\Office\15.0\Common\InstallRoot\Path
war nicht möglich, da dieser nicht existierte. Anschließend wurden weitere Registry Pfade durchsucht, bis im Wert des Pfades
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\UserData\S-1-5-18\Products
00005109C80000000000000000F01FEC\InstallProperties\DisplayVersion
die Versionsnummer *15.0.4737.1003* und im Wert des Pfades
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\UserData\S-1-5-18\Products
00005109C80000000000000000F01FEC\InstallProperties\DisplayName
der Name "Office 15 Click-to-Run Extensibility Component" gefunden wurde.

Die gleiche Suche mit den gleichen Ergebnissen konnte vier Mal hintereinander beobachtet werden. Nach dem vierten Mal wurden Benutzerdaten in die Registry geschrieben. Diese Daten sind in Tabelle 4.3 aufgeführt.

Registrypfad	**Datentyp**	**Wert**
HKCU\Software\Classes\Local Settings\Software\Microsoft\ MSIPC\9e589980-d0b8-4473-bcf1-3bcd2f73ab76.rms.eu. aadrm.com\UserConsent = HKCU\Software\Classes\Local Settings\Software\ Microsoft\MSIPC\$rmsServerHost$\UserConsent	REG_- DWORD	1
HKCU\Software\Microsoft\Office\15.0\Common\Identity\ Identities\0000000000000000_OrgId\IdP	REG_- DWORD	2
HKCU\Software\Microsoft\Office\15.0\Common\Identity\ Identities\0000000000000000_OrgId\ProviderId	REG_ SZ	0000000000000000
HKCU\Software\Microsoft\Office\15.0\Common\Identity\ Identities\0000000000000000_OrgId\SigninName	REG_ SZ	2zkiib2u1fhxol9s@ tresorit.onmicrosoft.com = 2zkiib2u1fhxol9s@ $tenantHost$ = $mail_{RMS_A}$
HKCU\Software\Microsoft\Office\15.0\Common\Identity\ Identities\0000000000000000_OrgId\Persisted	REG_- DWORD	1

Tabelle 4.2.: Übertragung der Nutzerdaten per Regsitry

Um zu verifizieren, dass das Schreiben dieser Daten in die Regsitry, das Anmelden des Nutzers in Microsoft Office verursachte, wurde der Vorgang manuell durchgeführt. Dazu wurde die Sicherungskopie der virtuellen Maschine (X100) (siehe Abschnitt 2.3.2) gestartet. Auf dieser Maschine war Tresorit noch nicht installiert. Die in Tabelle 4.2 aufgelisteten Daten wurden in die zugehörigen Pfade der Registry geschrieben. Da einige Pfade nicht existierten, wurden sie vorher erstellt. Wenn keine Internetverbindung bestand, konnte beim Start von Microsoft Word festgestellt werden, dass der Nutzer angemeldet war. Sobald die Internetverbindung hergestellt war, wurde der Nutzer aufgefordert, sich erneut anzumelden. Wenn der erste Start von Microsoft Word, nach dem manuellen Eintragen der Daten, mit Internetverbindung stattfand, war der Nutzer nicht angemeldet.

Die manuelle Anmeldung ohne Internetverbindung konnte nicht genutzt werden, um Microsoft RMS geschützte Dateien mit Microsoft Word zu öffnen.

Aufgrund dieser Beobachtungen konnte gefolgert werden, dass auch andere Umstände für die erfolgreiche Anmeldung in Microsoft Office notwendig waren. Zusätzliche Registry oder Datei Pfade konnten nicht identifiziert werden.

4.4. Tresor Erstellung

Die Nachricht GetRmsGroupNames() in Sequenzdiagramm 4.3 lieferte eine ID zurück. Diese wurde aus folgenden Gründen $ID_{RMS_{RShT}}$ bzw. $ID_{RMS_{tresor}}$ genannt:
Zum einen war die ID Bestandteil der drei Gruppen E-Mail Adressen für die Zuordnung der Microsoft RMS Berechtigungen des Tresors: "group-$ID_{RMS_{RShT}}$-managers@$tenantHost$", "group-$ID_{RMS_{RShT}}$-editors@$tenantHost$" und "group-$ID_{RMS_{RShT}}$-readers@$tenantHost$", also beispielhaft "group-a884as6a4sd541c-managers@tresorit.onmicrosoft.com". Diese Gruppen E-Mail Adressen wurden bei der Anfrage der Publishing License während des Uploads erstellt.
Zum anderen war als *ReferralInfo* in der Anfrage der Publishing License diese $ID_{RMS_{RShT}}$ und die Mail Adresse des Nutzers $mail_A$ angegeben.
Letztlich konnte neben dem beschriebenen Versuch, bei der Untersuchung weiterer Log-Daten, festgestellt werden, dass die Gruppen E-Mail Adressen für verschiedene Dateien eines Tresors gleich, für verschiedene Tresors aber unterschiedlich waren.

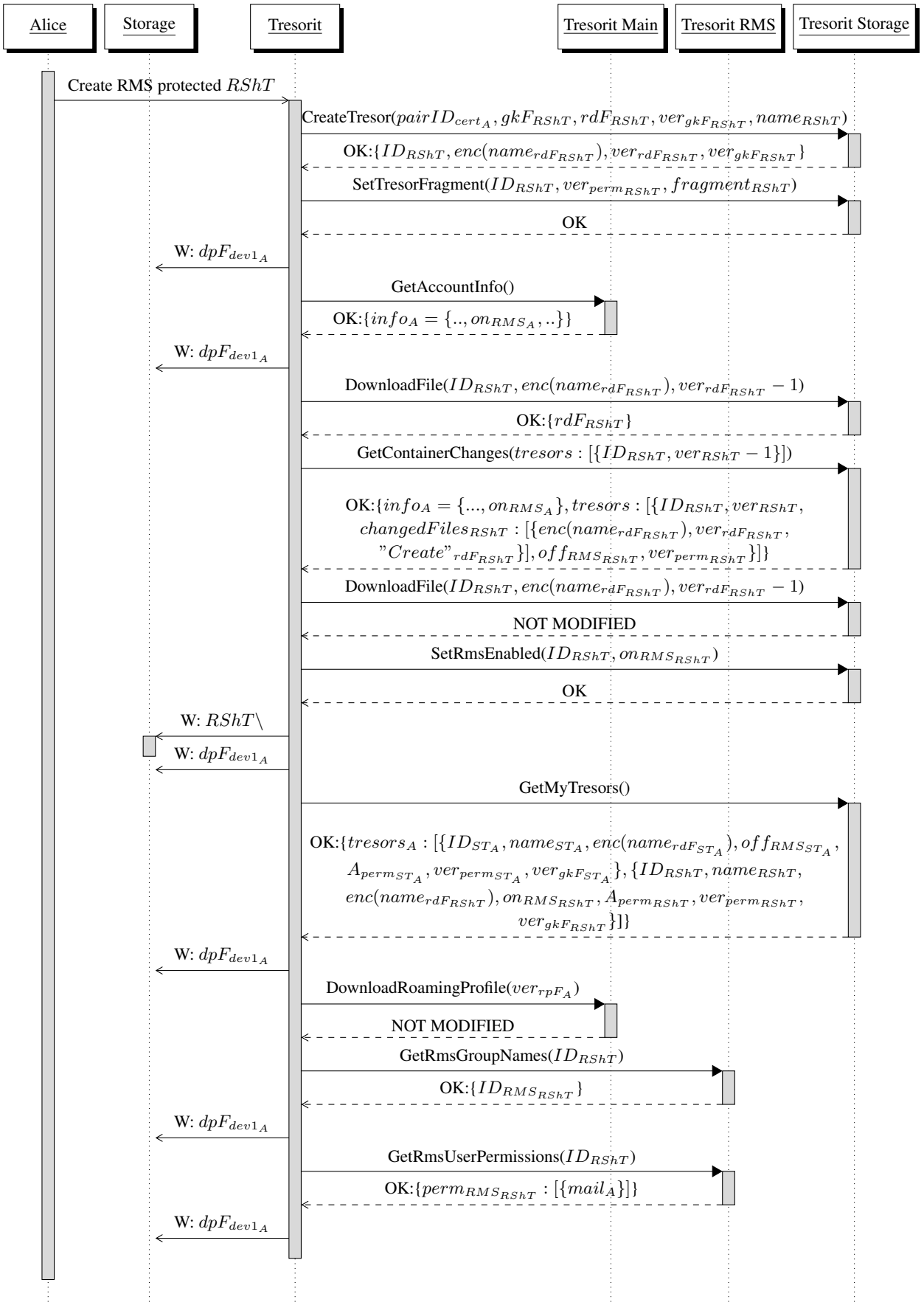

Abbildung 4.3.: Sequenzdiagramm der Erstellung eines Tresorit DRM geschützten Tresors

Nachdem der Tresor $RShT$ erstellt wurde, erreichte die Nachricht SetRmsEnabled(), dass der Tresor Tresorit DRM bzw. Microsoft RMS geschützt war. In Folge dessen wurden am Schluss des Protokolls die $ID_{RMS_{RShT}}$ und die berechtigten RMS Nutzer abgefragt. Wie in Sequenzdiagramm 4.3 erkennbar ist, wurden diese beiden Abfragen zwischen Client-Software und dem Tresorit RMS Server kommuniziert.

4.5. Datei Upload

Das Upload Protokoll für einen Microsoft RMS geschützten Tresor kann in drei Phasen eingeteilt werden: Initialisierungsphase, Authentisierungsphase und Upload.
Die Authentisierungsphase trat einmalig auf, als der Nutzer bzw. die Client-Software noch kein Zugangstoken ($accessToken_{RMS}$) für die Authentisierung am Microsoft RMS Server besaß.

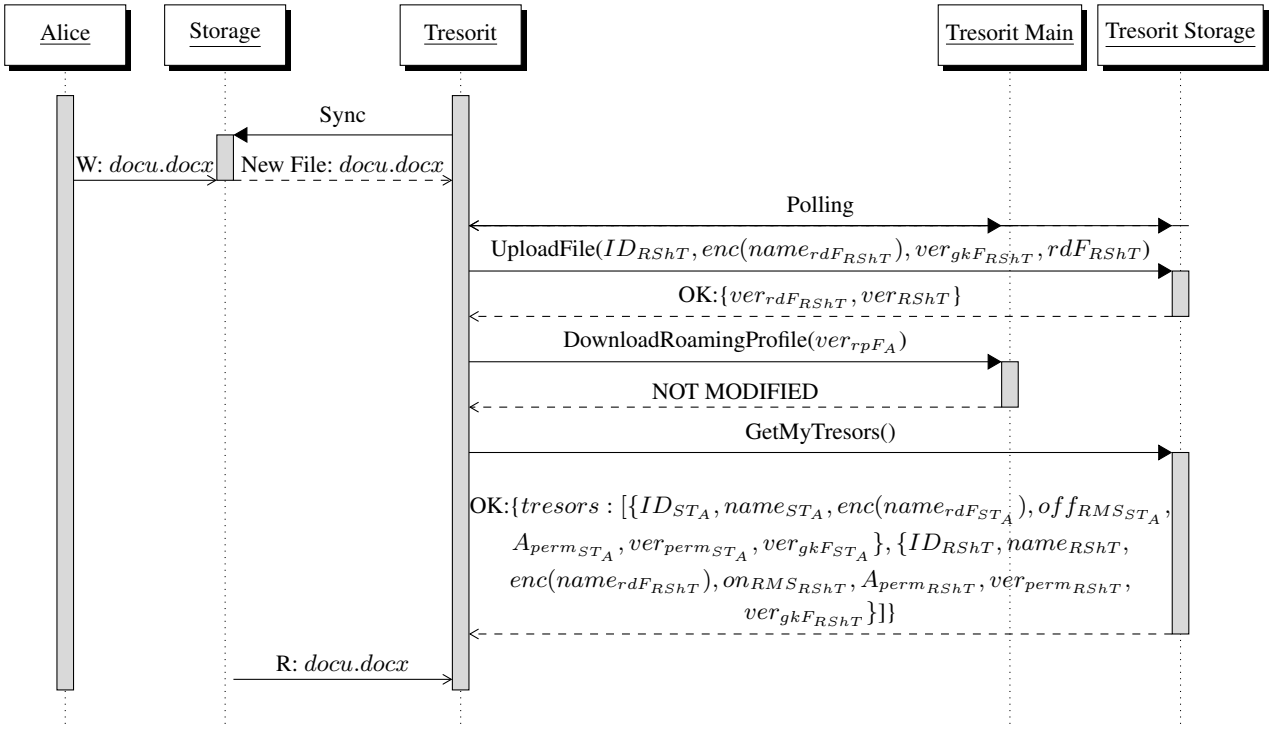

Abbildung 4.4.: Sequenzdiagramm der Initialisierung des Upload-Protokolls eines Tresorit DRM geschützten Tresors

Die Initialisierung des Uploads verlief, wie in den Sequenzdiagrammen 4.4 und 3.11 ersichtlich ist, unabhängig vom Microsoft RMS Schutz.

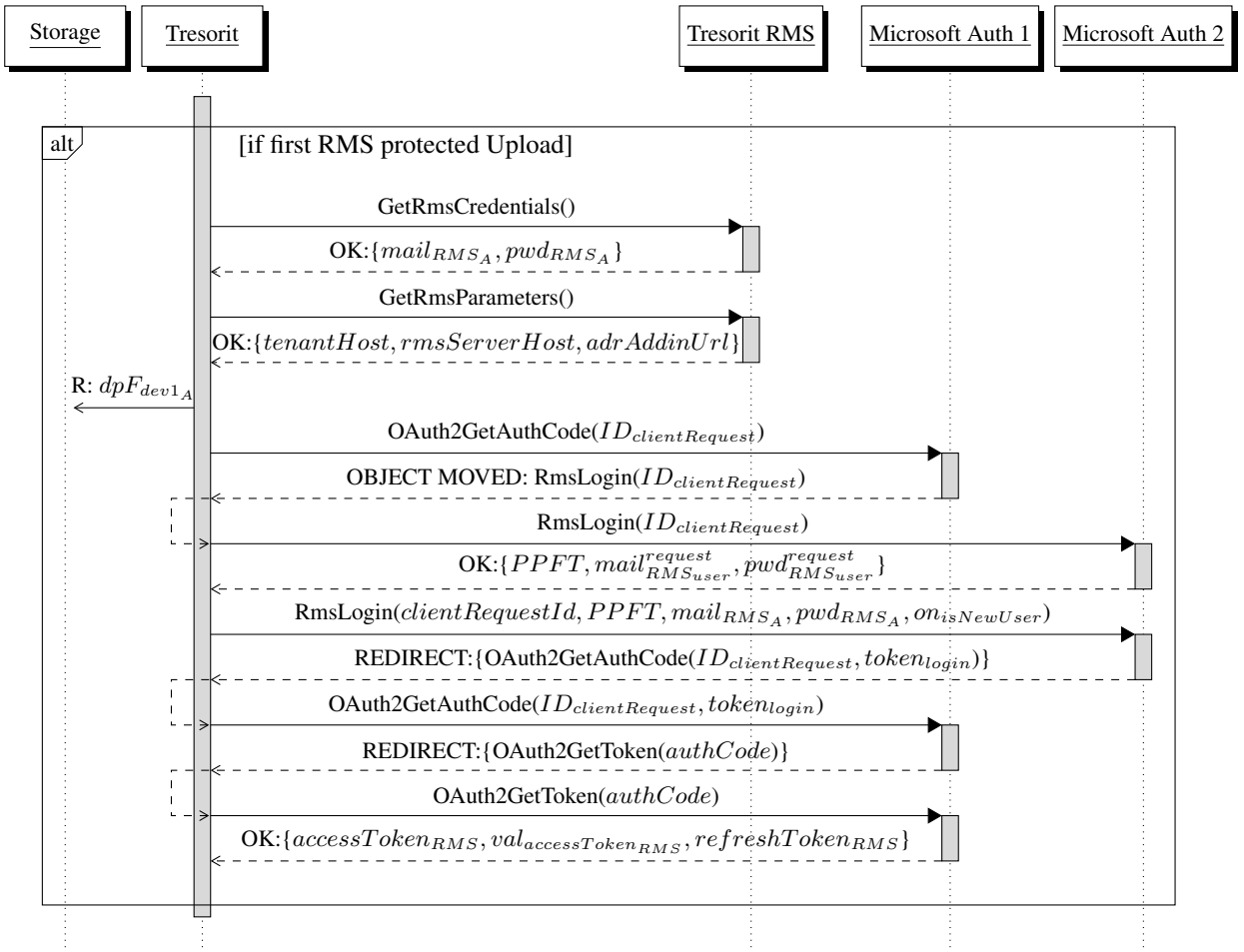

Abbildung 4.5.: Sequenzdiagramm des Authentifizierungsprotokolls beim Upload eines Tresorit DRM geschützten Tresors

Beim zweiten beobachteten Upload, der außerhalb des Versuchs durchgeführt wurde, war entweder nur der $accessToken_{RMS}$ oder auch der $refreshToken_{RMS}$ abgelaufen. Daraufhin fand kein Token-Update statt. Aufgrund dessen wurde die Anfrage nach der Publishing License für diesen zweiten Upload abgelehnt. In Folge dessen konnte der Upload nicht erfolgreich durchgeführt werden[1]. In den Log-Daten dieses zweiten Uploads konnte, wie beschrieben, keine zweite Authentifizierung beobachtet werden, bevor die Anfrage nach der Publishing License abgelehnt wurde.

[1] Nach einem Neustart der virtuellen Maschine führte Tresorit ein Update durch. Dieses entsprach nicht mehr der zuvor verwendeten Version und konnte daher nicht genutzt werden.

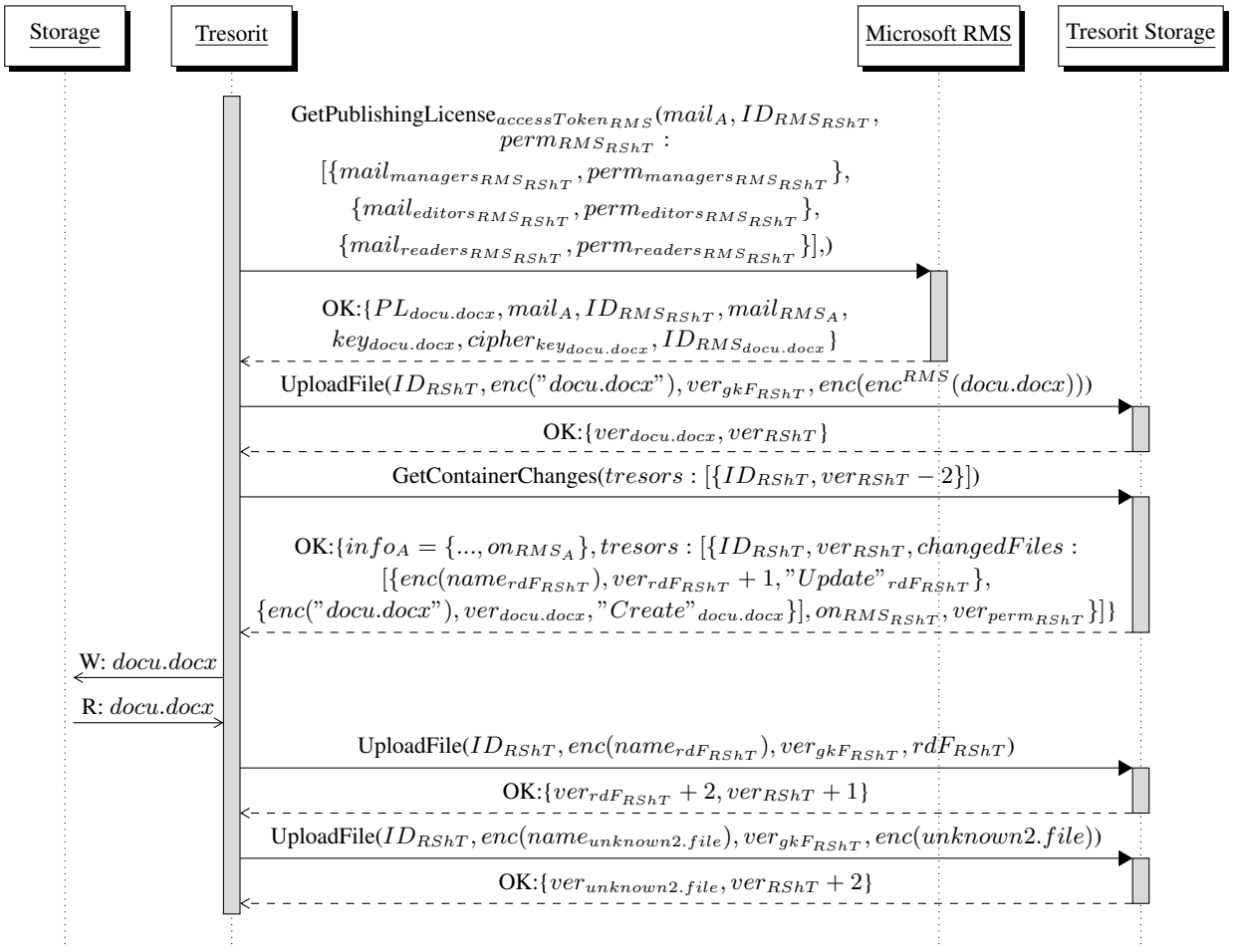

Abbildung 4.6.: Sequenzdiagramm vom Upload-Protokoll eines Tresorit DRM geschützten Tresors

Die zuletzt hochgeladene Datei konnte, wie bei der Synchronisation eines weiteren Geräts in Sequenzdiagramm 3.10, keiner auf dem lokalen Speicher existierenden Datei zugeordnet werden. Laut Protokoll betrug die Dateigröße 3 Byte.

Der $accessToken_{RMS}$, der während der initialen Authentifizierung übertragen wurde, konnte in der Anfrage GetPublishingLicense() in den Log-Daten des modifizierten Clients nicht gefunden werden. Aufgrund der Beobachtungen des beschriebenen zweiten Upload-Versuchs wurde trotzdem angenommen, dass der Token in der Nachricht, außerhalb des mitgeschnittenen Inhalts, mitgesendet wurde.

Die Publishing License wurde nach der Verschlüsselung der Datei $docu.docx$ an diese Datei angehängt und mittels UploadFile() auf den Server geladen. Auf dem lokalen Speicher des Nutzers wurde die zuvor unverschlüsselte Datei anschließend durch die verschlüsselte Datei mit angehängter Publishing License ersetzt.

Publishing License

Die Daten der Base64-enkodierten Publishing License waren im XML-Format formatiert. Als "Issuer" der Lizenz war "Tresorit Kft." mit der $rmsServerHost$ Adresse und dem zugehörigen öffentlichen Teil eines RSA-Schlüssels angegeben. Als "IssuedPrincipal" waren dieselben Daten ein zweites Mal eingetragen. Als "DistributionPoint" war die $rmsServerHost$ Adresse zusätzlich aufgeführt. Außerdem enthielt die Publishing License die RMS E-Mail Adresse des Datei-Besitzers $mail_{RMS_A}$ und die verschlüsselten Berechtigungsangaben zur Datei. Diese Daten (das gesamte Body-Element) waren RSA PKCS#1 V1.5 signiert. Der verwendete Hash-Algorithmus war SHA-256.

4.6. Zugriffsberechtigung Erteilen (Share)

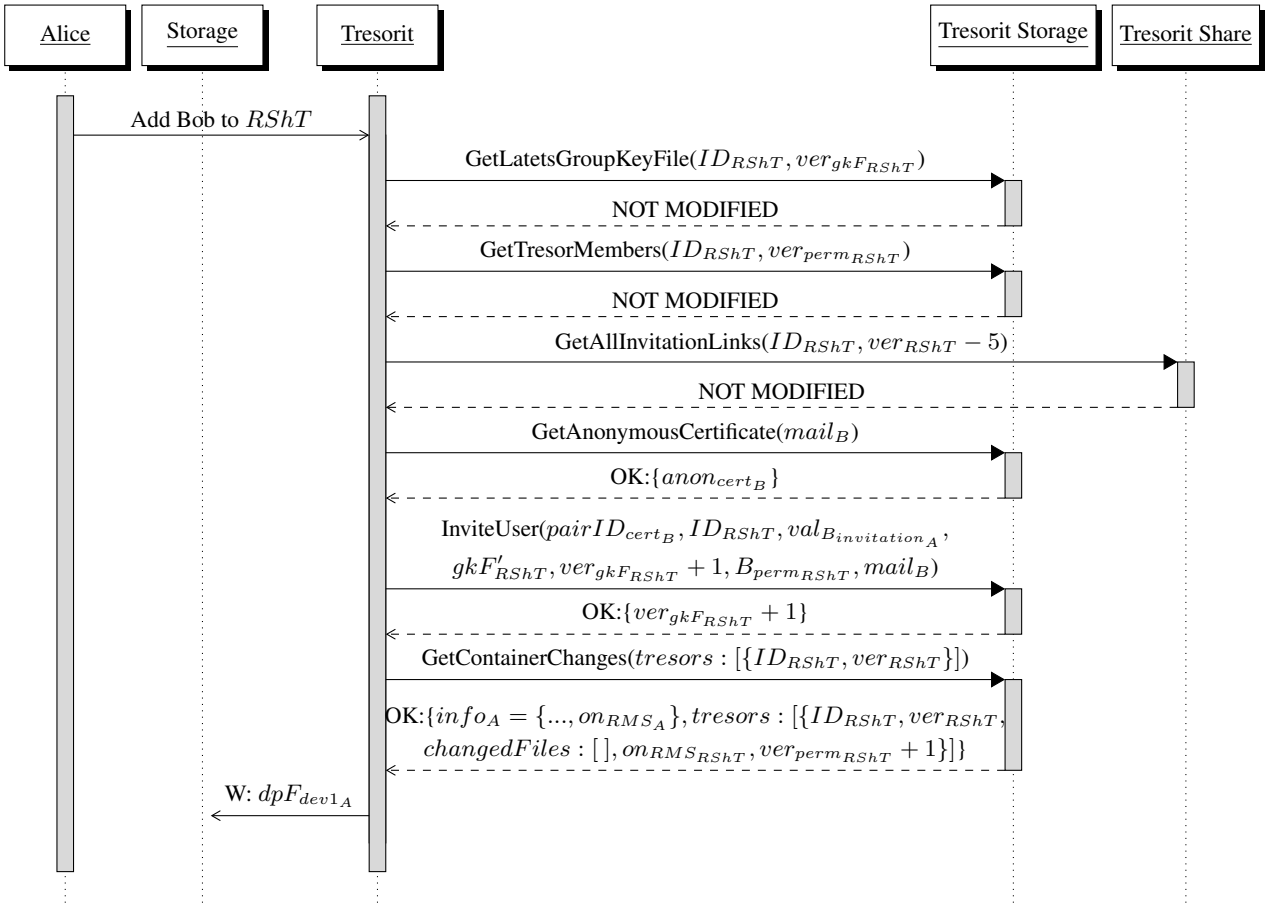

Abbildung 4.7.: Sequenzdiagramm des Erteilens von Berechtigungen zu einem Tresorit DRM geschützten Tresor

Das Protokoll in Sequenzdiagramm 4.7 gleicht dem in Sequenzdiagramm 3.13 bis auf einen Schreibzugriff auf das Device Profile. Damit änderte sich für das Erteilen von Berechtigungen unter der Verwendung von Tresorit DRM nichts im Vergleich zum Protokoll ohne Tresorit DRM. Daraus konnte geschlossen wer-

den, dass die Microsoft RMS Berechtigungen durch die Server von Tresorit verwaltet wurden.

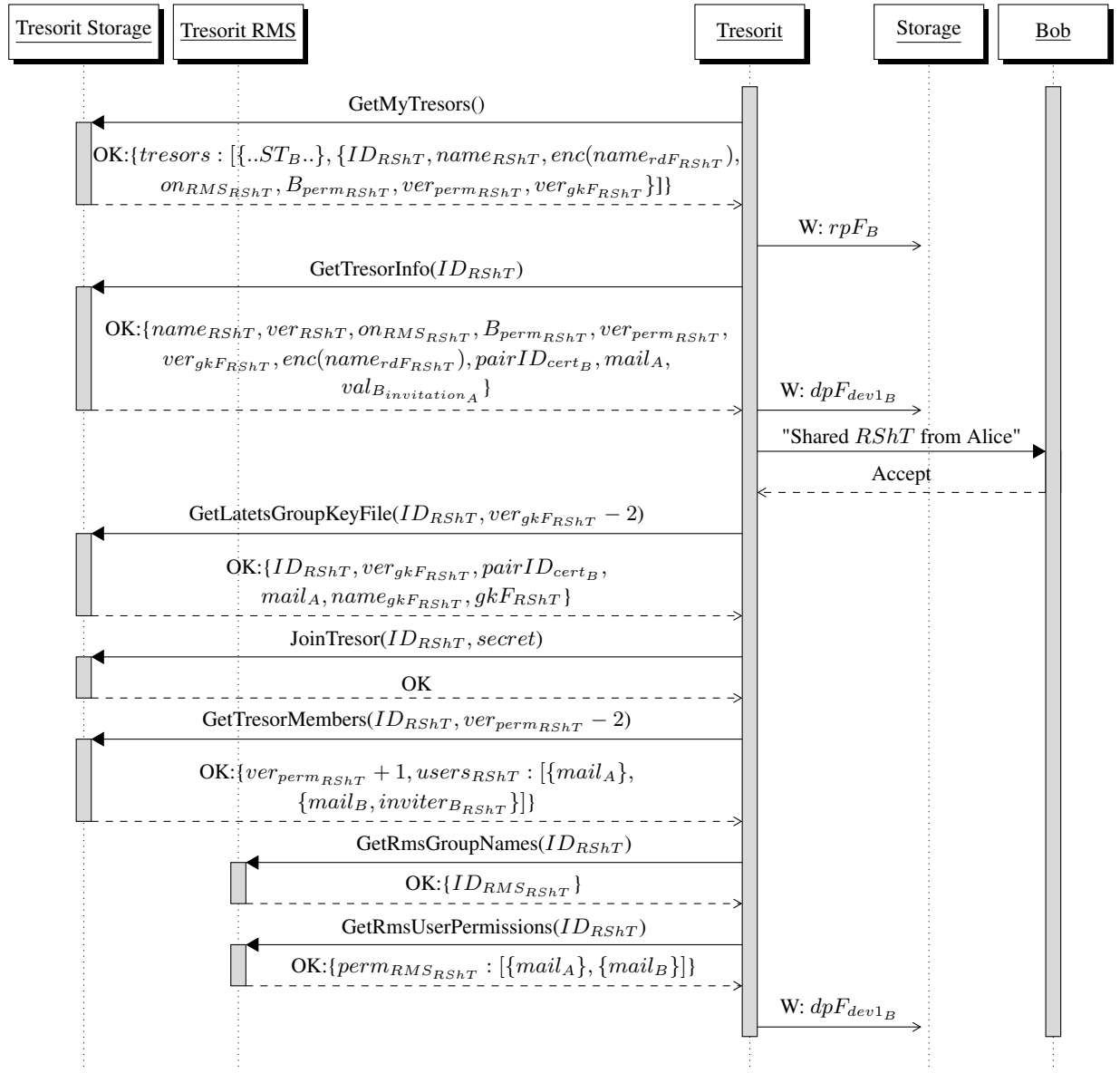

Abbildung 4.8.: Sequenzdiagramm vom Protokoll des Erlangens von Berechtigungen

Wie beim Erlangen von Berechtigungen zu einem Tresor ohne Microsoft RMS Schutz, übertrug der Empfänger der Einladung ein Geheimnis $secret$ an den Storage Server. Der Ursprung wurde auch hier in der Group Key File vermutet. Bis auf die Anfragen zu Microsoft RMS glich das Protokoll dem des Erlangens von Berechtigungen ohne RMS Schutz.

An der Antwort der Nachricht GetRmsUserPermissions() in Sequenzdiagramm 4.8 und der Nachricht InviteUser() in Sequenzdiagramm 4.7 kann erkannt werden, dass ein Server, der sowohl mit dem Tresorit Storage Server als auch dem Tresorit RMS Server verbunden war, die RMS Berechtigungen, nach dem Er-

teilen durch Nutzer Alice, aktualisiert hat. Dabei war nicht auszuschließen, dass eine direkte Verbindung zwischen den Servern bestand, und auf einem der beiden Server diese Aktualisierung durchgeführt wurde.

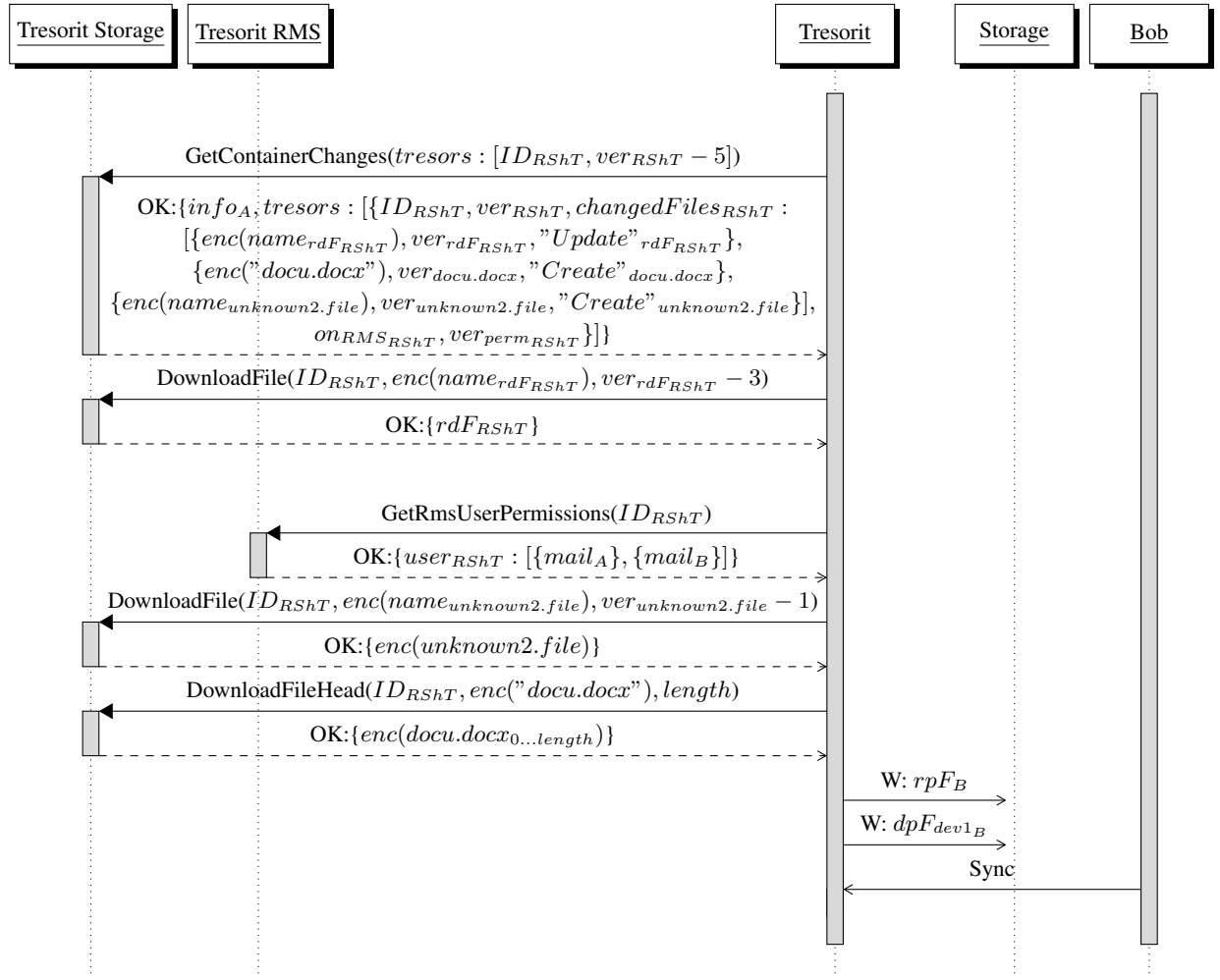

Abbildung 4.9.: Sequenzdiagramm der Synchronisation eines Tresorit DRM geschützten Tresors nach dem Erlangen der Berechtigungen

Die Synchronisation des Tresors verlief nach demselben Protokoll wie in Sequenzdiagramm 3.15 mit den entsprechenden Dateien.

4.7. Zugriffsberechtigung Entziehen (Revoke)

Auch das Entziehen der Berechtigungen verlief nach denselben Protokollen, wie es ohne Tresorit DRM beobachtet und beschrieben wurde. 20 Sekunden nach dem Entziehen wurden im regelmäßigen Polling des Clients von Alice - also der entziehenden Partei - die RMS Berechtigungen abgefragt. Trotz Entziehung der Zugriffsberechtigung war der entfernte Nutzer Bob in der Antwort des Tresorit RMS Servers noch als

berechtigt aufgeführt. Dies ließ darauf schließen, dass die Entziehung der Berechtigungen nicht zeitgleich von einer Instanz durchgeführt wurde. Bei einer später ausgelösten Abfrage der Berechtigungen war die Entziehung der Berechtigungen durchgeführt, sodass nur noch Alice in der Liste der berechtigten Nutzer des Tresorit RMS Servers aufgeführt war.

4.8. Öffnen der RMS geschützten Datei

Wie erwartet war das Öffnen der Datei mit der erteilten Berechtigung möglich. Nach dem Entziehen der Berechtigung konnte die Datei nicht mehr mit Microsoft Word geöffnet werden.

Da die Kommunikation zwischen Microsoft Word und den Servern verschlüsselt war, konnten keine Inhalte der Kommunikation ausgelesen werden. Aus diesem Grund konnte keine weitere Analyse dieser Protokolle durchgeführt werden.

4.9. Übersicht zu Tresorit DRM Entitäten

Aus den Ergebnissen der Protokollanalyse wurde eine schemenhafte Darstellung (4.10) der Entitäten und deren Beziehungen untereinander bezüglich der Datenverarbeitung von Tresorit DRM erstellt. Die durchgezogenen Linien in der Abbildung können mit den beschriebenen Beobachtungen belegt werden. Für die Beziehungen der Server, die mit gestrichelten Linien dargestellt sind, wurden Annahmen auf Grund der Beobachtungen getroffen.

Auf Anfrage bei den Entwicklern von Tresorit wurde bestätigt, dass die Relationen und Entitäten der Abbildung mit denen des realen Systems übereinstimmten.

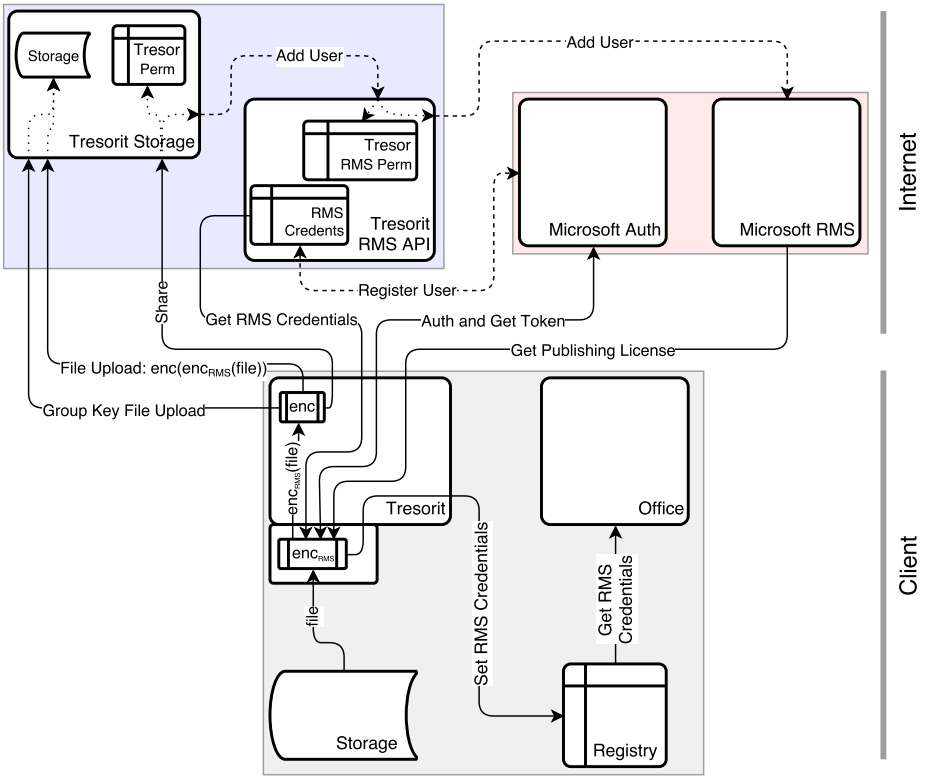

Abbildung 4.10.: Schema von Tresorit DRM

Im Folgenden werden die Aufgaben der Entitäten, gefolgert aus den beschriebenen Beobachtungen und Annahmen, erörtert.

Microsoft Office

Beim ersten Start eines Microsoft Office Programms während einer Systemsession [2] wurde in der Registry nach Microsoft Credentials gesucht. Mit gefundenen Credentials wurde der Benutzer dann in Microsoft Office angemeldet.

Tresorit Storage Server

Alle Berechtigungserteilungen und -entziehungen wurden durch die Tresorit Client-Software an den Tresorit Storage Server gesendet. Die Berechtigungen wurden an den Tresorit RMS Server weitergeleitet. Die daraus resultierenden Änderungen der Tresor E-Mail Gruppen wurden an den Microsoft RMS Server übertragen.

Tresorit RMS Server

Da die RMS Credentials und Berechtigungen beim Tresorit RMS Server angefragt wurden, wurde davon ausgegangen, dass diese Daten auf dem Server gespeichert waren. Neben dem Speichern der RMS Daten

[2] Systemsession wird hier als Zeit zwischen Systemstart und Herunterfahren von Windows definiert.

wurde angenommen, dass der Tresorit RMS Server die Weiterleitung der RMS Berechtigungen bzw. Aktualisierung der Tresor E-Mail Gruppen durchführte.

Daraus folgte, dass die RMS Credentials aller Tresorit Nutzer jederzeit im Klartext auf dem Tresorit RMS Server gespeichert waren. Zusätzlich hatte dieser Server die vollständige Kontrolle über die RMS Berechtigungen der RMS geschützten Dateien, die auf dem Tresorit Storage Server gespeichert waren.

Microsoft Authentication Server

Die Microsoft Authentication Server kannten die RMS Credentials der Benutzter und stellten diesen während des Authentifizierungsprozesses Token für die Authentisierung beim Microsoft RMS Server aus. Die RMS Credentials wurden zuvor zwischen Tresorit RMS Server und Microsoft Authentication Servern kommuniziert.

Microsoft RMS Server

Die $accessToken$ der Microsoft Authentication Server wurden durch den Microsoft RMS Server als valide anerkannt oder, bei Ablauf der Gültigkeit, abgelehnt. Die Gültigkeit wurde dafür in den Token integriert oder zwischen den betreffenden Servern ausgetauscht.

Auf Anfrage der Tresorit Client-Software mit gültigem Token wurde eine Publishing License ausgestellt. Zur Auflösung der Gruppen E-Mail Adressen eines Tresors in einzelne berechtigte Nutzer wurden die Gruppenmitglieder zwischen Tresorit RMS oder Storage Server und Microsoft RMS Server kommuniziert.

Tresorit Client-Software

Die Berechtigungserteilungen und -entziehungen eines Tresors wurden mit dem Tresorit Storage Server kommuniziert. Die RMS Credentials und der Status der RMS Berechtigungen wurden beim Tresorit RMS Server angefragt. Zum erfolgreichen Anfragen der Publishing License beim Microsoft RMS Server wurden Token durch die Microsoft Authentication Server ausgestellt. Dazu wurden die Credentials aus der Kommunikation mit dem Tresorit RMS Server verwendet. Diese Credentials wurden dem Nutzer zur manuellen Authentisierung an Microsoft RMS fähiger Software angezeigt. Außerdem wurden sie automatisch in die Registry geschrieben, damit Software wie Microsoft Office eine automatische Anmeldung des Nutzer durchführen konnte.

Zur Verschlüsselung von Dateien bei aktiviertem Tresorit DRM könnte die Rights Management Services SDK 2.1 [10] verwendet worden sein. Dazu wurde die angefragte Publishing License des Microsoft RMS Servers und der angehängte Schlüssel genutzt. Nach der RMS Verschlüsselung wurde eine Datei, wie ohne Tresorit DRM Schutz, Ende-zu-Ende verschlüsselt und auf den Tresorit Storage Server geladen.

5. Evaluierung der Sicherheit

In diesem Kapitel wird abschließend, beruhend auf den beschriebenen Beobachtungen, evaluiert, ob das System, vor dem Hintergrund der Sicherheitsannahme, als sicher zu bewerten ist.

5.1. Sicherheitsmaßnahmen

Protokollsicherheit

Die Authentizität, Integrität und Vertraulichkeit der Nachrichten war durch die Nutzung der Zertifikate gewährleistet. Wie beschrieben, wurden Nachrichten sowohl signiert als auch verschlüsselt. Mit der Nutzung von Client-TLS-Zertifikaten wurde zusätzlich zur Server-Authentizität die Authentizität des Clients erreicht.

Obwohl die jeweiligen Antworten der Server Aufschluss auf korrekte Verarbeitung boten, wurden bei der Erteilung und Entziehung von Zugriffsberechtigungen Bestätigungen zur vollständigen und korrekten Durchführung der Aktion angefragt. Damit wäre das Blockieren dieser Nachrichten zusätzlich erkannt worden.

Tresorit DRM Integration

Der zusätzliche Schutz durch Micrsoft RMS bot eine dateigebundene Durchsetzung der Berechtigungen. Dabei war Tresorit DRM eine Anbindung der Microsoft RMS Funktionalitäten. Ähnlicher Schutz wäre erreicht worden, wenn die Dateien unabhängig von Tresorit Microsoft RMS geschützt worden wären. Allerdings war mit der Integration auch eine Kombination der Berechtigungen verbunden. Aus dieser folgte, dass die Änderungen der Berechtigungen sowohl für Tresorit als auch für Microsoft RMS umgesetzt wurden.

Der Status der RMS Berechtigungen konnte von der Tresorit Client-Software über Abfragen an den Tresorit RMS Server festgestellt werden. Damit war eine Beobachtung der Berechtigungen ermöglicht.

Kryptografie

Die Algorithmen und Schlüssellängen der verwendeten Kryptoprafie entsprachen dem Stand der Technik.

Zertifikatsstruktur

Zu jedem Gerät lagen zweckgebundene Zertifikate vor. Diese Zertifikatsstruktur ermöglichte die authentifizierte Nutzung mehrerer Geräte. Außerdem besaß jeder Nutzer ein Agreement-Zertifikat-Paar. Dieses bot

Anonymität für die initiale Kommunikation zweier Nutzer.

Die CA-Zertifikate von Tresorit waren in der Client-Software hinterlegt, sodass ein Unterbrechen des TLS-Kanals mit gefälschten Zertifikaten erkannt wurde.

5.2. Einschränkungen der Sicherheit

TLS Server Konfiguration

Die TLS Konfigurationen aller Server, mit denen während der Beobachtungen kommuniziert wurde, wurden mit dem SSL Server Test von Qualys SSL Labs[1] untersucht. Einer der Microsoft Authentifizierungsserver (sts.aadrm.com) und der Microsoft RMS Server (api.aadrm.com) erlaubten die Nutzung der, als unsicher geltenden, RC4 Chiffre. Der Zertifizierungspfad des TLS Zertifikats des Microsoft RMS Servers endete laut Test außerdem in keiner vertrauenswürdigen CA. Bei allen anderen Servern konnten keine sicherheitskritischen Konfigurationen erkannt werden.

Tresorit DRM Integration

Wie beschrieben, beschränkte sich die Kontrolle der Microsoft RMS Berechtigungen auf die Beobachtung mittels Abfragen an den Tresorit RMS Server. Die Kontrolle über die Microsoft RMS Berechtigungen hatten in erster Linie die Microsoft RMS Server. Die Aktualisierung der Berechtigungen wurde unter Vertrauen zu den Tresorit Servern von ebendiesen an den Microsoft RMS Servern durchgeführt. Damit wurde sowohl Vertrauen zu den Tresorit als auch Microsoft Servern benötigt. Dieses Vertrauen war mit der getroffenen Sicherheitsannahme gedeckt, die besagt, dass Service Provider ehrlich aber neugierig sind (siehe Abschnitt 2.2.1). Als Grund für die zentrale Steuerung der Berechtigungen wurde vermutet, dass die Erstellung von Azure Active Directory Gruppen administrativen Zugang zu Active Directory verlangte. Ein administrativer Zugang sollte dem Nutzer vermutlich nicht gewährt werden.
Da die beobachtete Änderung der RMS Berechtigungen nicht gleichzeitig mit der Änderung der Tresorit Berechtigungen stattfand und die darauf folgende Anfrage der RMS Berechtigungen keine Änderung erkennen ließ, war die Kontrolle der RMS Berechtigungen wirkungslos.

Zusätzlich zur Verarbeitung von Berechtigungen speicherte und kommunizierte der Tresorit RMS Server die Microsoft RMS Credentials aller Nutzer. Da diese Daten beim Server im Klartext vorlagen, konnte ein Zugriff auf sie durch die Service Provider nicht ausgeschlossen werden. Falls Tresorit als Service Provider Zugriff auf Tresorit DRM geschützte Daten bekommen hätte, die nicht mit der Tresorit Ende-zu-Ende Verschlüsselung geschützt gewesen wären, hätte es die Microsoft RMS Credentials nutzen können, um Zugriff auf die Daten zu bekommen. Dieses Szenario ist nicht von der Sicherheitsannahme gedeckt, da sich Tresorit ehrlich aber neugierig verhalten hätte. Im Angreifermodell eines neugierigen und ehrlichen Servers bzw. Service Providers ist damit der Besitz der Klartext RMS Credentials ein Angriffsvektor, der bei gleichzeitigem Besitz der Tresorit-entschlüsselten Datei zur Verletzung der Vertraulichkeit führt. Dass dieses Szenario

[1] `https://www.ssllabs.com/ssltest/`

Relevanz besitzt, wird dadurch unterstützt, dass es von den Entwicklern in der eigenen Veröffentlichung erstellt wurde (siehe Abschnitt 2.2.1). In dieser Veröffentlichung wurde allerdings behauptet, dass Tresorit in dem Szenario keinen Zugriff auf die Daten hätte.

Zertifikatsstruktur

Es konnte festgestellt werden, dass, wie in der Untersuchung der Zertifizierung durch Forscher der Johns Hopkins University, Baltimore, Maryland [9] beschrieben, Tresorit als CA für die Nutzer-Zertifikate diente. Dadurch hatte Tresorit die vollständige Kontrolle über die Zertifikate der Nutzer und hätte sie zu dem, in der Veröffentlichung beschriebenen, Angriff nutzen können. Der Angriff, gefälschte Zertifikate für die Nutzer auszustellen, um als Man-in-the-Middle die Ende-zu-Ende Verschlüsselung zu umgehen, hat mit der getroffenen Sicherheitsannahme keine Gültigkeit, da die Ausstellung falscher Zertifikate unehrlich wäre. Die Überprüfbarkeit der Ehrlichkeit, also speziell die Überprüfung und der Vergleich der Zertifikate, um diesen Angriff auszuschließen, war für Nutzer der Standard Client-Software nicht möglich.

6. Fazit

6.1. Erreichte Ziele und erlangte Erkenntnisse

Die Analyse der Systemarchitektur erfolgte in den drei geplanten Schritten. Dazu wurde zunächst die vorhandene Dokumentation untersucht. Die daraus resultierenden Erkenntnisse konnten dann für die Beobachtung der Software in einer Testumgebung genutzt werden. Alle Beobachtungen und Erkenntnisse wurden schließlich bezüglich Sicherheit evaluiert. Mit dieser Herangehensweise konnte eine umfangreiche Beschreibung des Systems erfolgen.

Die Dokumentation lieferte wesentliche Erkenntnisse zur genutzten Technik. Sie offenbarte außerdem ein grobes Schema aller Entitäten und Relationen des Systems. Die Spezifikationen waren dabei auf drei verschiedenen Wegen veröffentlicht: Zunächst boten die beiden wissenschaftlichen Veröffentlichungen eine theoretische Basis des Systems [3], [5]. Außerdem wurden drei Whitepaper zu Tresorit veröffentlicht, die die aktuellere Implementation näher beschrieben [6], [4], [1]. Schließlich ergänzte und aktualisierte das Support Forum von Tresorit die Spezifikationen [8].

Mit der beschriebenen Testumgebung wurden alle Aktivitäten des Tresorit Clients aufgezeichnet. Mit den Aufzeichnungen konnten die Protokolle des Systems vollständig beschrieben werden. Zusätzlich zu den Protokollen konnte mit der Beobachtung des Systems die Zertifikatsstruktur untersucht werden.

Im Fokus der Untersuchung der Protokolle stand die Integration von Microsoft RMS. Die Integration konnte auf zwei Ebenen untersucht werden. Zum einen konnte mit Hilfe der Aufzeichnungen die Kommunikation auf dem Betriebssystem zwischen Tresorit und Microsoft Office beschrieben werden. Zum anderen wurden die Netzwerkaktivitäten ausgewertet, um die Aktivitäten der involvierten Server zu beschreiben. Die Folgerungen der Beobachtungen aus beiden Ebenen führten zu einem umfangreichen Bild der Entitäten und Relationen von Tresorit und Tresorit DRM.

Abschließend wurden die Beschreibungen hinsichtlich Sicherheit evaluiert. Des Weiteren bieten die Beschreibungen einen Ansatzpunkt für anschließende Untersuchungen von Tresorit.

6.2. Bewertung der Sicherheit von Tresorit

Die Sicherheit von Tresorit beruhte auf dem Vertrauen in das System. Dieses Vertrauen war unabdingbar, zumal die Quellcodes der Client- und Server-Software nicht veröffentlicht waren. Die gewählte Sicherheitsannahme, die das Vertrauen in die Ehrlichkeit der Service Provider fordert, entstammte der Veröffentlichung der ursprünglich zugrundeliegenden Speicherarchitektur.

Aufgrund der Daten, die Tresorit für die Integration von Tresorit DRM vom Nutzer besaß und für ihn verwaltete, war das genannte Vertrauen zum Erreichen von Vertraulichkeit der Dateien erforderlich:
Hätte in die Microsoft RMS Server, die Tresorit Server und die Client-Software getrennt Vertrauen ausgesprochen werden können, wäre das Vertrauen in die Client-Software unabdingbar, da diese den Datenfluss zu den Servern kontrollierte. Einem der Service Provider - Microsoft oder Tresorit - hätte dann aber das Vertrauen entzogen werden können, ohne dass dies das Vertrauen in das gesamte System beeinflusst hätte. Dies wäre möglich gewesen, wenn die Tresorit Client-Software die Microsoft RMS Berechtigungen selbst gesteuert hätte und die Microsoft RMS Credentials nicht auf den Tresorit Servern gespeichert gewesen wären. Denn dann hätte die korrekte Durchsetzung der Berechtigungen eines der beiden Systeme die Vertraulichkeit der Daten sichergestellt.

Es konnte gezeigt werden, dass die Integration von Microsoft RMS keinen zusätzlichen Schutz vor Zugriffen des Service Providers bot. Falls dieser Zugriff auf unverschlüsselte Dateien bekommen hätte, hätte der Microsoft RMS Schutz den Zugriff nicht verhindert. Vor dem Hintergrund, dass dieses Szenario in Veröffentlichungen von Tresorit mit dem gegenteiligen Schluss betrachtet wurde, ist dieser Teil der Systemarchitektur bezüglich Sicherheit zu bemängeln. Insgesamt ist die Sicherheit des Systems davon unbeeinträchtigt.

6.3. Verwandte Arbeiten

Die Zertifizierung der Nutzer-Schlüssel in Tresorit wurde bereits ansatzweise untersucht [9]. Da die Untersuchung mittels Reverse-Engineering stattfand, konnte nur herausgefunden werden, dass Tresorit als Herausgeber der Nutzer-Zertifikate handelte. In dieser Bachelorarbeit wurde zusätzlich beschrieben, welche Zertifikate im System existierten, wie die Zertifizierungspfade aussahen und wann die Zertifikate an Nutzer übertragen wurden.

Neben Tresorit existieren weitere Ende-zu-Ende verschlüsselte Cloud Storage Systeme:
Wuala [11] basiert auf Cryptree [12], einer baumbasierten, kryptografisch gesicherten Verzeichnisstruktur. Die Einstellung des Dienstes Wuala wurde am 17.08.2015 angekündigt[1]. TeamDrive [13] verwendet ein proprietäres hybrides Verschlüsselungsverfahren, welches dem von Tresorit ähnelt. Ein weiteres Konzept wurde in Zusammenarbeit der Ruhr-Universität Bochum und der Technischen Universität Dortmund erstellt: Sec[2]

[1] `https://support.wuala.com/2015/08/wuala-shutdown-notice/`

[14] nutzt eine Trennung der Daten und zugehörigen Schlüssel, damit die Daten auf einem unsicheren Speicher abgelegt werden können. Im Gegensatz zu den genannten Systemen strebt Qabel [15], das sich noch in der Entwicklung befindet, an, kein Vertrauen der Nutzer in das System zu erfordern. Laut der Spezifikation wird die Schlüsselübertragung persönlich bzw. über einen sicheren Kanal durchgeführt.

Die Integration von Microsoft RMS konnte bei keinem ähnlichen Projekt gefunden werden.

6.4. Ausblick

Diese Arbeit bietet die Grundlage für weitere Untersuchungen der Architektur von Tresorit und Tresorit DRM bezüglich Sicherheit:

Zunächst wurden neben Windows keine weiteren Betriebssysteme in die Analyse einbezogen. Die Untersuchung der Software auf anderen Betriebssystemen kann analog zur Herangehensweise dieser Arbeit durchgeführt werden. Die Analyse des Systems auf mobilen Endgeräten sowie die Untersuchung von *Encrypted Links* stellen weitere Fortsetzungen der Sicherheitsevaluierung dar (Siehe Tabelle 2.2).

In dieser Arbeit wurden nur einzelne Dateien im System beobachtet. Eine Überprüfung, welchen Einfluss Verzeichnisstrukturen auf die beschriebenen Protokolle haben, und darüber hinaus ein Vergleich zwischen der vorherrschenden Speicherstruktur und der ursprünglichen Beschreibung [3] können weiterführend vollzogen werden.

Zum Erreichen der Sicherheitsziele werden in Tresorit zertifizierte Schlüssel verwendet. Mit der erstellten Zertifikatsstruktur und den gesammelten Rohdaten aus der Beobachtung des Netzwerkverkehrs kann die Verwendung der Zertifikate detaillierter untersucht werden. Außerdem kann der TLS-Kanal zwischen Client und Server auf bekannte Angriffe geprüft werden.

Die gesammelten Informationen zur genutzten Kryptografie können verifiziert werden. Zumal alle Schlüssel im System direkt oder durch Verschlüsselung vom gewählten Nutzer-Passwort abhängen, kann weitergehend der Einsatz der Schlüsselableitungsfunktion untersucht werden.

Literaturverzeichnis

[1] *Tresorit Encrypted Link White Paper*, Tresorit, 2015. [Online]. Available: https://tresorit.com/files/encrypted-link-whitepaper.pdf

[2] Colin Boyd, Anish Mathuria, *Protocols for Authentication and Key Establishment*. Springer, 1998.

[3] Levente Buttyán. István Lám, Szilveszter Szebeni, "Tresorium: cryptographic file system for dynamic groups over untrusted cloud storage," in *41st International Conference on Parallel Processing Workshops*, 2012, pp. 296–303.

[4] *Tresorit DRM White Paper*, Tresorit, 2015. [Online]. Available: https://tresorit.com/files/tresorit-drm-whitepaper.pdf

[5] Levente Buttyán, István Lám, Szilveszter Szebeni, "Invitation-oriented tgdh: Key management for dynamic groups in an asynchronous communication model," in *41st International Conference on Parallel Processing Workshops*, 2012, pp. 269–276.

[6] *Tresorit White Paper*, Tresorit, 2015. [Online]. Available: https://tresorit.com/files/tresoritwhitepaper.pdf

[7] *Tresorit Passwort Authentication Protocols*, Tresorit, 2015. [Online]. Available: https://tresorit.zendesk.com/attachments/token/gb0y4gye4lhhwqk/?name=Tresorit+password+authentication.pdf

[8] *Tresorit Security Support Forum*, Tresorit, 2012-2015. [Online]. Available: https://tresorit.zendesk.com/forums/23076328-Security

[9] Duane C. Wilson, Giuseppe Ateniese, "To share or not to share in client-side encrypted clouds," Johns Hopkins University, Computer Science Department, Information Security Institute, Baltimore, Maryland, Tech. Rep., 2014.

[10] *Rights Management Services SDK 2.1 Informationswebseite*, Microsoft, 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/hh535290%28v=vs.85%29.aspx

[11] LaCie AG, "Wuala webseite." [Online]. Available: https://www.wuala.com/

[12] Dominik Grolimund, Luzius Meisser, Stefan Schmid, Roger Wattenhofer, "Cryptree: A folder tree structure for cryptographic file systems," in *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*, 2006, pp. 189–198.

[13] TeamDrive Systems GmbH, "Teamdrive webseite." [Online]. Available: http://teamdrive.com/

[14] Juraj Somorovsky, Christopher Meyer, Thang Tran, Mohamad Sbeiti, Jörg Schwenk, Christian Wietfeld, "Sec^2: Secure mobile solution for distributed public cloud storages," 2012.

[15] Qabel GmbH, "Qabel dokumentation." [Online]. Available: http://qabel.github.io/docs/

A. Anhänge

A.1. Nachrichten

Im Folgenden ist jeweils die erste beobachtete Übertragung eines Nachrichtentyps angegeben. Falls die Antwort darauf *HTTP Status Code: 304* war, ist die nächste Übertragung mit anderer Antwort aufgeführt. Zur Übersichtlichkeit sind einige Werte der ursprünglichen Übertragung gekürzt. Sie können in der beiliegenden Tabelle Events.ods auf der CD oder im SVN des NDS eingesehen werden.

Beobachtete Anfrage	**Beobachtete Antwort**	**Anfrage in Sequezdiagrammen**	**Antwort in Sequezdiagrammen**
authenticatewithpass?email= *value*&clientresponse= *value*	{"AuthSessionId":"*value*", "EncAuthSessionKey": "*value*","Validity":900, "TwoFactorOptions":[]}	AuthWithPwd($mail_A$, $clientResponse_A$)	OK:{$ID_{authSession}$, $enc(key_{authSession})$, $val_{authSession}$}
changepassword?authsessionid= *value*	{"ETag":"*value*", "NewClientSalt":"*value*", "NewRoamingProfile":"*value*", "NewSaltedPassword":"*value*"}	ChangePassword($ID_{authSession}$, $salt^{new}_{pwd_A}$, $authSecret^{new}_A$, rpF^{new}_A)	OK
creatermscredentials	*HTTP Status Code: 200*	CreateRmsCredentials()	OK
createtresor: {"AgreeCertUniqueId": "*value*", "GroupKeyFileInBase64": "*value*","Metadata":"", "RootDirFileInBase64": "*value*","RootDirKeyVersion": 1,"SharedContainerAttributes":0, "TresorName":"*value*"}	{"ContainerId":"*value*", "RootDirFileName":"*value*", "RootDirVersion":1, "RootDirEtag":"*value*", "GroupKeyFileVersion":1, "GroupKeyFileEtag":"*value*"}	CreateTresor($pairID_{cert_A}$, gkF_{ST_A}, rdF_{ST_A}, $ver_{gkF_{ST_A}}$, $name_{ST_A}$)	OK:{ID_{ST_A}, $enc(name_{rdF_{ST_A}})$, $ver_{rdF_{ST_A}}$, $ver_{gkF_{ST_A}}$}

Beobachtete Anfrage	**Beobachtete Antwort**	**Anfrage in Sequezdiagrammen**	**Antwort in Sequezdiagrammen**
downloadfile/*value*/*value*?version=0&etag=	*value*	DownloadFile(ID_{ST_A}, $enc(name_{rdF_{ST_A}})$, $ver_{rdF_{ST_A}} - 1$)	OK:{rdF_{ST_A}}
downloadfilehead/*value*/*value*?version=0&etag=&length=16384	*value*	DownloadFileHead(ID_{ST_A}, $enc("start.pdf")$, $size_{start.pdf}$)	OK:{$enc(start.pdf_{0...length})$}
downloadroamingprofile?version=0&etag=*value*	*HTTP Status Code: 304*	DownloadRoamingProfile(ver_{rpF_A})	NOT MODIFIED
getaccountinfo	{"ActualBytes":0,"MaxBytes":*value*,"ActualShares":0,"MaxShares":3,"MaxShareMembers":10,"MaxDownloadSpeed":*value*,"MaxUploadSpeed":*value*,"OriginalDownloadSpeed":*value*,"OriginalUploadSpeed":*value*,"MaxFileSize":*value*,"MaxTresorSize":0,"PreviousQuotaResetTimeDelta":10,"NextQuotaResetTimeDelta":*value*,"ActualDownloadTraffic":0,"ActualUploadTraffic":0,"MaximumTraffic":*value*,"IsRmsEnabled":false,"MaxNumberOfClients":3,"MaxPreviousFileVersions":0,"MaxHistoryTimeInterval":*value*,"SubscriptionInfo":"free","UserLanguage":	GetAccountInfo()	OK:{$info_A$}

Beobachtete Anfrage	Beobachtete Antwort	Anfrage in Sequezdiagrammen	Antwort in Sequezdiagrammen
	{"LanguageCode":"en-US", "ReadableForm":"*value*"}, "UserId":"","DomainId":"", "DomainName":"", "InviteFromFirstName":"", "InviteFromLastName":"", "InviteFromEmail":"", "MembershipId":"", "Role":"","Status":"", "TargetGroupName":"", "MaxLinkShareExpirationTime": *value*, "MaxLinkShareFileSize": *value*, "ActualLinkShareTraffic":0, "MaxLinkShareTraffic":*value*, "ActualLinkShareCount":0, "MaxLinkShareCount":10, "ActualLinkShareDownload Count":0, "MaxLinkShareDownload Count":100, "MaxLinkShareDownload CountPerLink":20, "IsPasswordProtectedLink Enabled":false, "GamificationSteps":[...], "GamificationBonusSize": *value*,		

Beobachtete Anfrage	**Beobachtete Antwort**	**Anfrage in Sequezdiagrammen**	**Antwort in Sequezdiagrammen**
	"IsAllowedToCreateEncrypted Links":true, "IsAllowedToCreateTresors": true, "CanInviteOtherThanManager": false, "IsAllowedToSyncTresors":true, "TresorSharingMode":"*value*"}		
getagreecertificate? emailaddress=*value*& containerid=&certificateid=	{"AgreeCert":"*value*", "IntermediateCert":"*value*"}	GetAgreeCert($mail_A, ID_{ShT}$)	OK:{$agree_{cert_A}$}
getallinvitationlinks? containerid=*value*& lastchangestamp=0&linkstatus= active	*HTTP Status Code: 304*	GetAllInvitationLinks(ID_{ShT}, ver_{ShT})	NOT MODIFIED
getanonymouscertificate? emailaddress=*value*& certificateid=	{"AnonymCert":"*value*", "IntermediateCert":"*value*"}	GetAnonymousCertificate($mail_B$)	OK:{$anon_{cert_B}$}
getavailablelocalizationdata	[{"LanguageCode":"en-US", "ReadableForm":"*value*"}, {"LanguageCode":"de", "ReadableForm":"*value*"}]	GetAvailableLocalizationData()	OK:{$language$}
getchallenge?email=*value*& clientnonce=*value*	{"ServerNonce":"*value*", "ClientSalt":"*value*"}	GetChallange($mail_A, nonce_A$)	OK:{$nonce_{server}, salt_{pwd_A}$}
getcontainerchanges: [{"ContainerId":"*value*", "LastChangestamp":0}]	{"AccountInfo":{"ActualBytes": 0,"MaxBytes":*value*, "ActualShares":1,"MaxShares":3, "MaxShareMembers":10, "MaxDownloadSpeed":*value*, "MaxUploadSpeed":*value*,	GetContainerChanges($tresors$: [{ID_{ST_A}, ver_{ST_A}}])	OK:{$info_A, tresors$: [{$ID_{ST_A}, ver_{ST_A}+1$, $changedFiles$: [{$enc(name_{rdF_{ST_A}}), ver_{rdF_{ST_A}}$, $"Create"_{rdF_{ST_A}}$}], $off_{RMS_{ST_A}}$, $ver_{perm_{ST_A}}$}]}

Beobachtete Anfrage	Beobachtete Antwort	Anfrage in Sequezdiagrammen	Antwort in Sequezdiagrammen
	"OriginalDownloadSpeed": *value*, "OriginalUploadSpeed": *value*,"MaxFileSize":*value*, "MaxTresorSize":0, "PreviousQuotaResetTimeDelta": 41,"NextQuotaResetTimeDelta": *value*, "ActualDownloadTraffic":0, "ActualUploadTraffic":0, "MaximumTraffic":*value*, "IsRmsEnabled":false, "MaxNumberOfClients":3, "MaxPreviousFileVersions":0, "MaxHistoryTimeInterval": *value*, "SubscriptionInfo":"free"}, "ChangedContainers": [{"ContainerId":"*value*", "LastChangestamp":1, "ContainerSize":235, "IsOutOfDate":false, "ChangedFiles": [{"FileName":"*value*", "ETag":"*value*","Version":1, "LastOperation":"Create"}], "LastPermissionChangestamp":1, "IsRmsEnabled":false, "Attributes":0,"Metadata":"", "LastReturnedChangestamp":1, "Error":null}]}		

Beobachtete Anfrage	**Beobachtete Antwort**	**Anfrage in Sequezdiagrammen**	**Antwort in Sequezdiagrammen**
getdevicecert?authsessionid=*value*&expirationinseconds=0: {"EncCsr":"*value*","SignCsr":"*value*","SslCsr":"*value*"}	{"EncDeviceCert":"*value*", "SslDeviceCert":"*value*", "SignDeviceCert":"*value*", "IntermediateCert":"*value*", "RoamingProfileETag":null}	GetDeviceCert($ID_{authSession}$, $enc^{request}_{dev1_{cert_A}}$, $sign^{request}_{dev1_{cert_A}}$, $ssl^{request}_{dev1_{cert_A}}$)	OK:{$enc_{dev1_{cert_A}}$, $sign_{dev1_{cert_A}}$, $ssl_{dev1_{cert_A}}$}
getlatestgroupkeyfile/*value*?version=0	{"GkfChangeStamp":1, "ContainerId":"*value*", "AgreeCertChangeStamp":1, "ChangedBy":"*value*", "ChangedFromIp":"*value*", "GkfFileName":"*value*", "GkfETag":"*value*", "GroupKeyDirty":"Clean", "GroupKeyFileInBase64":"*value*","AgreeCertUniqueId":"*value*"}	GetLatetsGroupKeyFile(ID_{ST_A}, $ver_{gkF_{ST_A}}$)	OK:{ID_{ST_A}, $ver_{gkF_{ST_A}}$, $pairID_{cert_A}$, $changedBy_{gkF_{ST_A}}$, $name_{gkF_{ST_A}}$, gkF_{ST_A}}
getlogintoken?authsessionid=*value*&validityinseconds=0	{"LoginToken":"*value*", "Validity":43200}	GetLoginToken($ID_{authSession}$, $val_{authSession}$)	OK:{$token_{session}$, $val_{token_{session}}$}
getmessages	[]	GetMessages()	OK:{ }
getmytresors	[{"ContainerId":"*value*", "TresorName":"*value*", "Metadata":"", "ContainerAttributes":0, "HasAccess":true, "CommittedBytes":0, "TresorMembershipState":"Member",	GetMyTresors()	OK:{$tresors_A$: [{ID_{ST_A}, $name_{ST_A}$, $enc(name_{rdF_{ST_A}})$, $off_{RMS_{ST_A}}$, $A_{perm_{ST_A}}$, $ver_{perm_{ST_A}}$, $ver_{gkF_{ST_A}}$}]}

Beobachtete Anfrage	**Beobachtete Antwort**	**Anfrage in Sequezdiagrammen**	**Antwort in Sequezdiagrammen**
	"RootDirFileName": "*value*","IsRmsEnabled":false, "Permission":4, "PermissionChangeStamp":1, "InvitationLinkChangeStamp":0, "EncryptedLinkChangeStamp":0, "GkfVersion":1,"CommitId":0, "TresorAlias":""}]		
publishinglicenses: {"AllowAuditedExtraction":false, "Policy":{ "EncryptedApplicationData": {"MSOTPLDESC":"*value*", "MSOTPLNAME":"*value*", "NOLICCACHE":"1"}, "UserRights":[{"Rights": ["EDITRIGHTSDATA","VIEW", "EDIT","EXTRACT", "EXPORT","PRINT", "OBJMODEL","DOCEDIT"], "Users":["*value*"]}, {"Rights": ["VIEWRIGHTSDATA", "VIEW","EDIT","OBJMODEL", "DOCEDIT"],"Users": ["*value*"]},{"Rights": ["VIEWRIGHTSDATA", "VIEW","OBJMODEL"], "Users":["*value*"]}]}, "PreferDeprecatedAlgorithms": true,"ReferralInfo":"*value*"}	{"SerializedPublishingLicense": "*value*","Id":"*value*", "Name":null,"Description":null, "Referrer":"*value*","Owner": "*value*","Key":{"Value": "*value*","CipherMode": "*value*" ,"Algorithm":"AES", "Size":16},"ContentId": "*value*"}	GetPublishing License$_{accessToken_{RMS}}(mail_A, ID_{RMS_{RShT}}, perm_{RMS_{RShT}}$: $[\{mail_{managers_{RMS_{RShT}}}, perm_{managers_{RMS_{RShT}}}\}, \{mail_{editors_{RMS_{RShT}}}, perm_{editors_{RMS_{RShT}}}\}, \{mail_{readers_{RMS_{RShT}}}, perm_{readers_{RMS_{RShT}}}\}],)$	OK:$\{PL_{docu.docx}, mail_A, ID_{RMS_{RShT}}, mail_{RMS_A}, key_{docu.docx}, cipher_{key_{docu.docx}}, ID_{RMS_{docu.docx}}\}$

Beobachtete Anfrage	**Beobachtete Antwort**	**Anfrage in Sequezdiagrammen**	**Antwort in Sequezdiagrammen**
getrmscredentials	{"RmsEmail":"*value*", "RmsPassword":"*value*"}	GetRmsCredentials()	OK:{$mail_{RMS_A}, pwd_{RMS_A}$}
getrmsgroupnames/*value*	"*value*"	GetRmsGroupNames(ID_{RShT})	OK:{$ID_{RMS_{RShT}}$}
getrmsparameters	{"TenantHost":"*value*", "RmsServerHost":"*value*", "EdrAddinUrl":"*value*"}	GetRmsParameters()	OK:{$tenantHost$, $rmsServerHost$, $adrAddinUrl$}
getrmsuserpermissions/*value*	[{"TresoritEmail":"*value*", "RmsRight":3}]	GetRmsUserPermissions(ID_{RShT})	OK:{$perm_{RMS_{RShT}}$: [{$mail_A$}]}
getroamingprofile?email=*value* &authsessionid=*value*	{"RoamingProfile":"*value*", "UserCert":"*value*", "IntermediateCert":"*value*", "UserCertFragment":null}	GetRoamingProfile($mail_A$, $ID_{authSession}$)	OK:{$rpF_A, cert_A$}
gettresorinfo/*value*	{"ChangeStamp":3,"ActualSize": 456679,"IsRmsEnabled":false, "Attributes":0,"Metadata":"", "PermissionChangestamp":1, "GkfChangeStamp":1, "InvitationLinkChangeStamp":0, "EncryptedLinkChangeStamp":0, "RootDirFileName":"*value*", "ContainerName":"*value*", "AgreeCertUniqueId":"*value*", "TresorMembershipState": "Member","Permission":4, "FirstName":"Franz", "LastName":"*value*", "InviterEmail":null, "InviterFirstName":null, "InviterLastName":null, "InviteDateTimeDelta":309,	GetTresorInfo(ID_{ST_A})	OK:{$name_{ST_A}, ver_{ST_A}$, $off_{RMS_{ST_A}}, A_{perm_{ST_A}}$, $ver_{perm_{ST_A}}, ver_{gkF_{ST_A}}$, $enc(name_{rdF_{ST_A}}), pairID_{cert_A}$, $mail_A, fragment_{ST_A}$}

Beobachtete Anfrage	**Beobachtete Antwort**	**Anfrage in Sequezdiagrammen**	**Antwort in Sequezdiagrammen**
	"InviteExpirationDateTime Delta":-309,"InviteMessage":null, "TresorFragment":"*value*"}		
gettresormembers/*value*? includeallstates=false& permissionchangestamp=0	{"PermissionChangeStamp":1, "TresorMembers": [{"UserEmail":"*value*", "TresorMembershipState": "Member","ContainerId":" *value*","Permission":4, "FirstName":"Franz", "LastName":"*value*", "InviterEmail":null, "InviteDateTimeDelta":309, "InviteExpirationDateTime Delta":-309}]}	GetTresorMembers(ID_{ST_A}, $ver_{perm_{ST_A}}$)	OK:{$ver_{perm_{ST_A}}, users_{ST_A}$: [{$mail_A$}]}
inviteuser: {"LatestVersion":2, "LatestETag":"*value*"}	{"AgreeCertUniqueId": "*value*","ContainerId": "*value*", "ExpirationTimeDeltaIn Seconds":*value*, "GroupKeyFileUpdateRequest": {"GroupKeyFileInBase64": "*value*","KeyVersion":2, "OldETag":"*value*", "OldVersion":1}, "InviteMessage":"","Permission": 3,"UserEmail":"*value*"}	InviteUser($pairID_{cert_B}, ID_{ShT}$, $val_{B_{invitation_A}}$, $gkF'_{ShT}, ver_{gkF_{ShT}} + 1$, $B_{perm_{ShT}}, mail_B$)	OK:{$ver_{gkF_{ShT}} + 1$}
jointresor/*value*: "*value*"	*HTTP Status Code 200*	JoinTresor($ID_{ShT}, secret$)	OK
login: {"LoginToken":"*value*", "Validity":0}	{"SessionId":"*value*", "Validity":1800}	Login($token_{session}$, $val_{token_{session}}$)	OK:{$ID_{session}, val_{session}$}

Beobachtete Anfrage	**Beobachtete Antwort**	**Anfrage in Sequezdiagrammen**	**Antwort in Sequezdiagrammen**
register?registrationid=&refid=: {"ClientSalt":"*value*","Csr": "*value*","Language": {"LanguageCode":"en-US"}, "RoamingProfile":"*value*", "SaltedPassword":"*value*", "TutorialGamificationKeys":[...]}	*HTTP Status Code 200*	Register($salt_{pwd_A}$, $cert_A^{request}$, rpF_A, $authSecret_A$)	OK
registeragreecertificate: {"AgreeCsr":"*value*", "AnonymCsr":"*value*"}	{"CertPairId":"*value*", "AnonymCert":"*value*", "AgreeCert":"*value*", "IntermediateCert":"*value*"}	RegisterAgreeCert($agree_{cert_A}^{reguest}$, $anon_{cert_A}^{reguest}$)	OK:{$pairID_{cert_A}$, $agree_{cert_A}$, $anon_{cert_A}$}
removeuser: {"ContainerId":"*value*", "GroupKeyFileUpdateRequest": {"GroupKeyFileInBase64": "*value*","KeyVersion":3, "OldETag":"*value*", "OldVersion":2}, "UserEmail":"*value*"}	{"LatestVersion":3, "LatestETag":"*value*"}	RemoveUser(ID_{ShT}, gkF_{ShT}, $ver_{gkF_{ShT}} + 1$, $mail_B$)	OK:{$ver_{gkF_{ShT}} + 1$}
setaslatestagreecertificate? certificateid=*value*	*HTTP Status Code 200*	SetAsLatestAgreeCert($pairID_{cert_A}$)	OK
setrmsenabled/*value*? rmsEnabled=true	*HTTP Status Code 200*	SetRmsEnabled(ID_{RShT}, $on_{RMS_{RShT}}$)	OK
settresorfragment/*value*? lastpermissionchangestamp=1: "*value*"	*HTTP Status Code 200*	SetTresorFragment(ID_{ST_A}, $ver_{perm_{ST_A}}$, $fragment_{ST_A}$)	OK
uploadfile/*value*/*value*? etag=*value*&filesize=164& rootdirkeyversion=1: *file*	{"ETag":"*value*", "Version":2, "ContainerChangeStamp":2, "ContainerActualSize":400}	UploadFile(ID_{ST_A}, $enc(name_{rdF_{ST_A}})$, $ver_{gkF_{ST_A}}$, rdF_{ST_A})	OK:{$ver_{rdF_{ST_A}}$, ver_{ST_A}}
uploadroamingprofile?etag= *value*: *file*	*HTTP Status Code 200*	UploadRoamingProfile(rpF_A)	OK