Analyse der Ende-zu-Ende-Verschlüsselung von Nextcloud

Emmanuel Syrmoudis
Betreuer: Dr. Holger Kinkelin
Seminar Future Internet SS2018
Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: emmanuel.syrmoudis@tum.de

KURZFASSUNG

Diese Seminararbeit untersucht die Ende-zu-Ende-Verschlüsselung (E2EE) der Filehosting-Software Nextcloud. Nextcloud ermöglicht das Selfhosting von Daten und damit Unabhängigkeit von Filehosting-Anbietern wie Dropbox. Die E2EE soll dabei sicherstellen, dass vertrauliche Daten nur auf Endgeräten entschlüsselt werden können. Im Mittelpunkt der Arbeit steht die Sicherheit der E2EE. Zu ihrer Analyse wird zunächst die Implementierung sowie algebraische, logische, Tool-basierte und informelle Analysemethoden vorgestellt und anschließend eine informelle Analyse der E2EE durchgeführt. Dabei wird insbesondere eine Schwachstelle beim Schlüsselaustausch aufgezeigt, mit der die Schutzziele Vertraulichkeit, Integrität und Authentizität verletzt werden können. Abschließend werden mögliche Verbesserungsvorschläge gemacht.

Schlüsselworte

Nextcloud, Ende-zu-Ende-Verschlüsselung, Cloud, Filehosting, Selfhosting

1. EINLEITUNG

Die moderne Gesellschaft wird immer digitaler, Menschen auf der ganzen Welt vernetzen sich über das Internet und stellen ihre Daten in die „Cloud“. Urlaubsphotos landen auf Facebook oder Instagram, Gedanken werden auf Twitter oder Snapchat geteilt, Termine über Doodle vereinbart, E-Mails über GMail verschickt und Dokumente auf Dropbox gespeichert. Auch in die Geschäftswelt und in den Bildungssektor hat die Digitalisierung Einzug gehalten, so wird beispielsweise Slack zur Kommunikation innerhalb von Arbeitsgruppen genutzt und offizielle Diskussionsforen zu manchen Kursen der TUM werden auf Piazza eingerichtet.

All diese Beispiele haben eine Gemeinsamkeit: Die Daten der Nutzer dieser Dienste werden auf den Servern des jeweiligen Anbieters gespeichert. Dort erwecken sie natürlich Begehrlichkeiten von mehreren Seiten. So könnten Staaten und deren Geheimdienste Anspruch auf diese Daten erheben, sei es zur Terrorbekämpfung oder zur Bewertung von Personen im chinesischen Sozialkredit-System [13]. Auch für Werbekunden sind die Daten interessant, ermöglichen sie doch eine zielgerichtetere Ausspielung von Werbeanzeigen. Gerade für Firmen wie Facebook und Google, die einen Großteil ihrer Dienste kostenlos anbieten, ist dies ein wesentlicher Bestandteil des Geschäftsmodells.

Da die Datennutzung für den einzelnen Nutzer meist keine unmittelbar spürbaren negativen Konsequenzen hat, wird diese oft toleriert. Dass sie aber eine Auswirkung auf die gesamte Gesellschaft haben kann, zeigte sich beispielsweise im US-amerikanischen Präsidentschaftswahlkampf 2016. Damals hatte die Datenanalysefirma Cambridge Analytica, im Auftrag der Kampagne des republikanischen Bewerbers Donald Trump, Werbung auf Facebook geschaltet. Hierbei nutzte sie detaillierte Daten von ca. 50 Millionen Facebook-Nutzern um als potentielle Wähler identifizierte Nutzer zu mobilisieren und Nutzer, die eher den Demokraten zugeneigt waren, zu demobilisieren [5]. Schlussendlich gewann der republikanische Kandidat.

Beispiele wie dieses zeigen, wie wichtig es ist, persönliche Daten nicht einfach unüberlegt fremden Anbietern zu überlassen. Eine Alternative hierzu ist *Selfhosting*, das Speichern der Daten auf eigenen Servern statt bei fremden Anbietern. Diese Seminararbeit befasst sich mit Nextcloud, einer Selfhosting-Alternative zu Filehosting-Anbietern wie Dropbox oder Onedrive. Nextcloud kann auf eigenen Servern installiert werden und ermöglicht unter anderem das Hochladen von Dateien, das Synchronisieren zwischen mehreren Geräten und das Teilen mit anderen Nutzern.

Statt auf eigenen Servern, bei denen man vollen und exklusiven Zugriff auf die Hardware hat, kann Nextcloud auch auf angemieteten Servern in einem fremden Rechenzentrum betrieben werden. Dies ist vor allem für kleinere Unternehmen interessant, hat aber wiederum den Nachteil, dass die Daten in fremde Hände gelangen. Hier schafft die Ende-zu-Ende-Verschlüsselung (E2EE) Abhilfe, die Nextcloud in Version 13 optional eingeführt hat [4] und die Hauptthema dieser Arbeit ist. Inhalte aus Ordnern bei denen die E2EE aktiv ist, können nur von berechtigten Nutzern eingesehen werden, nicht aber vom Serverbetreiber oder sonstigen Dritten. Die Ver- und Entschlüsselung der Daten erfolgt direkt auf den Endgeräten, wo auch die nötigen Schlüssel gespeichert sind.

Damit unterscheidet sich die E2EE auch von der ebenfalls von Nextcloud angebotenen serverseitigen Verschlüsselung. Bei dieser werden die Dateien auf dem Server ver- und entschlüsselt, der eingesetzte Schlüssel wird ebenfalls auf dem Server gespeichert. Das ist sinnvoll, wenn externe Speicher eingebunden werden, bietet aber keinen Schutz der Daten vor dem Betreiber des Servers. Bei der serverseitigen Verschlüsselung muss der Server also vertrauenswürdig sein, eine wirksame Ende-zu-Ende-Verschlüsselung setzt dieses Ver-

doi: 10.2313/NET-2018-11-1_10

trauen nicht voraus.

Die Ende-zu-Ende-Verschlüsselung ist auch dann sinnvoll, wenn man Nextcloud auf eigenen Servern betreibt und sensible Daten, wie etwa Firmengeheimnisse, vor einer Kompromittierung des Servers durch etwaige Sicherheitslücken schützen will.

Ziel dieser Seminararbeit ist es, eine Bewertung der Sicherheit der Nextcloud-E2EE abzugeben. Hierfür werden in Kapitel 2 zunächst zentrale Elemente der Implementierung und deren Funktionsweise vorgestellt. Anschließend werden in Kapitel 3.1 ein Angreifermodell und Schutzziele aufgeführt, anhand derer eine formalere Einordnung der Qualität von kryptographischen Protokollen möglich ist. Mehrere Methoden, mit denen kryptographische Protokolle analysiert werden können, präsentiert Kapitel 3.2. Kapitel 4 widmet sich dann der konkreten Analyse der Nextcloud-E2EE, mögliche Verbesserungsvorschläge zeigt Kapitel 5 auf. Abschließend verweist Kapitel 6 auf alternative Softwarelösungen; Kapitel 7 fasst die Ergebnisse der Arbeit zusammen.

2. IMPLEMENTIERUNG DER NEXTCLOUD-E2EE

Dieser Abschnitt beschreibt die zentralen Elemente der Ende-zu-Ende-Verschlüsselung von Nextcloud. Grundlage hierfür sind die Ausführungen im von Nextcloud veröffentlichten Whitepaper in der Version vom 20. September 2017 [4].

Die Nextcloud-E2EE stützt sich auf zwei zentrale Komponenten: Public-Key-Kryptographie bei der der Server als Zertifizierungsstelle fungiert und eine Metadaten-Datei, die für die Verschlüsselung auf Orderebene eingesetzt wird.

2.1 Public-Key-Kryptographie

Da das E2EE-Protokoll es ermöglichen soll, Dateien mit anderen Nutzern zu teilen, setzt Nextcloud Public-Key-Kryptographie ein. Damit ist es im Gegensatz zu symmetrischer Kryptographie möglich, dass Nutzerin Alice Daten verschlüsselt, die Nutzer Bob mit seinem Schlüssel entschlüsseln kann, ohne dass Alice und Bob vorher ein gemeinsames Geheimnis festlegen müssen. Alice verschlüsselt die Daten mit Bobs öffentlichem Schlüssel, Bob entschlüsselt sie anschließend mit seinem privaten Schlüssel.

Verwendet Alice erstmals die Nextcloud-E2EE, so erzeugt die Nextcloud-Anwendung auf ihrem Gerät ein Public-Key-Paar. Zudem erzeugt die Anwendung ein Passwort, mit dem der *private key* verschlüsselt wird. Dieses Passwort hat die Form eines *mnemonics*, mit einer Länge von 12 Worten, die aus einer Liste von 2048 Worten zufällig ausgewählt werden. Es hat damit eine Entropie von 132 bit und könnte beispielsweise so aussehen: „ghost eight waste vicious copper dizzy lonely bench lava slab split forward“.

Wie Abbildung 1 veranschaulicht, wird der mit dem mnemonic als Passwort unter Verwendung des AES-Algorithmus verschlüsselte *private key* auf den Server hochgeladen. Sobald Alice Nextcloud auf einem anderen Gerät verwendet, lädt dieses den verschlüsselten *private key* herunter, fragt Alice nach dem mnemonic und speichert den entschlüsselten *private key* anschließend auf dem Gerät.

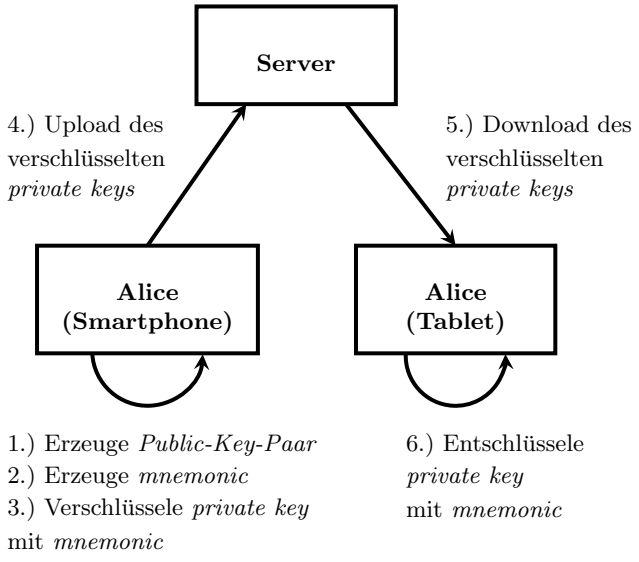

Abbildung 1: Austausch des *private keys* zwischen mehreren Endgeräten

Damit Alice ihren Ordner mit Bob teilen kann, benötigt sie Bobs *public key*. Um die öffentlichen Schlüssel zu verteilen, bietet der Server diese zum Download an und nimmt zusätzlich die Funktion einer Zertifizierungsstelle (CA) ein. Dieser Vorgang wird in Abbildung 2 dargestellt. Nachdem Alice ihr Schlüsselpaar erzeugt hat, erstellt sie einen Certificate Signing Request (CSR). Der CSR enthält Alice' *public key* und ihre User ID. Sie sendet den CSR an den Server, der dann ein X.509-Zertifikat [9] ausstellt.

Um ihren Ordner mit Bob zu teilen, lädt sie also dessen Zertifikat vom Server herunter, prüft die Signatur des Zertifikats und verwendet ab jetzt Bobs *public key*. Dabei vertraut sie darauf, dass der Server zum Zeitpunkt des Downloads des Zertifikats nicht kompromittiert ist.

2.2 Metadaten-Datei

Nachdem E2EE vom Serveradministrator aktiviert wurde, können die Nutzer auf Orderebene entscheiden, ob der komplette Ordner verschlüsselt werden soll. Entscheidet sich Alice für die Verschlüsselung eines Ordners, wird darin eine Metadaten-Datei im JSON-Format angelegt, in der die verschlüsselten Dateien verwaltet werden. Abbildung 3 zeigt den schematischen Aufbau dieser Datei, Tabelle 1 gibt einen Überblick über die verwendeten Schlüssel.

Der Kopf der Metadaten-Datei enthält eine Liste von *metadataKeys*. Der erste *metadataKey* wird beim Anlegen der Datei erzeugt und mit dem *public key* von Alice verschlüsselt. Hierbei wird der RSA-Algorithmus verwendet. Teilt Alice den Ordner mit Bob und Charlie, so wird die Liste der *metadataKeys* zusätzlich auch mit den *public keys* von Bob und Charlie verschlüsselt und kann somit sowohl von Alice, von Bob, als auch von Charlie entschlüsselt werden. Entschließt sich Alice dazu, den Ordner nicht mehr mit Charlie zu teilen, erzeugt sie einen neuen *metadataKey*, hängt diesen an die Liste der *metadataKeys* an und verschlüsselt die

doi: 10.2313/NET-2018-11-1_10

Tabelle 1: Übersicht der eingesetzten Schlüssel und Passwörter

Bezeichnung	Aufgabe	Algorithmus
encryptionKey	Verschlüsselung des Inhalts einer einzelnen Datei	AES/GCM/NoPadding
metadataKey	Verschlüsselung der geschützten Inhalte in der Metadaten-Datei	AES/GCM/NoPadding
Public-Key-Paar	Verschlüsselung der metadataKeys	RSA/ECB/OAEP+Padding
mnemonic	Passwort zur Verschlüsselung des private keys auf dem Server	AES/GCM/NoPadding

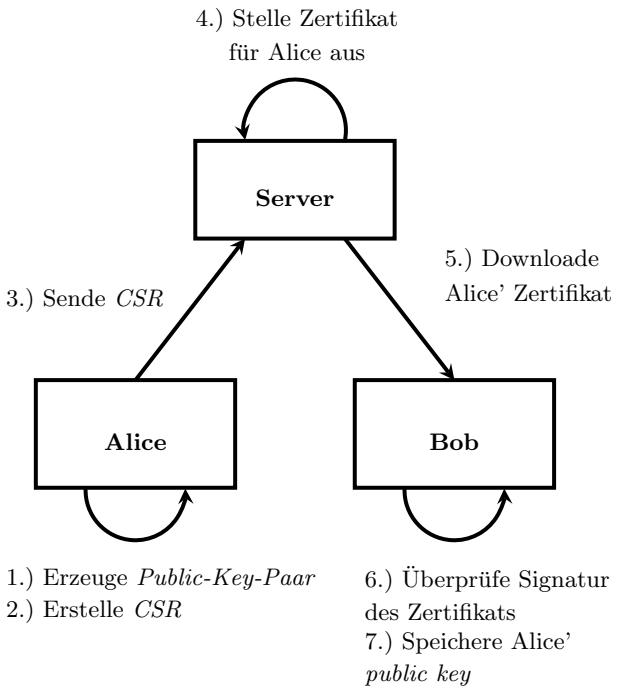

Abbildung 2: Austausch der *public keys* zwischen mehreren Nutzern

```
{
  "metadata": {
    // Liste der metadataKeys (RSA-verschluesselt)
    "metadataKeys": {
      "0": "aeltester metadataKey",
      "1": "neuester metadataKey"
    },
    "sharing": {
      // Public Keys aller berechtigten Nutzer (AES-verschluesselt mit neuestem metadataKey)
      "recipient": {
        "alice": "PUBLIC KEY",
        "bob": "PUBLIC KEY"
      },
    },
    "files": {
      // UUID der verschluesselten Datei
      "ioDuSxIfa...": {
        // verwendeter metadataKey
        "metadataKey": 1,
        // geschuetzte Daten (AES-verschluesselt mit angegebenem metadataKey)
        "encrypted": {
          // encryptionKey mit dem die Datei verschluesselt wurde
          "key": "...",
          "filename": "nuclearcodes.txt",
          "mimetype": "plain/text"
        },
        "initializationVector": "...",
        // Galois/Counter Mode Authentication Tag
        "authenticationTag": "..."
      }
      // analog dazu weitere Dateien...
    }
  }
}
```

Abbildung 3: Verkürzte Darstellung der Metadaten-Datei

Liste nur noch mit ihrem und Bobs *public key*.

Außer den *metadataKeys* enthält die Metadaten-Datei eine Liste der *public keys* aller berechtigten Nutzer (in obigem Beispiel Alice und Bob), sowie eine Liste der Dateien, die sich in dem Ordner befinden. Jede Datei wird durch eine zufällige Zeichenfolge (UUID) identifiziert. Ihr zugeordnet sind ein *encryptionKey*, der Dateiname, der MIME-Type, ein Initialisierungsvektor, ein Galois/Counter Mode Authentication Tag, sowie ein Verweis auf einen der *metadataKeys*. *encryptionKey*, Dateiname und MIME-Type sind dabei mit dem angegebenen *metadataKey* AES-verschlüsselt, die restlichen Elemente sind im Klartext einsehbar.

Bei der Verschlüsselung der einzelnen Dateien kommt AES als Blockchiffre mit Galois/Counter Mode (GCM) als Betriebsmodus zum Einsatz. GCM ermöglicht authentifizierte Verschlüsselung, wodurch sowohl Vertraulichkeit, als auch Integrität und Authentizität erreicht werden können [14]. Integrität und Authentizität können dabei mit Hilfe des GCM Authentication Tag überprüft werden.

Da RSA ausschließlich zur Verschlüsselung der *metadataKeys* mit den *public keys* verwendet wird und für alle anderen Verschlüsselungsvorgänge AES zum Einsatz kommt, wird die Verwendung teurer asymmetrischer Verschlüsselung auf ein Mindestmaß beschränkt und ansonsten effizient anwendbare symmetrische Verschlüsselung genutzt.

doi: 10.2313/NET-2018-11-1_10

Lesen einer Datei. Will ein berechtigter Nutzer, beispielsweise Bob, den Inhalt einer Datei lesen, benötigt er dazu den *encryptionKey*, mit dem der Inhalt dieser Datei verschlüsselt wurde. Um diesen zu erhalten, öffnet er zunächst die Metadaten-Datei. Nun kann Bob die Liste der *metadataKeys* mit seinem *private key* entschlüsseln. Im „files"-Abschnitt der Metadaten-Datei sucht er dann nach der gewünschten Datei.

Kennt er die UUID der Datei bereits, kann er einfach zur gewünschten Datei springen und dort in Erfahrung bringen, mit welchem *metadataKey* die geschützten Metadaten der Datei verschlüsselt wurden. Er entschlüsselt sie mit dem passenden *metadataKey* und kennt nun den *encryptionKey* mit dem der Inhalt der Datei verschlüsselt wurde. Diesen *encryptionKey* sowie den Initialisierungsvektor nutzt er schlussendlich um die Datei zu entschlüsseln. Zudem kann er mit dem GCM Authentication Tag Integrität und Authentizität der Datei prüfen. Falls er die UUID der Datei noch nicht kennt, erstellt er zunächst eine Auflistung aller Dateien, die sich im Ordner befinden indem er auf dieselbe Weise die Dateinamen aller Dateien entschlüsselt.

Schreiben einer Datei. Zum Schreiben einer Datei müssen analog zu oben zunächst die *metadataKeys* in Erfahrung gebracht werden. Die Nutzerin, die die Datei schreiben will, hier Alice, generiert dann einen zufälligen *encryptionKey*, sowie einen zufälligen Initialisierungsvektor und verschlüsselt die Datei mit AES unter Nutzung des Galois/Counter Mode. Anschließend lädt sie die verschlüsselte Datei auf den Server hoch und aktualisiert bzw. erstellt den entsprechenden Eintrag in der Metadaten-Datei. Zur Verschlüsselung des *encryptionKeys*, des Dateinamen und des MIME-Types nutzt sie den neuesten *metadataKey*.

3. GRUNDLAGEN UND ANALYSEMETHODEN

3.1 Angreifermodell und Schutzziele

Vor dem Entwurf oder der Analyse kryptographischer Protokolle oder Systeme ist es sinnvoll, einige formale Annahmen über die Fähigkeiten eines möglichen Angreifers zu treffen, sowie Ziele festzulegen, die das System erfüllen soll.

Ein weit verbreitetes Angreifermodell ist das *Dolev-Yao-Modell* [11]. In diesem Modell kontrolliert der Angreifer das Netzwerk und kann Nachrichten senden, empfangen, abfangen und modifizieren. Dabei kann er jegliche Teile einer Nachricht verändern, beispielsweise auch die Identität des Absenders. Zur Analyse der Nextcloud Ende-zu-Ende-Verschlüsselung bietet es sich an, das Angreifermodell um die Annahme zu erweitern, dass der Angreifer den Server kontrolliert. Somit kann er, analog zum Dolev-Yao-Modell, Dateien auf dem Server lesen, erstellen, löschen und verändern.

In der Regel unterscheidet man folgende Schutzziele [12]:

Vertraulichkeit Nur berechtigte Nutzer können den Inhalt einer Datei lesen. Im Kontext des eben festgelegten Angreifermodells wäre dieses Schutzziel beispielsweise verletzt, wenn der Angreifer über eine Sicherheitslücke an einen der eingesetzten Schlüssel gelangt und so den Dateiinhalt einer verschlüsselten Datei lesen kann.

Integrität Unberechtigten Nutzern ist es nicht möglich, unbemerkt den Inhalt einer Datei zu verändern. In unserem Angreifermodell hat der Angreifer die Möglichkeit Dateien zu verändern. Ist das Schutzziel erfüllt, können berechtigte Nutzer jedoch feststellen, dass eine unautorisierte Änderung vorgenommen wurde. Manchmal wird *Autorisierung* auch als separates Schutzziel betrachtet.

Authentizität Die Echtheit und Überprüfbarkeit einer Nachricht oder eines Dokuments ist sichergestellt. Nutzer können also überprüfen, ob eine Datei tatsächlich von einem berechtigten Nutzer bearbeitet wurde.

Verfügbarkeit Die angebotenen Dienste sind verfügbar und in ihrer Nutzbarkeit nicht eingeschränkt. Angriffe auf dieses Schutzziel werden als *Denial of Service* bezeichnet und beabsichtigen beispielsweise eine Nicht-Erreichbarkeit des Servers aus dem Internet.

Verbindlichkeit Es ist nicht möglich, durchgeführte Handlungen abzustreiten. Ändert Benutzerin Alice beispielsweise eine Datei, so kann sie gegenüber Benutzer Bob nicht glaubwürdig abstreiten, diese Änderung vorgenommen zu haben.

Privatheit Die Privatsphäre und die personenbezogenen Daten der Nutzer werden geschützt. Wie dieses Schutzziel umzusetzen ist, ist oftmals in Datenschutzgesetzen festgelegt.

Nextcloud selbst gibt an, dass die E2EE die Schutzziele Vertraulichkeit, sowie Authentizität und Integrität erfüllen soll [4], wobei Integrität und Authentizität hier einhergehen. Ein Angreifer soll also keine Möglichkeit haben, verschlüsselte Dateiinhalte im Klartext zu lesen und Dateien nicht unbemerkt verändern können.

Privatheit wird im Rahmen dieser Arbeit alleinig als die Privatheit der Daten der Nutzer aufgefasst und damit auch vom Schutzziel Vertraulichkeit abgedeckt. Um eine umfassende Privatheit zu gewährleisten, sind Maßnahmen erforderlich, die über den bloßen Einsatz einer E2EE hinausgehen, da hierbei beispielsweise auch der Umgang mit Metadaten berücksichtigt werden muss. Die übrigen Schutzziele liegen nicht im typischen Aufgabenbereich einer Ende-zu-Ende-Verschlüsselung und werden daher in dieser Arbeit nicht behandelt.

3.2 Analysemethoden

Nachdem ein Angreifermodell und die angestrebten Schutzziele festgelegt wurden, verbleibt die Auswahl einer passenden Analysemethode. Dieser Abschnitt stellt einige dieser Methoden kurz vor. Einen tieferen Einblick geben beispielsweise [15] oder [8].

3.2.1 Algebraische Analyse

Die aufwendigste, aber auch zielführendste Methode ist die algebraische Analyse. Hierbei wird das Protokoll oder System in ein formales algebraisches Modell überführt und die Eigenschaften dieses Modells mathematisch bewiesen.

doi: 10.2313/NET-2018-11-1_10

3.2.2 Modallogik

Besonders in den 1990er Jahren verbreitet, war die Anwendung von Modallogik zur Analyse von kryptographischen Protokollen. Die meistverwendete dieser Logiken ist die BAN-Logik [7]. Ausgehend von bestimmten Annahmen („Alice glaubt, dass nur Bob und Charlie Schlüssel X kennen“) und der gesendeten Nachrichten, werden dann wiederum neue Annahmen gebildet. Die BAN-Logik kann nur Aussagen zur Authentizität treffen. Mit der GNY-Logik besteht eine Weiterentwicklung, die allerdings deutlich komplizierter ist und deshalb kaum genutzt wird. [15]

3.2.3 Tool-basierte Analyse

Abgesehen von der „händischen“ Analyse, können kryptographische Protokolle auch mit der Hilfe von Tools analysiert werden. Zwei der bekanntesten Analysetools sind AVISPA [6] und Scyther [10]. Scyther beispielsweise definiert eine eigene Sprache, in der Protokolle modelliert werden können. Nachdem dies geschehen ist, analysiert Scyther dieses Modell und trifft Aussagen über die Erfüllung der Schutzziele, sowie über mögliche Angriffe.

3.2.4 Informelle Analyse

Deutlich weniger aufwendig als algebraische Analysen, dafür auch mit geringerer Aussagekraft, sind informelle Analysemethoden. Sie sind geeignet, um sich einen Überblick über mögliche Schwachstellen des Protokolls oder Systems zu verschaffen und Angriffsziele zu identifizieren. Auch wenn das untersuchte System zu komplex ist um es formal zu modellieren, können stattdessen informelle Methoden zur Analyse eingesetzt werden. Dazu können beispielsweise die Komponenten des Systems einzeln untersucht und Annahmen darüber getroffen werden, auf welche Weisen ein Angreifer eines der Schutzziele verletzen könnte.

Ein mögliches Hilfsmittel hierbei ist das Erstellen von „Attack Trees“ [16]. Damit kann modelliert werden, was zur erfolgreichen Ausführung eines Angriffs nötig ist und folglich eine Einschätzung der Erfolgswahrscheinlichkeit vorgenommen werden. Das Angriffsziel ist hierbei der Wurzelknoten und die möglichen Wege, das Ziel zu erreichen, dessen Kinder. Diese Kindknoten können selbst ebenfalls Kinder haben. Schneier [16] wählt als Beispiel das Öffnen eines Tresors. Dazu kann beispielsweise das Schloss geknackt werden oder die Kombination in Erfahrung gebracht werden. Um die Kombination herauszufinden, kann sie in schriftlicher Form gefunden werden oder dem Opfer selbst entlockt werden, ... Findet man schließlich einen realistischen Weg, so ist auch der Angriff als solcher durchführbar.

4. ANALYSE DER NEXTCLOUD-E2EE

In Abschnitt 3.1 wurde ein Angreifermodell definiert und die Schutzziele Vertraulichkeit, Authentizität und Integrität festgelegt, dieser Abschnitt untersucht nun, ob die eben vorgestellte Implementierung der Nextcloud E2EE diese Schutzziele erfüllt.

Das Angreifermodell besagt, dass der Angreifer sowohl das Netzwerk als auch den Server kontrolliert. Da, wie in Abschnitt 2 gesehen, der Server sowieso entweder Empfänger oder Sender aller Nachrichten ist (es findet keine direkte Kommunikation zwischen mehreren Nutzern oder den Endgeräten eines Nutzers statt), richtet sich das Augenmerk hier auf das Lesen, Schreiben, Löschen oder Modifizieren von Dateien auf dem Server durch den Angreifer. Zur Kommunikation über das Netzwerk sei hier noch angemerkt, dass alle Nachrichten über TLS gesendet werden [4].

Für die angestrebten Schutzziele gilt es nun, eine geeignete Analysemethode aus den in Abschnitt 3.2 vorgestellten Methoden auszuwählen. Da insbesondere auch das Schutzziel Vertraulichkeit von Interesse ist, ist die BAN-Logik hierfür nicht geeignet. Ideal wäre eine algebraische Analyse, die aber im Rahmen einer Seminararbeit nicht zu leisten ist. Deshalb beschränkt sich die Analyse auf eine informelle Untersuchung der Lese- und Schreibvorgänge und der Möglichkeiten, die sich daraus für den Angreifer ergeben.

Die Attack Trees in den Abbildungen 4 und 5 stellen Möglichkeiten dar, wie der Angreifer verschlüsselte Dateien lesen (Verletzung des Schutzziels Vertraulichkeit), bzw. schreiben (Verletzung der Schutzziele Integrität und Authentizität) kann. Da alle Schlüssel und Blockgrößen größer oder gleich 128 bit sind, kann Brute-Force generell als nicht möglich betrachtet werden. Die folgenden Abschnitte analysieren, ob der Angreifer anderweitige Möglichkeiten hat, an einen der Schlüssel zu gelangen.

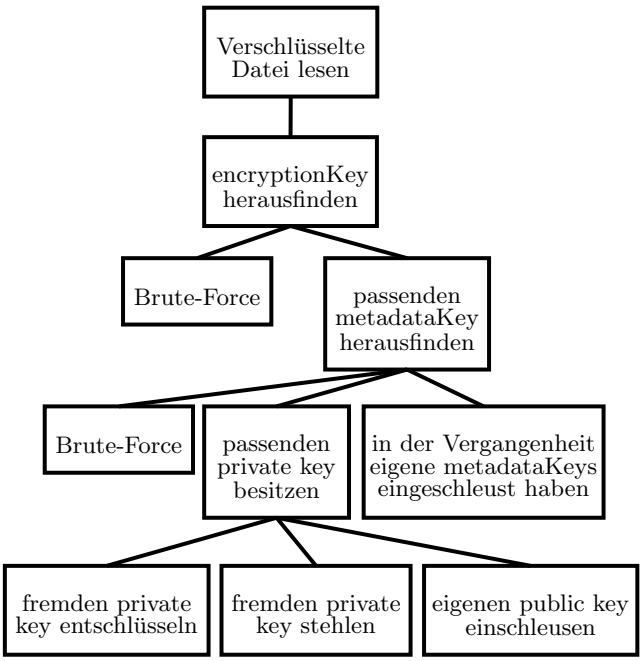

Abbildung 4: Attack Tree für das Lesen einer Datei

4.1 Schlüsselaustausch

Ein möglicher Angriffspunkt ist der in Abschnitt 2.1 und in den Abbildungen 1 und 2 beschriebene Schlüsselaustausch zwischen mehreren Endgeräten eines Nutzers bzw. zwischen mehreren Nutzern.

Betrachten wir zunächst den Fall der Übertragung des *private keys* auf ein weiteres Endgerät (Endgerät 2) eines Nutzers. Der *private key* wurde mit dem *mnemonic* als Passwort auf Endgerät 1 verschlüsselt (unter Verwendung von

doi: 10.2313/NET-2018-11-1_10

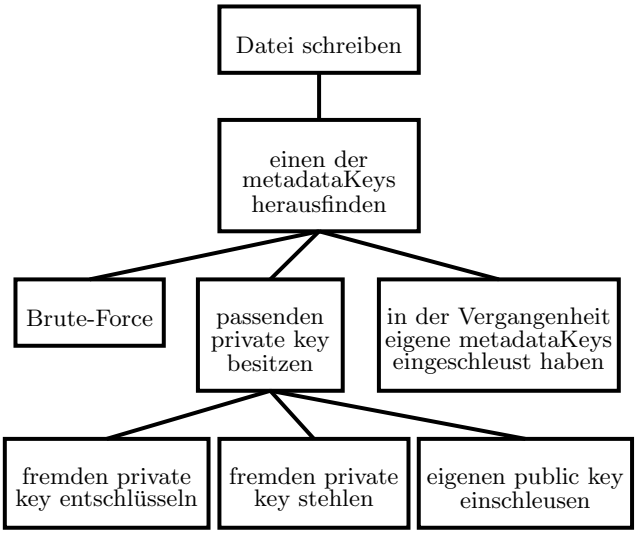

Abbildung 5: Attack Tree für das Schreiben einer Datei

AES-GCM) und anschließend auf dem Server gespeichert. Das *mnemonic* verbleibt ausschließlich beim Nutzer. Da das *mnemonic* eine Entropie von 132 bit hat ($12 \cdot 11$: 12 zufällig ausgewählte Wörter aus einer Liste von $2^{11} = 2048$ Wörtern) und die verwendete Blocklänge 128 bit beträgt, kann der Angreifer den *private key* nicht mittels Brute-Force im Klartext erlangen. Weiter hat der Angreifer die Möglichkeit, dem Endgerät 2 eine modifizierte Version des verschlüsselten *private keys* zu senden, beispielsweise einen *private keys*, den der Angreifer selbst erzeugt hat. Da zur Verschlüsselung GCM eingesetzt wurde, kann das Endgerät 2 die Verletzung der Integrität aber erkennen und den Schlüssel folglich ablehnen. Der Angreifer kann somit lediglich verhindern, dass das Endgerät 2 den *private key* erhält, was aber weder die Vertraulichkeit der Daten, noch deren Integrität oder Authentizität gefährdet.

Anders sieht es im Fall der Übertragung des *public keys* zu anderen Nutzern aus. Hier übernimmt der Server die Funktion einer CA, stellt also Zertifikate für einzelne Nutzer aus. Will Alice den *public key* von Bob abrufen, kann der Angreifer einfach ein Zertifikat mit seinem eigenen *public key*, aber der Identität von Bob erstellen. Teilt Alice dann einen Ordner mit Bob, kann der Angreifer die Liste der *metadataKeys* mit seinem *private key* entschlüsseln und hat folglich Zugriff auf alle Dateien in diesem Ordner.

Anzumerken ist, dass im Falle der Zertifikate „Trust On First Use (TOFU)“ gilt, das Zertifikat eines neuen Nutzers also nur einmal abgefragt und diesem dann vertraut wird. Ist ein Server zum Zeitpunkt der Abfrage vertrauenswürdig und wird erst später kompromittiert, hat der Angreifer hier keine Möglichkeit mehr, seinen eigenen *public key* einzuschleusen.

4.2 Dateioperationen

Betrachten wir nun verschiedene Dateioperationen. Der Angreifer ist in der Lage, Dateien zu lesen. Ist er nicht im Besitz des passenden *encryptionKeys*, kann er den Dateiinhalt nicht entschlüsseln, die Vertraulichkeit des Inhalts bleibt also gewahrt. Des weiteren ist der Angreifer in der Lage, neue Dateien anzulegen oder bestehende Dateien zu modifizieren. Den Inhalt der Datei, sowie Initialisierungsvektor und den *encryptionKey* kann er frei wählen und den GCM Authentication Tag selbst berechnen. Um insbesondere den GCM Authentication Tag in die Metadaten-Datei einzutragen, muss der Angreifer in Besitz mindestens eines *metadataKeys* sein. Ist dies nicht der Fall, so können Nutzer die Modifikation anhand des fehlerhaften Authentication Tags erkennen und die Schutzziele Integrität und Authentizität bleiben gewahrt.

Der Angreifer muss also in Besitz des *encryptionKeys* einer verschlüsselten Datei oder eines *metadataKeys* kommen, um eines der festgelegten Schutzziele verletzen zu können.

Im Besitz des *encryptionKeys* ist zunächst der Nutzer, der die Datei zuletzt gespeichert hat. Zudem ist der *encryptionKey* mit dem zum Speicherzeitpunkt neuesten *metadataKey* verschlüsselt in der Metadaten-Datei abgelegt. Dieser *metadataKey* ist mit den *public keys* aller zu diesem Zeitpunkt berechtigten Nutzer verschlüsselt in der Metadaten-Datei abgelegt. Zudem wird er beim Hinzufügen eines neuen Nutzers auch mit dessen *public key* verschlüsselt. Zugriff auf den Dateiinhalt haben also alle zum Speicherzeitpunkt berechtigten Nutzer sowie alle später hinzugefügten Nutzer. Andere Nutzer (wie der Angreifer) können keinen Zugriff auf den Dateiinhalt erlangen, das Schutzziel Vertraulichkeit ist erfüllt.

metadataKeys werden immer vom Besitzer des Ordners erzeugt. Außer dem Besitzer kennen Benutzer, die zu irgendeinem Zeitpunkt zur Menge der berechtigten Nutzer gehört haben, zumindest den ältesten *metadataKey*. Andere Nutzer haben keine Möglichkeit, Kenntnis eines *metadataKeys* zu erlangen. War der Angreifer nie in der Menge der berechtigten Nutzer, sind die Schutzziele Integrität und Authentizität erfüllt. Auch von einem Benutzer, der aus der Menge der berechtigten Nutzer entfernt wurde, würde man erwarten, dass er diese Schutzziele nicht verletzen kann. Dies ist aber nicht der Fall, da der Angreifer Einträge in der Metadaten-Datei auch mit einem alten *metadataKey* verschlüsseln kann.

4.3 Ersetzen der metadataKeys

Wie bereits in Abschnitt 2.2 beschrieben, werden die *metadataKeys* mit den *public keys* aller berechtigten Nutzer RSA-verschlüsselt in der Metadaten-Datei gespeichert. Da alle *public Keys* unverschlüsselt auf dem Server gespeichert werden (siehe Abschnitt 2.1), kennt diese auch der Angreifer. Zudem werden die Namen der Ordner und die Liste der Nutzer mit denen ein Ordner geteilt wurde unverschlüsselt auf dem Server gespeichert, da die Verwaltung der Ordner außerhalb der E2EE-App erfolgt [4].

Der Angreifer kennt also die berechtigten Nutzer sowie deren *public keys*. Er kann eigene *metadataKeys* erzeugen und diese mit den passenden *public keys* verschlüsseln. Anschließend kann er die ursprünglichen *metadataKeys* in der Metadaten-Datei durch die eben erzeugten ersetzen. Befinden sich bereits im Dateien im Ordner, werden diese durch das Ersetzen der *metadataKeys* unlesbar. Um seinen Angriff zu verschleiern, kann der Angreifer die vorhandenen Dateien löschen und eigene Dateien darin ablegen, die zum Namen des Ord-

doi: 10.2313/NET-2018-11-1_10

ners passen.

Bleibt der Angriff unentdeckt, kann der Angreifer fortan alle in diesem Ordner gespeicherten Dateien lesen und schreiben und damit die Schutzziele Vertraulichkeit, Integrität und Authentizität verletzen.

4.4 Einspielen alter Dateiversionen

In der Metadaten-Datei befindet sich kein Zeitstempel, der angibt, wann sie zuletzt bearbeitet wurde. Der Angreifer hat daher die Möglichkeit, Metadaten-Datei und Ordnerinhalt durch alte Versionen zu ersetzen, ohne dass dies von den Endgeräten erkannt werden kann.

5. VERBESSERUNGSVORSCHLÄGE

In Abschnitt 4 wurden mehrere Schwachstellen in der Ende-zu-Ende-Verschlüsselung von Nextcloud identifiziert. Dieser Abschnitt soll Verbesserungsvorschläge aufzeigen, mit denen sie beseitigt oder zumindest abgemildert werden können.

Das größte Problem in der Implementierung der Nextcloud-E2EE liegt im Schlüsselaustausch der *public keys*. Hier übernimmt der Server gleichzeitig die Funktion einer Zertifizierungsstelle und muss daher beim erstmaligen Teilen von Ordnern mit neuen Nutzern vertrauenswürdig sein.

Hier sollte man die Möglichkeit schaffen, optional andere CAs zu verwenden. Zur Überprüfung der Zertifikate könnte man dann beispielsweise den Stammspeicher des Betriebssystems verwenden. Des weiteren sollte man Nutzern ermöglichen, fremde *public keys* von Hand zu überprüfen. Dies könnte beispielsweise durch einen QR-Code geschehen, der in der Nextcloud-App angezeigt wird und den andere Nutzer dann persönlich mit ihrer Nextcloud-App scannen können. Auch die Anzeige eines Fingerabdrucks des Public Keys, der dann über andere Kanäle (z.B. telefonisch) abgeglichen werden kann, wäre denkbar.

Ein weiteres Problem liegt darin begründet, dass die Integrität und Authentizität der Metadaten-Datei nicht sichergestellt wird. Dadurch kann der Angreifer sowohl die *metadataKeys* durch selbst erstellte Schlüssel ersetzen, als auch alte *metadataKeys* zum erstellen und bearbeiten von Einträgen in der Metadaten-Datei nutzen.

Um das Ersetzen der *metadataKeys* erkennen zu können, sollte der Besitzer des Ordners die Liste der *metadataKeys* mit seinem *private key* signieren. Anhand dieser Signatur kann dann auf den Endgeräten geprüft werden, ob die *metadataKeys* wirklich vom Besitzer des Ordners erzeugt wurden.

Zudem sollten diejenigen Abschnitte der Metadaten-Datei, die von allen berechtigten Nutzern bearbeitet werden dürfen, mit dem neuesten *metadataKey* authentifiziert werden, beispielsweise in Form eines HMAC. Dieser HMAC wird nach jedem Bearbeiten der Metadaten-Datei neu berechnet und umfasst die Abschnitte „sharing“ und „files“. Damit wird auch verhindert, dass ein Angreifer, der ehemals zu den berechtigten Nutzern gehört hat, die Integrität und Authentizität der Dateien gefährden kann. Da er den neuesten *metadataKey* nicht kennt, kann er keinen passenden HMAC berechnen und die Nutzer die Verletzung der Integrität erkennen.

Um ein Wiedereinspielen alter Versionen des Ordners erkennen zu können, sollte ein Zeitstempel in der Metadaten-Datei gespeichert werden, der ebenfalls vom oben vorgeschlagenen HMAC authentifiziert wird. Der Zeitstempel wird immer dann aktualisiert, wenn die Datei bearbeitet wurde. Speichern die Nextcloud-Apps auf den Endgeräten den zuletzt gesehenen Zeitstempel, sowie den zu diesem Zeitpunkt neuesten *metadataKey*, können sie alte Versionen erkennen.

6. ÄHNLICHE PROTOKOLLE UND ALTERNATIVEN

Im Bereich der Ende-zu-Ende-Verschlüsselung bietet sich ein Vergleich der Nextcloud-E2EE zum Messaging-Protokoll Signal [2] an, welches unter anderem in der gleichnamigen Kommunikationssoftware, aber auch in Whatsapp oder Google Allo eingesetzt wird. Während der Anwendungszweck ein anderer ist, teilen Signal und die Nextcloud-E2EE die Problematik, dass die Kommunikation über einen zentralen Server läuft, aber nur auf den Endgeräten ver- und entschlüsselt werden soll und daher Public Keys zwischen Nutzern ausgetauscht werden müssen. Im Gegensatz zur Nextcloud-E2EE könnte ein Angreifer, der seinen eigenen Public Key einschleust allerdings nur zukünftige Kommunikation mitlesen, bereits gesendete Nachrichten können nicht nachträglich entschlüsselt werden. Die Software Signal bietet ähnlich zu den Vorschlägen in Abschnitt 5 die Möglichkeit Public Keys per QR-Code oder in Form einer „Safety Number“ zu verifizieren.

Als mögliche Filehosting-Alternativen zur Nextcloud-E2EE können Seafile [1] und Boxcryptor [3] betrachtet werden. Seafile ist ähnlich wie Nextcloud eine Filehosting-Software, die auf eigenen Servern betrieben werden kann. Auch Seafile bietet die Möglichkeit einer Ende-zu-Ende-Verschlüsselung. Im Gegensatz zu Nextcloud, können verschlüsselte Dateien aber nicht mit anderen Nutzern geteilt werden. Zudem schützt Seafile die Metadaten nicht, ein Angreifer könnte also beispielsweise die Dateinamen auslesen [1]. Seafile nutzt AES mit CBC als Betriebsmodus. Damit wird nur die Vertraulichkeit der Dateiinhalte sichergestellt, nicht aber Integrität und Authentizität, die auch nicht auf anderem Wege umgesetzt werden.

Boxcryptor ist eine Software, die mit bestehenden Filehosting-Diensten, wie z.B. Dropbox, genutzt werden kann. Ihr Einsatzzweck ist speziell die Ende-zu-Ende-Verschlüsselung von Ordner auf eben diesen Diensten. Die E2EE ist dabei ähnlich umgesetzt wie die E2EE von Nextcloud. Auch ein Teilen von einzelnen Dateien oder Ordnern mit anderen Nutzern ist möglich. Hier übernimmt der Anbieter von Boxcryptor die Verteilung der Public Keys und muss daher als vertrauenswürdig erachtet werden. Generell muss zur Nutzung von Boxcryptor dessen Entwickler vertraut werden, da Anwendungen anders als bei Nextcloud und Seafile nicht quelloffen zur Verfügung gestellt werden. Wie Seafile stellt auch Boxcryptor nur die Vertraulichkeit, nicht aber die Integrität und Authentizität des Dateiinhalts sicher. [3]

7. ZUSAMMENFASSUNG

doi: 10.2313/NET-2018-11-1_10

Zusammenfassend kann festgestellt werden, dass die Ende-zu-Ende-Verschlüsselung von Nextcloud prinzipiell sinnvoll konstruiert ist, aber aufgrund zweier Schwachstellen, die die Schutzziele Vertraulichkeit, Integrität und Authentizität gefährden, noch nicht für den alltäglichen Gebrauch geeignet ist.

Eine Schwachstelle betrifft das erstmalige Teilen von Ordnern mit einem anderen Nutzer. Da der Public Key des neuen Nutzers in Form eines Zertifikats vom Server heruntergeladen und auch von diesem signiert wird, muss der Server zu diesem Zeitpunkt vertrauenswürdig sein. Die zweite größere Schwachstelle resultiert aus einer fehlenden Authentizitätsprüfung in der Metadaten-Datei. Hier kann es einem Angreifer unter Umständen gelingen, eigenes Schlüsselmaterial einzuschleusen, welches die Nutzer dann zur Verschlüsselung der Dateien verwenden.

Werden die in dieser Arbeit beschriebenen Schwachstellen behoben, ist die einzige Information, die ein Angreifer über einen Ordner mit aktiver E2EE in Erfahrung bringen kann, die Anzahl der Dateien, die sich darin befinden und deren Größe. Die Nutzung von Nextcloud mit aktiver Ende-zu-Ende-Verschlüsselung ist dann eine gute Möglichkeit, vertrauliche Daten über mehrere Geräte hinweg zu synchronisieren und wo nötig auch mit anderen Nutzern zu teilen. Leider kommt die E2EE auch mit einigen Komforteinbußen, so ist es konstruktionsbedingt nicht möglich, sie im Webbrowser zu nutzen (dieser müsste Javascript-Code ausführen, der von einem kompromittierten Server modifiziert werden könnte) oder die Dateiversionierung zu nutzen [4]. Da die E2EE aber auf Ordnerebene aktiviert werden kann, besteht die Möglichkeit, jeweils zwischen dem Schutz der Daten und dem Nutzungskomfort abzuwägen.

Im Vergleich zu den Alternativen Seafile und Boxcryptor stellt die Verwendung der Ende-zu-Ende-Verschlüsselung von Nextcloud die beste der angeführten Lösungen dar um vertrauliche Daten in der Cloud zu schützen. Kommt ein Selfhosting der Daten nicht in Frage, kann auch der Einsatz von Boxcryptor in Betracht gezogen werden.

8. LITERATUR

[1] Security features. Seafile Server Manual. `https://manual.seafile.com/security/security_features.html`.

[2] Technical information. Signal. `https://signal.org/docs/`.

[3] Technical overview. Boxcryptor. `https://www.boxcryptor.com/en/technical-overview/`.

[4] End-to-end encryption design. Nextcloud, 20. September 2017. `https://nextcloud.com/endtoend/`.

[5] The antisocial network. The Economist, 22. März 2018.

[6] A. Armando et al. The avispa tool for the automated validation of internet security protocols and applications. In *International conference on computer aided verification*, pages 281–285. Springer, 2005.

[7] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, Feb. 1990.

[8] H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *Journal of Telecommunications and Information Technology*, pages 5–15, 2002.

[9] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 5280, May 2008. `https://tools.ietf.org/html/rfc5280`.

[10] C. J. Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *International Conference on Computer Aided Verification*, pages 414–418. Springer, 2008.

[11] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[12] C. Eckert. *IT-Sicherheit: Konzepte-Verfahren-Protokolle*. Walter de Gruyter, 2013.

[13] A. Landwehr. China schafft digitales Punktesystem für den „besseren“ Menschen. `https://heise.de/-3983746`.

[14] D. McGrew and J. Viega. The galois/counter mode of operation (gcm). *NIST Modes of Operation Process*, 20, 2004.

[15] C. A. Meadows. Formal verification of cryptographic protocols: A survey. In *International Conference on the Theory and Application of Cryptology*, pages 133–150. Springer, 1994.

[16] B. Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

doi: 10.2313/NET-2018-11-1_10