Universität Bremen

Entwurf und Implementierung einer Scyther-Spezifikation für das C3DC-Profil zur Autorisierung in Constrained Environments

Bachelorarbeit am
Technologie-Zentrum Informatik und Informationstechnik
Fachbereich 3
Universität Bremen

von

Dominik Sauer

Betreuer:

Dr.-Ing. Olaf Bergmann

Gutachter:

Dr.-Ing. Olaf Bergmann
Prof. Dr.-Ing. Carsten Bormann

Tag der Anmeldung: 03. Mai 2019
Tag der Abgabe: 06. September 2019

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Bremen, den 06. September 2019

Zusammenfassung

In dieser Arbeit soll das Constrained Client/Cross-Domain Capable Authorization Profile for ACE des ACE-OAuth-Frameworks auf mögliche Sicherheitslücken analysiert werden. Dieses Profil beschreibt, wie zwei eingeschränkte Geräte die notwendigen Schlüsselmaterialien erhalten, um geschützt miteinander kommunizieren zu können. Hierzu werden zwei weniger eingeschränkte Geräte benutzt, welche die eingeschränkten Geräte in dem Schlüsselaustausch unterstützen. Für diese Arbeit wird das Scyther-Werkzeug herangezogen, welches eine formale Analyse von Sicherheitsprotokollen vornehmen kann. Mittels des Scyther-Werkzeugs soll eine Testspezifikation für dieses Profil entworfen und implementiert werden. Diese wird in der formalen Sprache des Scyther-Werkzeugs definiert. Zunächst soll hierfür geprüft werden, ob eine bestehende Analyse als Grundlage verwendet werden kann, um die Analyse des Profils in dieser Arbeit darauf aufbauen zu können. Mittels der Analyse soll festgestellt werden, ob das Protokoll der Spezifikation entsprechend ausgeführt wird und der im Protokoll definierte Schlüsselaustausch geschützt stattfindet. Die Analyse des C3DC-Profils deutet daraufhin, dass der Schlüsselaustausch geschützt erfolgt und somit der Protokollspezifikation entspricht. Dennoch zeigen sich Schwächen im Protokoll hinsichtlich des Designs. Hierbei müssen Akteure nicht unbedingt am Protokollablauf teilnehmen. Dadurch liegt eine Verletzung der Sicherheitsziele vor. Entsprechend werden Lösungsansätze vorgestellt, die diese Schwächen beheben könnten.

Danksagung

An dieser Stelle möchte ich mich bei meiner Familie, sowie bei meinen Freunden und Kommilitonen für die Motiviationsförderung, für die gemeinsame Zeit während des Studiums, sowie für die Unterstützung bei der Korrektur dieser Arbeit bedanken.

Dann gilt mein Dank meinem Arbeitgeber, welcher mir das duale Studium an der Universität Bremen ermöglicht hat.

Auch gebührt mein Dank meinem Betreuer, der mir während der Bearbeitung der Bachelorthesis hilfreiche Anregungen bezüglich dem Aufbau der Bachelorthesis und der implementierten Testspezifikation gegeben hat.

Abschließend möchte ich mich beim Cognitive Systems Lab der Universität Bremen bedanken für die Erlaubnis der Verwendung ihrer Vorlage zur Bachelorthesis.

Dominik Sauer

Bremen, den 06. September 2019

Inhaltsverzeichnis

Abbildungsverzeichnis

Codedarstellungsverzeichnis

1 Einleitung

Bei der Kommunikation zweier Gesprächspartner im Internet kommt es nicht nur darauf an, dass gesendete Nachrichten empfangen werden, sondern auch darauf, ob dies gesichert geschieht. Unter Sicherheit wird verstanden, dass gewisse Sicherheitsziele, wie unter anderem die Vertraulichkeit, die Integrität und die Verfügbarkeit gewahrt werden. Dies gilt natürlich auch für Geräte im Internet of Things (IoT). Hierbei werden insbesondere auch kleine Geräte eingesetzt, die hinsichtlich ihrer Eigenschaften stark eingeschränkt sind. Dabei handelt es sich um Eigenschaften wie die Rechenkapazität, der Speicher oder die Übertragungskapazität [GBB15] der Geräte. Bei solch eingeschränkten Geräten stellt sich wiederum die Frage, wie der Schlüsselaustausch für eine sichere Kommunikation stattfinden kann, sodass unter anderem die Vertraulichkeit bei der späteren Kommunikation eingehalten werden kann. Dafür spezifiziert das ACE-OAuth-Framework allgemein Vorgänge, wie die Autorisierung der Kommunikationspartner beziehungsweise der Schlüsselaustausch ablaufen soll. Dieses Framework baut auf dem bereits existierenden Framework OAuth 2.0[1] und weiteren Bausteinen auf und definiert Profile, die auf (eingeschränkte) IoT-Geräte angepasst sind [SSW+19]. Im Zuge dieser Arbeit soll eines der Profile des ACE-OAuth-Frameworks näher untersucht werden. Dabei handelt es sich um das C3DC-Profil für ACE [GBB18b]. Dieses Profil spezifiziert, wie zwei autonome Geräte, die in Hinsicht auf ihre Ressourcen beschränkt sind, die benötigten Schlüsselmaterialien erhalten, die notwendig sind, um sichere Kommunikationskanäle aufzubauen und darüber kommunizieren zu können. Die Analyse des C3DC-Profils soll helfen, mögliche Schwachstellen des Entwurfs zu identifizieren. Ausgehend von der Analyse sollen gegebenenfalls Vorschläge entwickelt werden, wie diese Schwachstellen behoben werden könnten.

1.1 Motivation

Sicherheitsprotokolle verfolgen wichtige Ziele. Unter anderem kann dies die gegenseitige Authentifizierung zweier Kommunikationspartner sein. Längst sind jedoch nicht alle Protokolle sicher, sondern können Sicherheitslücken aufweisen. Ein Beispiel hierfür ist das Needham-Schroeder Public-Key-Protokoll [NS78], für das erst Jahre nach der Entwicklung eine Sicherheitslücke entdeckt wurde [Low96]. Einige dieser Protokolle werden einem Standardisierungsprozess unterzogen. Nach D. Basin et al. ist es schwierig, Änderungen an einem Protokoll vorzunehmen, wenn dieses einmal standardisiert wurde [BCM18]. Dementsprechend sollten Analysen solcher Protokolle auf Sicherheitslücken schon im Entwicklungsprozess durchgeführt werden, um mögliche Probleme zu vermeiden. Derzeit befindet sich das ACE-OAuth-Framework in einem Standardisierungsprozess. Bisherige Analysen

[1] Die OAuth 2.0-Spezifikation, siehe https://tools.ietf.org/html/rfc6749, wird hier nicht näher betrachtet.

haben das ACE-OAuth-Framework beziehungsweise Teile davon bereits untersucht [AT19, Tsc18]. Ebenso war das DTLS-Profil des ACE-OAuth-Framework Bestandteil einer Untersuchung [GBB18a]. Das C3DC-Profil ist bisher jedoch kein Bestandteil einer Analyse gewesen. Um sicherzustellen, dass das C3DC-Profil auch den Sicherheitsansprüchen entspricht, soll hierfür eine Analyse durchgeführt werden.

1.2 Zielsetzung

Das Ziel dieser Arbeit besteht darin, eine Analyse des vorliegenden Profils des ACE-OAuth-Frameworks auf Sicherheitslücken vorzunehmen und daraus resultierend festzustellen, ob dieses Profil den Annahmen entsprechend sicher ist oder ob Verbesserungen notwendig sind und diese gegebenenfalls darzustellen. Als Instrument für diese Untersuchung soll das Werkzeug Scyther [Cre14b] herangezogen werden, anhand dessen die Spezifikation getestet werden kann. Zuvor soll eine bereits vorhandene Analyse des ACE-OAuth-Frameworks [Tsc18] kritisch in Bezug auf dessen Testspezifikation betrachtet werden. Hierbei soll nachgeprüft werden, ob die Testspezifikation für das ACE-OAuth-Framework, wie sie besteht, noch richtig ist und ob dementsprechend Teile daraus verwendet werden können, die eine Erweiterung auf das C3DC-Profil ermöglichen, oder ob hierfür eine neue Testspezifikation erstellt werden muss. Die Ergebnisse, die aus der Analyse mittels Scyther gewonnen werden, sollen im Anschluss evaluiert werden.

1.3 Aufbau der Arbeit

Die Struktur der Arbeit lässt sich in die folgenden Schritte aufteilen. Zunächst soll in Kapitel 2 ein allgemeines Verständnis über die verwendete Terminologie der jeweiligen Protokolle erbracht werden, bevor die eigentlichen Protokolle vorgestellt werden. Daraufhin wird sich in Kapitel 3 mit verwandten Arbeiten beschäftigt, die Einfluss auf diese Arbeit nehmen könnten. Hierbei wird unter anderem auch das Werkzeug vorgestellt, welches zur Analyse zu Hilfe genommen wird. In Kapitel 4 wird dann die besagte Analyse und eine Evaluierung der Ergebnisse vorgenommen, bevor Lösungsansätze vorgestellt werden. Abschließend wird die Arbeit in Kapitel 5 als Ganzes betrachtet und es wird auf Aspekte eingegangen, die eine Fortführung dieser Arbeit ermöglichen könnten. In der Anlage lässt sich zunächst die Scyther-Spezifikation für das CTP von H. Tschofenig [Tsc18] finden, welche die Grundlage für diese Arbeit bildet. Daraufhin folgt die Angabe der vollständigen Spezifikation des C3DC-Profils mit englischen Kommentaren. Es wurde sich hierbei für die Verwendung englischer Kommentare entschieden, um Interessenten dieser Spezifikation das Lesen dieser zu erleichtern. Anschließend folgen die Angriffsgraphen, die mittels Scyther generiert wurden, wobei diese in ihrer Struktur geändert wurden zur Verbesserung der Lesbarkeit. Der letzte Abruf der Quellen erfolgte am 04. September 2019.

2 Grundlagen

Um ein allgemeines Verständnis über die Protokolle zu geben, werden in den folgenden Sektionen die wesentlichen Grundzüge dieser Protokolle dargelegt. Zunächst werden dafür die gültigen Definitionen der verwendeten Terminologie aufgezeigt, bevor die eigentlichen Protokolle näher betrachtet werden.

2.1 Terminologie

Diese Sektion soll die gültigen Definitionen der verwendeten Begriffe beschreiben.

2.1.1 Resource

Im Zusammenhang mit der ACE-OAuth-Architektur[2] definieren S. Gerdes et al. eine Resource als einen Gegenstand, der für eine Partei von Interesse ist [GSSB18]. Hierbei werden unter Gegenstände immaterielle Informationen beziehungsweise Daten, welche beispielsweise Sensorwerte oder Ähnliches repräsentieren können, verstanden [GSSB18]. Unter Partei werden hier der Resource Owner[3] und die Requesting Party[4] verstanden.

2.1.2 Rolle

Als Rolle (oder Akteur[5]) wird ein Teilnehmer an einem Protokoll definiert. Dieser Teilnehmer fungiert als logische Einheit und führt im Zuge eines Protokollablaufs eine oder mehrere definierte Aufgaben aus [GSSB18].

2.1.3 Client

Ein Client ist eine Rolle beziehungsweise eine Entität, die ein Interesse an einer Resource bekundet, indem sie versucht diese Resource bei einem Resource Server anzufragen [GSSB18]. Um Zugriff auf die Resource zu erhalten, muss der Client zunächst die erfoderlichen Berechtigungen einholen.

2.1.4 Resource Server

Der Resource Server ist ebenfalls als Rolle zu betrachten. Der Resource Server stellt eine Resource zur Verfügung [GSSB18], an der unter anderem der Client ein Interesse haben kann. Da auf eine Resource nur bei Vorhandensein der erforderlichen Berechtigungen zugegriffen werden darf, muss der Resource Server diese Berechtigungen prüfen, wenn beispielsweise ein Client auf die Resource zugreifen möchte.

[2] Siehe Abschnitt 2.2.

[3] Siehe Unterabschnitt 2.1.7.

[4] Siehe Unterabschnitt 2.1.8.

[5] In „An architecture for authorization in constrained environments“ [GSSB18] wird von Akteuren anstatt von Rollen gesprochen. Um jedoch einheitlich in der Benennung mit Scyther zu bleiben, wurde sich hier für Rolle entschieden. Rolle steht hierbei ausdrücklich nicht im Zusammenhang mit der rollenbasierten Zugriffskontrolle (RBAC).

2.1.5 Authorization Server

Im Gegensatz zu dem Resource Server, der eine Resource bereitstellt, dient der Authorization Server der Bereitstellung von Informationen zur Autorisierung und Authentifizierung des Resource Servers [GSSB18]. Der Authorization Server fungiert als Rolle. Die Autorisierung und Authentifizierung wird durch den Authorization Server vorbereitet, indem unter anderem Schlüsselmaterialien erstellt und diese danach an die beteiligten Rollen verteilt werden.

2.1.6 Client Authorization Server

Der Client Authorization Server ist vergleichbar mit dem Authorization Server. Der Unterschied zwischen den beiden Rollen liegt darin, dass sich der Client Authorization Server um die Autorisierung und Authentifizierung des Clients kümmert [GSSB18]. Dies bedeutet, dass der Client Authorization Server mit dem Authorization Server die Berechtigungen zum Zugriff auf eine Resource durch den Client aushandelt.

2.1.7 Resource Owner

Resource Owner sind verantwortlich für ihre jeweiligen Resourcen und kontrollieren deren Zugriffsberechtigungen [GSSB18]. Hierbei wird unter anderem festgelegt, welche Clients auf diese Resourcen zugreifen dürfen. Resource Owner sind Personen, die als Rolle definiert werden und stellen die höchste Ebene der Hierarchie, hinsichtlich des Authorization Servers und der Resourcen, dar.

2.1.8 Requesting Party

Die Requesting Party ist eine Instanz beziehungsweise eine Rolle, welche für den Client verantwortlich ist. Zusammen mit dem Resource Owner zählt die Requesting Party zu den Overseeing Principals, welche durch Individuen repräsentiert werden [GSSB18]. Diese Rolle kontrolliert die Requests, die der Client an andere Rollen senden darf und die Responses, die der Client erhält [GSSB18].

2.1.9 Access Token

Unter einem Access Token werden Zugangsdaten verstanden, die notwendig sind, um sich gegen eine geschützte Resource auf einem Resource Server authentisieren und darauf zugreifen zu können [SSW+19]. Diese Datenstruktur ist so aufgebaut, dass sie die Berechtigungen zum Zugriff auf eine Resource durch einen Client enthält. Hierbei kann der Access Token das Schlüsselmaterial entweder selbst oder eine Referenz, die auf das Schlüsselmaterial beim Authorization Server verweist, enthalten. Der Access Token wird durch den Authorization Server ausgestellt. Der Client kann den Access Token dem Resource Server präsentieren, um Zugang zu der Resource zu erhalten [SSW+19].

2.1.10 Access Information

Access Information sind Daten, die den Resource Server betreffen und werden durch den Authorization Server an den Client ausgehändigt. Neben verschiedenen Parametern[6], kann hier auch Schlüsselmaterial betreffend des Resource Servers versendet werden[7] [SSW+19].

[6] Die Parameter, die in der Spezifikation verwendet werden, werden hier nicht näher betrachtet.
[7] Dies kann beispielsweise der öffentliche Schlüssel des Resource Servers sein.

2.1.11 Introspection

Die Introspection dient dem Resource Server, um den ihm präsentierten Access Token auf Gültigkeit überprüfen zu lassen [SSW+19]. Da der Resource Server eingeschränkt sein kann und die Überprüfung möglicherweise nicht selber durchführen kann, muss dieser die Überprüfung an den Authorization Server übergeben. Dies geschieht durch die Introspection. Der Authorization Server liefert entsprechende Informationen an den Resource Server zurück. Ebenfalls kann die Introspection genutzt werden, um die im Access Token enthaltene Referenz auf das Schlüsselmaterial aufzulösen, wenn das Schlüsselmaterial nicht selbst im Access Token enthalten ist [SSW+19].

2.2 Authentication and Authorization for Constrained Environments using the OAuth 2.0 Framework

Das ACE-OAuth-Framework dient der Authentifizierung und Autorisierung von Geräten im IoT. Explizit soll dies für eingeschränkte Geräte gelten. Hiermit wird ein Verfahren beschrieben, welches einen Client befähigt, entsprechende Berechtigungen von einem Authorization Server einzuholen, um Zugriff auf eine Resource auf einem Resource Server zu erhalten. Wie der Name des Frameworks bereits andeutet, nutzt dieses das OAuth 2.0 und weitere Frameworks als Grundblöcke, um darauf aufzubauen.

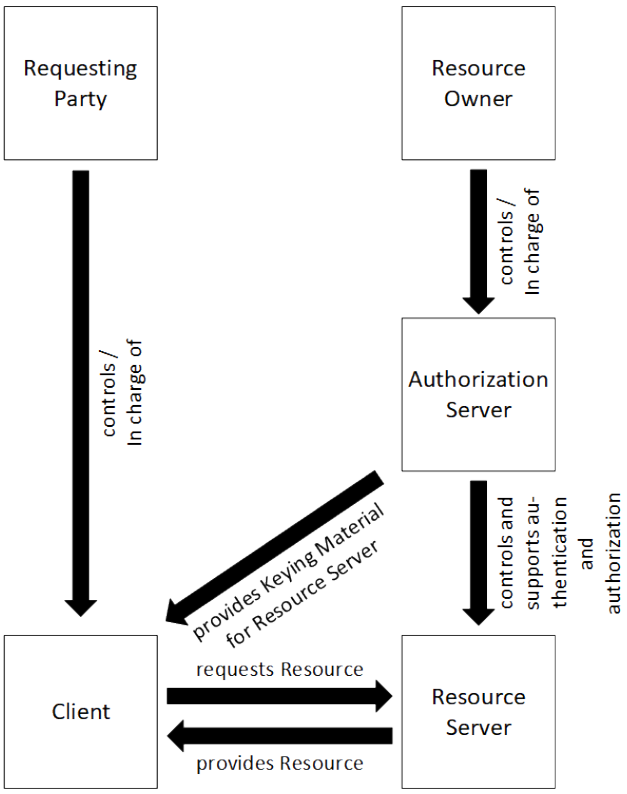

Abbildung 2.1: Darstellung der Rollen in der ACE-OAuth-Architektur nach S. Gerdes et al.[8] [GBB18a].

[8]Die in den Dokumenten abgedruckten Versionen der Darstellung der Rollen in der ACE-OAuth-Architektur wurde leicht abgeändert, um einen einheitlichen Stil in dieser Arbeit zu gewährleisten.

Der ACE-OAuth-Architektur sind verschiedene Rollen[9] bekannt. Neben den bereits beschriebenen Rollen des Clients, des Resource Servers und des Authorization Servers existieren in der ACE-OAuth-Architektur zusätzlich der Resource Owner und die Requesting Party[10].

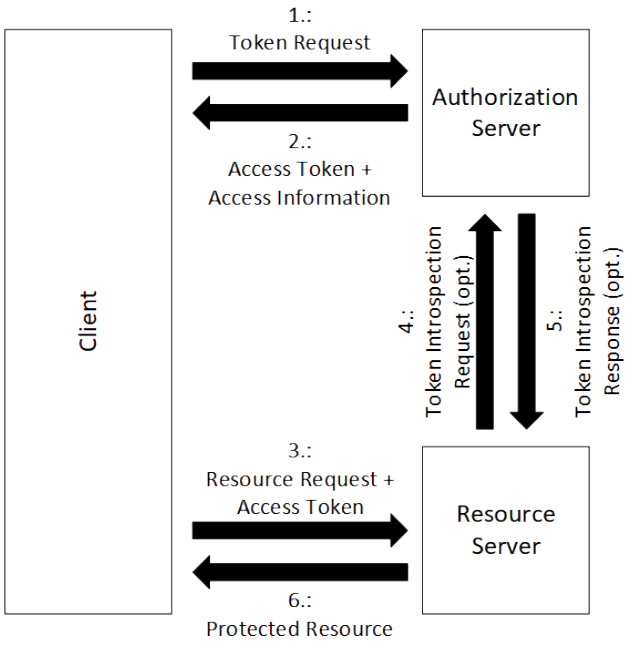

Abbildung 2.2: Darstellung der ACE-OAuth-Architektur nach H. Tschofenig [Tsc18] und L. Seitz et al.[11] [SSW+19].

Der Ablauf der ACE-OAuth-Architektur wird in Abbildung 2.2 dargestellt. Es handelt sich hierbei im Wesentlichen um die folgenden Schritte[12]:

1. **Token Request:**
   Damit ein Client die erforderlichen Zugriffsberechtigungen und die jeweiligen Schlüsselmaterialien erhält, um auf eine Resource auf dem Resource Server zugreifen zu können, muss der Client zunächst einen Token Request an den Authorization Server senden. In diesem Request kann der Client den Resource Server angeben, für den er die Berechtigungen erfragt. Darüber hinaus kann der Client spezifizieren, welche Methoden er unter anderem zur Verschlüsselung[13] nutzen möchte [SSW+19]. Um diese Kommunikation zu schützen, nutzt der Client das mit dem Authorization Server zuvor etablierte Schlüsselmaterial[14] [SSW+19].

[9] Siehe Unterabschnitt 2.1.2.

[10] Darüber hinaus definiert „An architecture for authorization in constrained environments“ [GSSB18] den Client Authorization Server (siehe Abschnitt 2.3) als weitere Rolle, die hier keine Anwendung findet beziehungsweise in der ACE-OAuth-Spezifikation nicht implementiert ist.

[11] Die in den Dokumenten abgedruckten Versionen der Darstellung der ACE-OAuth-Architektur wurden leicht abgeändert, um einen einheitlichen Stil in dieser Arbeit zu gewährleisten.

[12] Die Schritte zum Entdecken der jeweiligen Server werden hier nicht betrachtet.

[13] Hierbei sei beispielsweise die symmetrische Kryptographie genannt.

[14] Das ACE-OAuth-Framework nimmt an, dass der Client sich bereits vor dem Request beim Authorization Server registriert und entsprechendes Schlüsselmaterial ausgetauscht hat. Dies ist jedoch kein Bestandteil des eigentlichen Frameworks.

2. **Access Token + Access Information:**
   Auf den Request des Clients folgt die Antwort (Response) des Authorization Servers. Der Authorization Server überprüft zunächst, ob der Request valide ist und der Client die erforderlichen Befugnisse durch den Resource Owner besitzt, um auf die Resource zugreifen zu dürfen [SSW+19]. Kann der Authorization Server den Request des Clients erfolgreich verarbeiten, stellt der Authorization Server einen Access Token aus. Die Funktionsweise des Access Tokens wird in Unterabschnitt 2.1.9 beschrieben. Die Informationen, das heißt die Schlüsselmaterialien, welche der Resource Server benötigt, um eine geschützte Verbindung zum Client aufbauen zu können, können im Access Token entweder selbst oder nur als Referenz enthalten sein [SSW+19]. Ist im Access Token nur eine Referenz enthalten, so befindet sich das Schlüsselmaterial, welches für die Kommunikation zwischen dem Client und dem Resource Server verwendet werden muss, beim Authorization Server. Andernfalls liegt das Schlüsselmaterial verschlüsselt im Access Token vor. Darüber hinaus kann der Authorization Server Access Information an den Client senden. Darunter verstehen sich Informationen, wie in Unterabschnitt 2.1.10 beschrieben. Um den Access Token und die Access Information zu schützen, verschlüsselt der Authorization Server diese Informationen und nutzt darüber hinaus einen Integritätsschutz.

3. **Resource Request + Access Token:**
   Nachdem der Client den Access Token durch den Authorization Server ausgestellt bekommen hat, kann der Client einen Resource Request an den Resource Server senden. Dieser Request dient dem Client, um eine Resource anzufragen. In diesem Request gibt der Client den zuvor erhaltenen Access Token, unter Benutzung der erhaltenen Access Information, an [SSW+19]. Da zwischen dem Resource Server und dem Client noch kein Schlüsselmaterial etabliert wurde, handelt es sich hierbei um einen nicht geschützten Request[15].

4. **Token Introspection Request** (optional)**:**
   Der Token Introspection Request dient dem Resource Server, um die Token Introspection des Authorization Servers zu nutzen. Dies ist ein optionaler Schritt. Sofern der Access Token das tatsächliche Schlüsselmaterial und nicht nur eine Referenz enthält und zusätzlich der Resource Server in der Lage ist den Access Token selbst zu validieren, kann auf den Schritt der Token Introspection und damit auch auf den folgenden Schritt verzichtet werden. Ist der Resource Server nicht in der Lage den Access Token selbst zu validieren oder enthält der Access Token nur eine Referenz auf das Schlüsselmaterial, welches auf dem Authorization Server hinterlegt ist, so muss der Resource Server die Token Introspection nutzen, um den Access Token durch den Authorization Server auf Validität überprüfen zu lassen [SSW+19]. Dazu sendet der Resource Server einen Request an den Authorization Server unter Angabe des Access Tokens. Hierbei muss die Kommunikation zwischen dem Resource Server und dem Authorization Server verschlüsselt erfolgen [SSW+19].

[15] Hierbei handelt es sich insofern um einen nicht geschützten Request, als dass dieser nicht zwischen dem Client und dem Resource Server verschlüsselt wird.

5. **Token Introspection Response** (optional)**:**
   Der Token Introspection Request wird mittels einer Antwort des Authorization Servers entgegnet. Dies ist ein optionaler Schritt, wie unter dem Token Introspection Request beschrieben. Wenn der Request erfolgreich verarbeitet werden konnte, sendet der Authorization Server eine Antwort an den Resource Server [SSW+19]. In dieser Response gibt der Authorization Server Informationen über den Access Token zurück. Falls im Access Token nur eine Referenz auf das Schlüsselmaterial beim Authorization Server enthalten ist, so wird hier eben dieses Schlüsselmaterial, welches der Resource Server nutzen muss, versendet. Andernfalls kann der Authorization Server in seiner Antwort auch weitere Parameter übergeben, die über die Validität des Access Tokens Rückschlüsse bieten. Mit diesen Informationen soll der Resource Server in der Lage sein, zu entscheiden, ob ein Resource Request zugelassen oder abgelehnt werden muss [SSW+19].

6. **Protected Resource:**
   Nach der Überprüfung des Access Tokens auf Validität und der erfolgreichen gegenseitigen Authentifizierung, sendet der Resource Server eine Antwort an den Client. Diese enthält die angeforderte Resource. Hierbei wird die Antwort mit dem zuvor ausgetauschten Schlüsselmaterial geschützt [SSW+19].

Das ACE-OAuth-Framework überlässt die Spezifizierung bestimmter Sachverhalte sogenannten Profilen. Ein Profil ist eine Erweiterung des eigentlichen Frameworks um spezifische Details, die von den eigentlichen Spezifikationen der Frameworks offen gelassen werden, da diese oftmals unterschiedlich implementiert werden können und es somit den Profilen obliegt, dies zu spezifizieren. Zwei dieser Profile sind das C3DC-Profil und das DTLS-Profil, die im Folgenden vorgestellt werden.

2.3 Constrained Client/Cross-Domain Capable Authorization Profile for ACE

Das Constrained Client/Cross-Domain Capable Authorization Profile for ACE (C3DC-Profil) ist, wie der Name bereits suggeriert, ein Profil der ACE-OAuth-Architektur[16] und baut dementsprechend darauf auf. Dieses Profil beschreibt, wie zwei eingeschränkte Geräte die notwendigen Informationen und Schlüsselmaterialien erhalten, um geschützt miteinander kommunizieren zu können [GBB18b]. Hierfür definiert dieses Profil die Nutzung zweier weniger eingeschränkten Geräte. Für den Resource Server ist dies der Authorization Server, wie in Unterabschnitt 2.1.5 beschrieben. Für den Client wird der Client Authorization Server spezifiziert, wie in Unterabschnitt 2.1.6 beschrieben. Dementsprechend lässt sich der Client Authorization Server in Abbildung 2.1 zwischen der Requesting Party und dem Client einordnen, wie dies der Fall für den Authorization Server zwischen dem Resource Owner und dem Resource Server ist. Der Client Authorization Server und der Authorization Server dienen der Übernahme von schwierigen Aufgaben, wie beispielweise der Aushandlung der Zugriffsberechtigungen und Schlüsselmaterialien [GBB18b].

[16] Siehe Abschnitt 2.2.

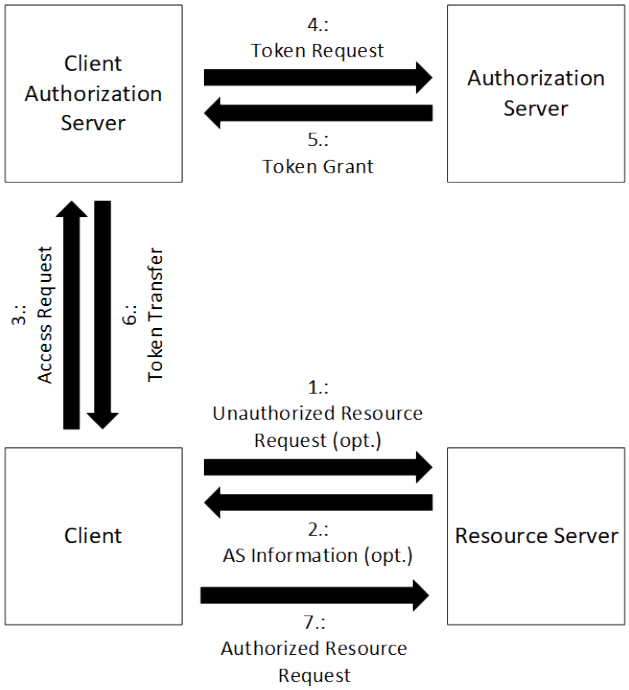

Abbildung 2.3: Darstellung des C3DC-Profils der ACE-OAuth-Architektur nach S. Gerdes et al.[17] [GBB18b].

Das C3DC-Profil, dessen Protokoll-Ablauf in Abbildung 2.3 dargestellt wird, lässt sich durch die folgenden Schritte beschreiben:

1. **Unauthorized Resource Request** (optional)**:**
   Zunächst sendet der Client einen Unauthorized Resource Request an den Resource Server. Mittels des Requests versucht der Client Informationen über den Authorization Server zu erhalten, welcher zuständig für den Resource Server ist [GBB18b]. Diese Informationen nutzt der Client, um mittels des Client Authorization Servers die benötigten Berechtigungen vom Authorization Server einzuholen. Dieser Schritt ist optional und muss nicht vom Client durchgeführt werden.

2. **AS Information** (optional)**:**
   Auf den Unauthorized Resource Request des Clients antwortet der Resource Server mit den AS Information[18]. Dies geschieht, da der Client keine Berechtigungen besitzt auf die angefragte Resource zuzugreifen. Dementsprechend werden die AS Information versendet, um es dem Client zu ermöglichen, sich über den Client Authorization Server die Berechtigungen durch den Authorization Server einzuholen. Diese Informationen enthalten insbesondere die Adresse des Authorization Servers [GBB18b]. Dieser Schritt ist optional und erfolgt nur als Antwort auf den Unauthorized Resource Request des Clients.

[17] Die in den Dokumenten abgedruckten Versionen der Darstellung des C3DC-Profils der ACE-OAuth-Architektur wurde leicht abgeändert, um einen einheitlichen Stil in dieser Arbeit zu gewährleisten.

[18] In der aktuellen Version des ACE-OAuth-Frameworks [SSW+19] wird dieser Schritt als AS Request Creation Hints bezeichnet.

3. **Access Request:**
   Mittels des Access Requests wird es dem eingeschränkten Client ermöglicht, das Aushandeln des, für die Kommunikation mit dem Resource Server benötigten, Schlüsselmaterials auf den weniger eingeschränkten Client Authorization Server zu verlagern. Um die Aushandelung des Schlüsselmaterials zu verlagern, sendet der Client eine Anfrage an den Client Authorization Server.

4. **Token Request:**
   Der Token Request wird vom Client Authorization Server an den Authorization Server gerichtet, um die erforderlichen Schlüsselmaterialien für den Client auszuhandeln. Hierbei überprüft der Client Authorization Server zunächst, ob der Client auf den Resource Server zugreifen darf und ob der Authorization Server überhaupt die Berechtigungen besitzt, Informationen über den Resource Server und die Schlüsselmaterialien herauszugeben [GBB18b]. Dies geschieht unter Einbezug der Informationen, welche die Requesting Party dem Client Authorization Server zur Verfügung stellt. Ebenfalls führt der Authorization Server eine Überprüfung durch, ob der Client durch den Resource Owner berechtigt ist, auf den Resource Server zuzugreifen und ob der Client Authorization Server berechtigt ist, Informationen über den Client weiterzugeben [GBB18b]. Nachdem der Client Authorization Server und der Authorization Server jeweils bestätigen konnten, dass die Kommunikation durch die jeweiligen Overseeing Principals zugelassen wird, authentifizieren sich die beiden Rollen gegenseitig. Da der Client Authorization Server und der Authorization Server weniger stark eingeschränkt sind, können diese beiden Rollen aufwendigere Protokolle zur Kommunikation und zur Verschlüsselung nutzen [GBB18b].

5. **Token Grant:**
   Auf den Token Request reagiert der Authorization Server, indem dieser die Schlüsselmaterialien für den Zugriff auf den Resource Server erzeugt. Hierfür stellt der Authorization Server den Access Token aus, den der Client dem Resource Server präsentieren muss, um Zugriff auf die Resource zu erhalten. In dem Access Token ist das Schlüsselmaterial enthalten, welches für den Resource Server gedacht ist. Darüber hinaus kann der Authorization Server die Access Information ausstellen, die Informationen für den Client enthalten [GBB18b]. Hierin enthalten, ist das Schlüsselmaterial, welches für den Client gedacht ist [GBB18b]. Die Antwort an den Client Authorization Server erfolgt verschlüsselt, um den Inhalt des Access Tokens und der Access Information zu schützen.

6. **Token Transfer:**
   Nachdem der Client Authorization Server den Access Token und die Access Information durch den Token Grant vom Authorization Server erhalten hat, sorgt der Client Authorization Server für die Weitergabe der Informationen an den Client. Die Weitergabe dieser Informationen erfolgt verschlüsselt über den aufgebauten DTLS-Kanal zwischen dem Client und dem Client Authorization Server. In diesem Schritt kann der Client Authorization Server, im Auftrag der Requesting Party, weitere Einschränkungen für den Zugriff

auf den Resource Server durch den Client erwirken, indem diese zu den Access Information hinzugefügt werden [GBB18b].

7. **Authorized Resource Request:**
   Durch den Erhalt des Access Tokens und der Access Information ist es dem Client möglich eine Resource beim Resource Server anzufragen. Zunächst muss jedoch ein DTLS-Kanal zwischen dem Client und dem Resource Server aufgebaut werden. Dabei behält der Client die Access Information bei sich und präsentiert den Access Token dem Resource Server [GBB18b]. Dies geschieht, indem der Client diesen im DTLS-Handshake verwendet. Der aufgebaute DTLS-Kanal wird daraufhin genutzt, um verschlüsselt kommunizieren zu können und die Resource anzufragen [GBB18b].

2.4 Datagram Transport Layer Security Profile for ACE

Ein weiteres Profil der ACE-OAuth-Spezifikation ist das Datagram Transport Layer Security Profile for ACE [GBB+18c]. Diese Spezifikation des Profils erlaubt es eingeschränkten Servern die Authentifizierung und die Autorisierung von Clients auf weniger eingeschränkte Server zu delegieren [GBB+18c]. Darüber hinaus definiert dieses Profil die Verwendung von CoAP [SHB14] über DTLS für die Kommunikation zwischen dem Client und dem Resource Server, sowie für die Kommunikation zwischen dem Client und dem Authorization Server zur Aushandlung und für den Austausch eines Access Tokens.

Dieses Profil definiert die Verwendung von zwei Sicherheitsmodi[19] der CoAP-Spezifikation [SHB14] und beschreibt die Unterschiede, welche die Nutzung des jeweiligen Sicherheitsmodus in diesem Profil mit sich zieht. Dabei wird der RawPublicKey Modus und der PreSharedKey Modus unterschieden. Neben den Unterschieden der beiden Sicherheitsmodi, ergeben sich daraus auch Unterschiede in den Informationen, welche die jeweiligen Nachrichten der ACE-OAuth-Spezifikation[20] enthalten [GBB+18c].

1. **RawPublicKey Modus**:
   In diesem Modus nutzt der Client einen Raw Public Key[21] und gibt diesen in seinem Token Request an den Authorization Server an. Bevor der Authorization Server einen Access Token als Antwort auf den Token Request ausstellen darf, muss der Authorization Server feststellen, ob der im Token Request angegebene Raw Public Key auf den Client zurückzuführen ist [GBB+18c]. Um dem Client mitzuteilen, dass dieses Profil bei der Kommunikation mit dem Resource Server verwendet werden muss, fügt der Authorization Server seiner Antwort einen Parameter hinzu, mittels dessen Wert ein Rückschluss auf die Verwendung des Profils möglich ist. In dem Access Token ist der Raw Public Key des Clients enthalten, den der Resource Server nutzen muss.

[19] Ein Sicherheitsmodus in CoAP beschreibt die jeweiligen Möglichkeiten zum Schutz der Kommunikation unter Angabe des zu nutzenden Schlüsselmaterials [SHB14].

[20] Siehe Abschnitt 2.2.

[21] Ein Raw Public Key ist ein asymmetrisches Schlüsselpaar ohne entsprechendes Zertifikat [SHB14].

Darüber hinaus versendet der Authorization Server in seiner Antwort auch den Raw Public Key des Resource Servers für den Client.

Nach Erhalt der Antwort muss der Client sicherstellen, dass diese Antwort zu dem zuvor gesendeten Token Request gehört [GBB+18c]. Bei der Verwendung von asymmetrischem Schlüsselmaterial, wie es in diesem Modus der Fall ist, muss der Client vor dem Aufbau der DTLS-Sitzung beachten, den Access Token bereits vor dem Handshake dem Resource Server bekannt zu machen. Dafür richtet der Client eine Anfrage an den Resource Server unter Angabe des Access Tokens. Bei der Ausführung des Handshakes wird der öffentliche Schlüssel des Clients im Access Token angegeben. Dies dient der Signalisierung der Nutzung des RawPublicKey Modus.

gibt der Client den öffentlichen Schlüssel an, den dieser zuvor im Access Token erhalten hat [GBB+18c], um zu signalisieren, dass der RawPublicKey Modus verwendet werden soll.

Nach dem Erhalt des Access Tokens durch den Client, muss der Resource Server zunächst prüfen, ob dieser gültig ist, durch den Authorization Server ausgestellt wurde und ob der Resource Server der angedachte Empfänger der Anfrage ist [GBB+18c]. Der Client und der Resource Server nutzen das erhaltene Schlüsselmaterial, um eine DTLS-Sitzung aufzubauen.

2. **PreSharedKey Modus**:
   Im PreSharedKey Modus wird ein Pre Shared Key[22] zur Verschlüsselung genutzt. Um einen Access Token zu erhalten, sendet der Client einen Token Request an den Authorization Server. In dieser Anfrage kann der Client eine Referenz auf einen Schlüssel spezifizieren. Diese Referenz kann der Authorization Server nutzen, um den bei sich gespeicherten symmetrischen Schlüssel[23] zu identifizieren und damit einen Access Token zu erzeugen. Die Erzeugung des Access Tokens erfolgt unter Berücksichtigung der Regeln, die der Resource Owner für den Client aufgestellt hat [GBB+18c].

   Anstelle des symmetrischen Schlüssels, kann der Authorization Server auch eine Funktion[24] und weitere Informationen im Access Token angeben, die es dem Resource Server ermöglichen daraus einen Schlüssel zu berechnen [GBB+18c]. Um dem Client mitzuteilen, dass dieses Profil bei der Kommunikation mit dem Resource Server verwendet werden muss, fügt der Authorization Server seiner Antwort einen Parameter hinzu, mittels dessen Wert ein Rückschluss auf die Verwendung des Profils möglich ist. Darüber hinaus fügt der Authorization Server der Antwort weitere Informationen hinzu, die der Client benötigt, um die DTLS-Sitzung aufzubauen [GBB+18c]. Hier enthalten ist unter anderem der symmetrische Schlüssel, welcher vom

[22] Ein Pre Shared Key ist ein symmetrischer Schlüssel, welcher vor der Kommunikation zweier Partner ausgetauscht werden muss [ET05].

[23] Hierbei ist der Pre Shared Key gemeint.

[24] Hierbei ist eine Key Derivation Function gemeint. Eine Key Derivation Function ist eine Funktion zur Erstellung eines Schlüssels ausgehend von anderem kryptografischen Material, wie beispielsweise einem Passwort.

Client verwendet werden muss.

Nach Erhalt der Antwort muss der Client sicherstellen, dass diese Antwort zu dem zuvor gesendeten Token Request gehört [GBB+18c]. Mit den erhaltenen Informationen können der Client und der Resource Server den DTLS-Handshake durchführen, indem der Client den Access Token zu Beginn des Handshake an den Resource Server sendet. Der Resource Server ist verpflichtet den erhaltenen Access Token zunächst auf Validität zu überprüfen, bevor dieser den Handshake vollziehen darf.

Durch den Aufbau einer DTLS-Sitzung zwischen dem Client und dem Resource Server wird es dem Client ermöglicht über eine geschützte Verbindung auf die Resource des Resource Servers zuzugreifen. Jede Anfrage, die vom Client über diese Verbindung an den Resource Server gesendet wird, unterliegt bestimmten Regeln. So muss der Resource Server für jede Anfrage überprüfen, ob der Client zu dem jeweiligen Zeitpunkt immernoch die erforderlichen Berechtigungen besitzt. Insbesondere sei hier die Gültigkeit des Access Tokens genannt [GBB+18c]. Anfragen, welche den Regeln nicht entsprechen, müssen vom Resource Server abgelehnt werden.

Das DTLS-Profil definiert darüber hinaus eine Methode, die es dem Client ermöglicht, die Autorisierungs-Informationen dynamisch beim Resource Server aktualisieren zu lassen, ohne die aufgebaute DTLS-Sitzung selbst zu beeinträchtigen [GBB+18c]. Hierzu führt der Client erneut den Token Request durch, um einen neuen Access Token anzufordern und lässt diesen dem Resource Server daraufhin zukommen.

2.5 Client-Token-Protokoll

Das Client-Token-Protokoll (CTP) war ein Protokoll der ACE-OAuth-Spezifikation. Das CTP wurde in der dritten Version[25] [SSW+16] der ACE-OAuth-Spezifikation eben unter diesem Namen hinzugefügt und in Version elf[26] [SSW+18] wieder entfernt. In dem CTP wird ein optionaler Ansatz beschrieben, wie ein Client mit eingeschränkter Funktion hinsichtlich seiner Konnektivität, Zugriff auf einen Resource Server erhält [SSW+17].

Der grundlegende Ablauf des CTPs wird in Abbildung 2.4 dargestellt. Vorausgehend muss der Client einen Langzeit Access Token vom Authorization Server ausgestellt bekommen haben. Das Schlüsselmaterial für den Client ist im Access Token nicht selbst enthalten, sondern ist lediglich eine Referenz auf das Schlüsselmaterial beim Authorization Server. Dementsprechend wird davon ausgegangen, dass der Resource Server die Token Introspection nutzen muss, um mehr über das Schlüsselmaterial zu lernen. Hierbei wird davon ausgegangen, dass die Schritte

[25] Hierbei handelt es sich um die Versionsnummer -02 der ACE-OAuth-Spezifikation.
[26] Hierbei handelt es sich um die Versionsnummer -10 der ACE-OAuth-Spezifikation.

eins und zwei der ACE-OAuth-Spezifikation, wie in Abschnitt 2.2 beschrieben, bereits vollzogen wurden.

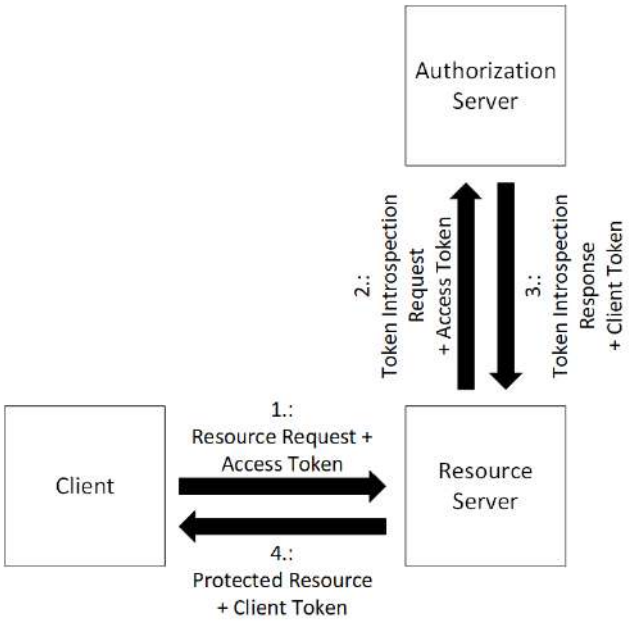

Abbildung 2.4: Darstellung des Client-Token-Protokolls nach H. Tschofenig[27] [Tsc18].

Mittels des Access Tokens kann der Client dann einen Request an den Resource Server senden, um eine Resource anzufragen. Dieser Request erfolgt unverschlüsselt, da kein Sicherheitskontext zwischen den beiden Rollen etabliert wurde. Da der Access Token lediglich eine Referenz auf das Schlüsselmaterial enthält, muss der Resource Server die Introspection nutzen, um die Validierung des Access Tokens an den Authorization Server zu übergeben [SSW+17]. Dies ist notwendig, da nur der Authorization Server die Referenz auflösen kann und wird im Schritt zwei dargestellt. Hierbei wird der Inhalt der Nachricht zwischen dem Resource Server und dem Authorization Server verschlüsselt. In der Antwort an den Resource Server übergibt der Authorization Server dem Resource Server die benötigten Informationen, welche beschreiben, welche Berechtigungen der Client besitzt. Darüber hinaus werden die benötigten Schlüsselmaterialien versendet. Die Nachricht wird zwischen dem Resource Server und dem Authorization Server verschlüsselt. In der Antwort wird der sogenannte *client_token*-Parameter verwendet, der den Schlüssel beziehungsweise den Client-Token beinhaltet, welcher der Client verwenden muss, um auf eine Resource auf dem Resource Server zugreifen zu können [SSW+17]. Der Client-Token ist durch den Authorization Server verschlüsselt, sodass nur der Client in der Lage ist, diesen zu entschlüsseln [SSW+17]. Nach Erhalt der Antwort vom Authorization Server, leitet der Resource Server im letzten Schritt den Client-Token.

[27] Die in den Dokumenten abgedruckten Versionen der Darstellung des Client-Token-Protokolls wurde leicht abgeändert, um einen einheitlichen Stil in dieser Arbeit zu gewährleisten.

3 Verwandte Arbeiten

In seiner Veröffentlichung beschreibt C. Cremers, dass die Analyse von Sicherheitsprotokollen schwierig ist und dementsprechend nach einer effektiven Methode geforscht wird, diese Probleme effizient zu bewältigen [Cre08a]. Die Ergebnisse der bisherigen Forschung haben verschiedene Werkzeuge hervorgebracht, wie beispielweise AVISPA [ABB+05], welche die automatisierte Verifikation beziehungsweise Falsifikation von Sicherheitsprotokollen vornehmen [Cre08a]. Darüber hinaus wurde das Werkzeug Scyther [Cre14b] entwickelt, welches ebenfalls für die formale Analyse von Sicherheitsprotokollen genutzt wird.

In den folgenden Sektionen wird zunächst beschrieben, welche Methode der formalen Analyse zugrundeliegt, bevor die verwendete Terminologie beschrieben und das Scyther-Werkzeug näher betrachtet wird. Daraufhin wird auf die Analyse des Client-Token-Protokolls eingegangen, bevor die Analyse der ACE-OAuth-Architektur betrachtet wird. Abschließend wird die Veröffentlichung von S. Gerdes et al. über die Vermeidung von Sicherheitslücken in Autorisierungsprozessen im Internet of Things vorgestellt.

3.1 Modellprüfung

Die Verifikation von Sicherheitsprotokollen kann mittels der Prüfung eines Modells[28] erfolgen. Dabei wird versucht, alle möglichen Wege zu identifizieren, die ein Angreifer nutzen kann, um Einfluss auf ein oder mehrere Protokolldurchläufe zu nehmen [BCM18].

Grundlage hierfür ist zunächst ein formales Modell, welches die Modellierung der Protokolle ermöglicht [BCM18]. Protokolle werden hierbei durch eine Menge von Rollen definiert. Die Rollen wiederum werden durch eine Menge von Nachrichten definiert, die wiederum durch Terme repräsentiert werden [BCM18]. Darüber hinaus verwenden Modellprüfer ein Angriffsmodell. Dieses Modell wird genutzt, um einen Angreifer in dem Netzwerk zu modellieren. In einigen dieser Werkzeuge wird das Dolev-Yao-Angreifermodell, wie durch D. Basin et al. [BCM18] vorgestellt, verwendet. Hierin wird beschrieben, dass alle über ein Netzwerk gesendeten Nachrichten von einem Angreifer abgefangen, modifiziert, sowie weitergeleitet werden können [Cre08b, BCM18]. Dementsprechend hat ein Angreifer in diesem Modell die volle Kontrolle über das Netzwerk [Cre08b]. Darüber hinaus besitzt der Angreifer Informationen über jene durch ihn kompromittierten Rollen. Damit besitzt der Angreifer auch die Schlüsselmaterialien der kompromittierten Rollen [Cre08b]. Eine weitere Annahme, die in dem Modell getroffen wird, ist, dass verschlüsselte Nachrichten nur mittels des entsprechenden Schlüsselmaterials

[28] Modellprüfung wird für den englischen Begriff „Model Checking“ verwendet.

entschlüsselt werden können [BCM18].

Um diese Werkzeuge nutzen zu können, muss eine gewisse Abstraktion der eigentlichen Protokolle erfolgen. Dies bedeutet, dass die Protokolle auf ihren grundlegenden Nachrichtenaustausch reduziert werden müssen. Im Falle der oben beschriebenen Protokolle[29] wird dies dementsprechend auf den Schlüsselaustausch reduziert. H. Tschofenig beschreibt in seiner Veröffentlichung, dass die Abstraktion der Protokolle ein essenzieller Bestandteil der Analyse von Sicherheitsprotokollen ist und die fehlerhafte Abstraktion dazu führen kann, Sicherheitslücken nicht identifizieren zu können [Tsc18].

Ein wichtiger Vorteil der Verwendung von den oben genannten Werkzeugen ist, dass diese Modellprüfer[30] tatsächliche Angriffe auf die Sicherheitsprotokolle identifizieren können und somit Rückschlüsse auf die Behebung dieser Probleme beziehungsweise auf die Verbesserung der Protokolle möglich sind [BCM18].

Das Werkzeug AVISPA [ABB+05] ist ein Beispiel für ein Werkzeug, welches die Modellprüfung eines Sicherheitsprotokolls vornimmt. AVISPA vereint hierfür mehrere Ansätze, diese Analyse durchzuführen und stellt hierfür eine einheitliche Oberfläche zur Verfügung [BCM18]. Zu überprüfende Protokolle werden in einer formalen Sprache definiert, die für dieses Werkzeug entwickelt wurde [ABB+05]. Die Ergebnisse der Analyse werden in Form von Nachrichten-Reihenfolge-Diagrammen[31] dargestellt.

Ein weiteres Werkzeug, welches die Modellprüfung vornimmt, ist das von C. Cremers entwickelte Scyther-Werkzeug, welches in Abschnitt 3.2 vorgestellt wird. Zuvor soll auf die Entscheidung für die Nutzung des Scyther-Werkzeugs eingegangen werden. Dies lässt sich im Wesentlichen auf die folgenden Aspekte zurückführen:

1. **Das Vorhandensein einer Spezifikation für das CTP:**
   Für das CTP aus der ACE-OAuth-Spezifikation[32] [SSW+17] ist bereits jeweils eine Implementierung des Protokolls in Scyther, sowie in AVISPA vorhanden, die aus der Analyse von H. Tschofenig [Tsc18] hervorgehen. Daher wird die Eingrenzung auf diese beiden Werkzeuge als sinnvoll erachtet, da dies gegebenenfalls ermöglicht, Teile der Arbeit von H. Tschofenig wiederzuverwenden. Sofern sich in der späteren Analyse herausstellt, dass diese Teile für die Analyse des C3DC-Profils verwendet werden können und mit der aktuellen Spezifikation des ACE-Frameworks übereinstimmen.

2. **Die Komplexität der Werkzeuge:**
   Scyther überzeugt hier durch einfache Konstrukte, die zur Modellierung der Protokolle benötigt werden. Anhand der komplexen Konstrukte, die

[29] Siehe Kapitel 2.

[30] Modellprüfer wird für den englischen Begriff „Model Checker“ verwendet.

[31] Nachrichten-Reihenfolge-Diagramm wird für den englischen Begriff „Message Sequence Chart“ verwendet.

[32] Hierbei handelt es sich um die Versionsnummer -09 der ACE-OAuth-Spezifikation.

in AVISPA benötigt werden [KKSL17], um die jeweiligen Sachverhalte darstellen zu können, erscheint es dementsprechend sinnvoller das Scyther-Werkzeug zu nutzen.

3.2 Scyther

Das Scyther-Werkzeug [Cre14b] ist ein Anatz, um den von C. Cremers [Cre08a] beschriebenen Problemen entgegenzuwirken. Dieses Werkzeug dient der Unterstützung bei der Analyse von Sicherheitsprotokollen und bei dem Beweisen der Korrektheit solcher Protokolle [Cre08a]. Zur Analyse eines Protokolls muss dieses in der formalen Sprache[33] des Werkzeugs spezifiziert werden. Anhand dieser Sprache werden die jeweiligen Nachrichten definiert, die im Protokoll spezifiziert sind. Darüber hinaus können Forderungen[34] an ein Protokoll gestellt werden, die dieses erfüllen muss. Das Werkzeug Scyther analysiert daraufhin, ob diese Forderungen durch einen Angreifer beeinträchtigt werden und stellt die Ergebnisse in Form eines Angriffsgraphen[35] dar. Hierin wird beschrieben, wie der Angreifer diese Forderungen verletzen könnte.

Scyther ist ein Werkzeug, welches die ungebundene Modellprüfung[36] durchführen kann [BCM18]. Damit wird erreicht, dass alle möglichen Verhaltensweisen des Protokolls berücksichtigt und somit die geforderten Eigenschaften[37] der Protokolle eingehalten werden können [Cre08b]. Ist das Werkzeug nicht in der Lage die ungebundene Verifikation durchzuführen, nutzt Scyther die Möglichkeit eine an Grenzen gebundene[38] Verifikation durchzuführen, wie es in anderen Werkzeugen[39] genutzt wird.

3.2.1 Terminologie

Diese Sektion soll die gültigen Definitionen der verwendeten Begriffe bezüglich des Scyther-Werkzeugs beschreiben.

3.2.1.1 Events

Das Werkzeug Scyther modelliert Ereignisse beziehungsweise Events, die als Grundbausteine zur Implementierung der Testspezifikation von Sicherheitsprotokollen dienen. Hiermit werden das Senden und Empfangen von Nachrichten dargestellt. Darüber hinaus können diese Events verwendet werden, um bestimmte Forderungen, sogenannte Claims, zu stellen. Scyther unterscheidet dementsprechend zwischen Send-, Receive- und Claim-Events.

[33] Siehe Unterabschnitt 3.2.2.

[34] Forderung wird für den englischen Begriff „Claim“ verwendet, siehe Unterunterabschnitt 3.2.1.2.

[35] Siehe Unterunterabschnitt 3.2.1.4.

[36] Ungebundene Modellprüfung wird für den englischen Begriff „unbounded (session) model checking“ verwendet.

[37] Siehe Unterunterabschnitt 3.2.1.2.

[38] Siehe Unterunterabschnitt 3.2.1.3.

[39] Unter anderem sei hier AVISPA [BCM18] genannt.

Send-Events:
Das Send-Event wird in Scyther verwendet, um den entsprechenden Nachrichtenaustausch der Protokollspezifikation zu modellieren. Hierbei handelt es sich um jene Nachrichten, die eine Rolle an eine andere Rolle verschickt. Dabei enthalten ist der jeweilige Inhalt der Nachricht, ausgehend von der Protokollspezifikation.

Receive-Events:
In Scyther muss modelliert werden, dass eine Rolle eine Nachricht erhält. Dies geschieht, indem ein Receive-Event genutzt wird, welches denselben Inhalt wie das korrespondierende Send-Event aufweist.

Claim-Events:
Mittels Claim-Events lassen sich Claims einfordern. Das Scyther-Werkzeug analysiert dann, ob diese Claims eingehalten werden oder ob Verletzungen dieser vorhanden sind.

3.2.1.2 Claims

Mittels Claims lassen sich Forderungen an das Protokollverhalten stellen. Die Claims, welche in Scyther verwendet werden können, sollen im Folgenden vorgestellt werden.

Secrecy:
Das Claim Secrecy hat zur Bedeutung, dass ein gegebener Ausdruck beziehungsweise eine Information nicht an einen Angreifer gelangt während der Kommunikation zweier nicht-kompromittierter Rollen [CM12]. Dies soll jedoch insbesondere dann gelten, wenn der Kommunikationskanal beziehungsweise das Netzwerk kompromittiert ist [CM12] und somit beispielsweise das Abhören des Kanals möglich ist. Das Werkzeug Scyther verwendet den Term Secret, um dieses Verhalten auszudrücken.

Aliveness:
Das Aliveness-Claim spezifiziert, dass ein Protokoll einem Kommunikationspartner A garantiert, dass ein anderer Kommunikationspartner B dasselbe Protokoll ausgeführt hat, wenn A einen Durchlauf des Protokolls mit B absolviert hat [Low97]. Hierbei muss jedoch beachtet werden, dass B möglicherweise nicht zugestimmt hat, mit A kommuniziert zu haben [Low97]. In Scyther wird hierfür der Term Alive verwendet.

Weak Agreement:
Als Erweiterung des Aliveness-Claims spezifiziert das Weak Agreement-Claim zusätzlich, dass der Kommunikationspartner B nun zustimmt, dass dieser das Protokoll mit A ausgeführt hat [Low97]. Der Term Weakagree wird in Scyther dafür verwendet.

Non-injective Synchronisation:
Mittels Non-injective Synchronisation wird das Verhalten beschrieben, dass das Senden und Empfangen von Nachrichten, wie im Protokoll spezifiziert, erfolgt und dass dies in der spezifizierten Reihenfolge geschieht [CM12]. Ebenso darf der gesendete Inhalt nicht von dem empfangenen Inhalt abweichen. Dieses Claim wird im Werkzeug Scyther mittels des Terms Nisynch adressiert.

Non-injective Agreement:
Unter Non-injective Agreement wird eine schwächere Form der oben beschriebenen Forderung der Non-injective Synchronisation verstanden. Während Non-injective Synchronisation sowohl fordert, dass die ausgetauschten Daten übereinstimmen und dabei die Reihenfolge der Nachrichten eingehalten wird, wird bei dem Non-injective Agreement nur die Übereinstimmung der Daten gefordert [CM12]. Dies hat insbesondere zur Folge, dass die beiden Kommunikationspartner zustimmen, dass der Inhalt gleich ist [CM12]. Scyther modelliert dieses Verhalten mittels des Terms Niagree.

3.2.1.3 Bounds

Das Werkzeug Scyther ermöglicht es die Anzahl der Zustände beziehungsweise der Protokolldurchläufe durch Grenzen, sogenannte Bounds, einzuschränken. Dies ist möglich, um beispielsweise die zur Berechnung der Verifikation benötigte Zeit zu reduzieren, da eine ungebundene Verifikation einen größeren Aufwand bedeuten kann. Mittels Festlegung einer numerischen Grenze lässt sich die Anzahl der Durchläufe auf diese Zahl begrenzen.

3.2.1.4 Angriffsgraphen

Angriffsgraphen[40] werden in Scyther genutzt, um die von Scyther identifizierten Schwachstellen darzustellen und einen möglichen Angriff darauf zu modellieren. In diesen Graphen werden die Schritte dargestellt, wie diese Schwachstellen zustande kommen. Dabei wird aufgezeigt, welche Rollen beteiligt sind, welche Schritte diese ausführen und welche Inhalte ein Angreifer erhalten kann.

3.2.2 Eingabesprache

Die Implementierung eines Sicherheitsprotokolls beziehungsweise dessen Testspezifikation findet in Scyther durch eine eigene Eingabesprache statt. Die Eingabesprache von Scyther ist angelehnt an die Syntax von C und Java [Cre14a]. Scyther betrachtet die jeweiligen Rollen beziehungsweise Akteure und dessen Interaktionen miteinander einzeln. Damit ist gemeint, dass für jede Rolle eine Spezifikation vorgenommen wird. Dies inkludiert die jeweiligen Events beziehungsweise Nachrichten, an der die jeweilige Rolle beteiligt ist, sowie die Forderungen, die an die jeweiligen Anweisungen gestellt werden. Dementsprechend existiert eine Menge von Rollen, die wiederum eine Menge von Anweisungen besitzen, die sequentiell ablaufen [Cre14a]. In der Eingabesprache werden Kommentare, wie in Java und ähnlichen Programmiersprachen, durch **//** für einzeilige Kommentare und durch **/*** ...***/** für mehrzeilige Kommentare angegeben. Leerzeichen werden in der Eingabesprache weitestgehend ignoriert, können jedoch zur Verbesserung der Lesbarkeit verwendet werden. Leerzeichen zwischen zwei Identifizierern werden benötigt, um diese voneinander zu trennen, jedoch bleiben Leerzeichen, die für Einrückungen oder Ähnliches genutzt werden, unberücksichtigt. Darüber hinaus gilt zu beachten, dass die Sprache zwischen Klein- und Großbuchstaben unterscheidet und daher key und KEY unterschiedliche Identifizierer sind [Cre14a].

[40] Angriffsgraph wird für den englischen Begriff „Attack Graph“ verwendet.

Um ein allgemeines Verständnis dafür zu geben, wie die in dieser Arbeit angegebenen Spezifikationen zu lesen sind und welche Bedeutungen die jeweiligen Anweisungen haben, soll im Folgenden eine Erklärung der jeweiligen Konstrukte erfolgen.

3.2.2.1 Protokoll-Definition

```
protocol name (A, B)
{
  // ...
}
```

Codedarstellung 3.1: Darstellung einer Protokoll-Definition in Scyther: Definiert wird ein Protokoll namens name mit den Rollen A und B.

Die Implementierung einer Testspezifikation eines Protokolls in Scyther beginnt mit der Definition des Protokolls[41]. Wie in der Codedarstellung 3.1 zu sehen, wird diese eingeleitet mittels des Bezeichners protocol, gefolgt von dem Namen des Protokolls (hier: name). Um dem Werkzeug mitzuteilen, welche Rollen definiert werden müssen, folgt die Übergabe einer Liste aller Rollen als n-Tupel[42]. Die Definition der Rollen und des Ablaufs des Protokolls erfolgt in den geschweiften Klammern [Cre14a].

3.2.2.2 Rollen-Definition

```
role A
{
  // ...
}
```

Codedarstellung 3.2: Darstellung einer Rollen-Definition in Scyther: Definiert wird eine (leere) Rolle namens A.

Eine Rolle des Protokolls wird implementiert, indem zunächst der Bezeichner role angegeben wird, gefolgt von dem Namen der Rolle. Die Codedarstellung 3.2 zeigt dies, indem es eine Definition der Rolle A vornimmt. Der Name einer Rolle darf nicht eines der Schlüsselwörter der Eingabesprache von Scyther entsprechen [Cre14a]. Diverse Implementierungen verschiedener Protokolle[43] verwenden Großbuchstaben als Namen für Rollen. Insbesondere werden die Bezeichner I und R für die Initiator-Rolle beziehungsweise für die Responder-Rolle als Konvention verwendet [Cre14a]. Benennungen der Rollen mit C für Client und S für Server sind ebenfalls denkbar.

3.2.2.3 Send-Event

Die Umsetzung des in Unterunterabschnitt 3.2.1.1 beschriebenen Konzeptes des Send-Events geschieht in Scyther durch das send-Schlüsselwort. Ausgehend von der Rolle A als Sender wird das Send-Event in der Rollen-Definition von A spezifiziert.

[41] Eine Ausnahme stellen hierbei die benutzerdefinierten Typen dar, sowie Makros.

[42] Als n-Tupel wird hier die kommaseparierte Liste verstanden: (A, B, C, ..).

[43] Beispielhafte Implementierungen von Sicherheitsprotokollen in Scyther lassen sich unter https://github.com/cascremers/scyther/tree/master/gui/Protocols finden.

```
role A
{
  // ...
  send_1(A, B, n);
  // ...
}
```

Codedarstellung 3.3: Darstellung eines Send-Events in Scyther: Der Sender A versendet eine Nachricht mit dem (nicht näher spezifizierten) Inhalt n an den Empfänger B.

Wie in der Codedarstellung 3.3 zu sehen, nimmt die Send-Funktion Parameter entgegen. Zunächst muss spezifiziert werden, welche Rolle der Sender der Nachricht ist. In diesem Falle ist A der Sender. Daraufhin wird der Emfpänger, in diesem Fall B, angegeben. Weitere Argumente stellen den Inhalt der Nachricht dar. Die Codedarstellung 3.3 stellt hier nur ein Argument (n) dar, jedoch sind weitere möglich. Das Label[44] _1 nach dem Schlüsselwort send bedeutet, dass dies die erste Nachricht des Protokolls ist [Cre14a]. Nachfolgende Nachrichten würden dieses Label entsprechend erhöhen. Somit würde das zweite Send-Event durch send_2 beschrieben werden.

3.2.2.4 Receive-Event

```
role B
{
  // ...
  recv_1(A, B, n);
  // ...
}
```

Codedarstellung 3.4: Darstellung eines Receive-Events in Scyther: Der Empfänger B erhält eine Nachricht mit dem (nicht näher spezifizierten) Inhalt n vom Sender A.

Send-Events haben in der Regel ein korrespondierendes Receive-Event[45] [Cre14a]. Das korrespondierende Receive-Event, zu dem in Unterunterabschnitt 3.2.2.3 beschriebenen Send-Event, wird in der Rolle B definiert. Korrespondierende Send- und Receive-Events müssen mit dem selben Label annotiert werden. Dementsprechend muss recv_1 verwendet werden, wie es die Codedarstellung 3.4 zeigt. Hierbei entspricht die Reihenfolge der Argumente der Argumentenfolge des Send-Events. Darüber hinaus muss dieses Event denselben Inhalt aufweisen.

[44] Unter Label versteht sich hier die Differenzierung der jeweiligen Events durch eine numerische Markierung.

[45] Send-Events müssen keine korrespondierenden Receive-Events besitzen, wenn die Nachricht direkt an einen Angreifer gelangen soll [Cre14a]. Da diese Events keine Relevanz für die Arbeit haben werden, da diese in der Implementierung (siehe Abschnitt 4.3) nicht verwendet werden, werden diese hier nicht näher betrachtet.

3.2.2.5 Claim-Event

```
0 claim_A1(A, Alive);
```

Codedarstellung 3.5: Darstellung eines Claim-Events in Scyther: A fordert mittels des Claims Aliveness.

Für die Nutzung eines Claim-Events in Scyther wird der Befehl claim zur Verfügung gestellt. Als Konvention wird das Claim mit einem Label ausgestattet, welches indiziert, von welcher Rolle dieses Claim stammt, gefolgt von einer fortlaufenden Nummer. Dies dient zur Differenzierung der verschiedenen Claims zwischen den einzelnen Rollen. Als Argument wird zunächst die Rolle angegeben, die diese Forderung stellt. Daraufhin wird jene Forderung angegeben. Jenachdem, ob weitere Argumente folgen müssen, wie beispielsweise bei dem Claim Secrecy, werden diese angegeben. In der Codedarstellung 3.5 wird dieses Verhalten dargestellt, indem die Rolle A Aliveness fordert.

3.2.2.6 Variablen und generierte Werte

```
0 role A
  {
2   fresh nA: Nonce;
    var nB: Nonce;
4 }
```

Codedarstellung 3.6: Darstellung von Variablen und generierten Werten in Scyther: Es wird ein Wert (vom Typ Nonce) in nA generiert und eine Variable nB (vom Typ Nonce) deklariert.

Bei der Implementierung der Rollen kann auf Variablen zurückgegriffen werden. Diese sind, wie in Programmiersprachen, an einen bestimmten Typen gebunden. Im obigen Beispiel (Codedarstellung 3.6) wird eine Variable nB vom Typ Nonce deklariert. Dazu wird das Schlüsselwort var verwendet. Variablen dienen dazu, empfangene Daten zu speichern. Solche Variablen sind lokal, das heißt, sie sind nur verfügbar für die Rolle, in der sie deklariert werden. Variablen können nur einen Wert speichern und können danach nicht überschrieben beziehungsweise geändert werden [Cre14a]. Dementsprechend müssen Werte, die in Variablen geschrieben werden, zunächst durch Receive-Events empfangen und beschrieben werden, bevor sie selbst in Send-Events verwendet werden können. Uninitialisierte Variablen dürfen nicht in Send-Events verwendet werden [Cre14a].

Damit Scyther echte Werte simulieren kann, ermöglicht das Werkzeug die Generierung von Werten. Dadurch wird es ermöglicht Instanzen eines Typs generieren zu lassen, wie beispielsweise einen benutzerdefinierten Typen. Insbesondere ist dies notwendig für Protokolle, welche auf das Generieren von zufälligen Werten angewiesen sind [Cre14a]. Wie die Codedarstellung 3.6 aufzeigt, geschieht die Generierung eines Wertes durch den Bezeichner fresh und der Angabe des Namens der Variablen (hier: nA), die den Wert enthalten soll. Danach erfolgt die Angabe des Typs.

3.2.2.7 Symmetrische Verschlüsselung

```
0 {n}k(A, B)
```

Codedarstellung 3.7: Darstellung der Nutzung symmetrischer Verschlüsselung in Scyther. Hierbei wird der (nicht näher spezifizierte) Inhalt n mit Hilfe des symmetrischen Schlüssels von A und B verschlüsselt.

In Scyther erfolgt das Verschlüsseln des Inhalts von Nachrichten, indem der Inhalt der Nachricht in geschweiften Klammern dargestellt und anschließend der Schlüssel angehängt wird. In der Codedarstellung 3.7 wird der Inhalt n mittels des symmetrischen Schlüssels von A und B verschlüsselt[46]. Es wird davon ausgegangen, dass es sich bei dem Schlüssel k(A, B) um einen Langzeit-Schlüssel handelt, welchen sich die Rollen A und B teilen [Cre14a].

3.2.2.8 Asymmetrische Verschlüsselung

```
0 {n}pk(A)
```

Codedarstellung 3.8: Darstellung der Nutzung asymmetrischer Verschlüsselung in Scyther. Hierbei wird der (nicht näher spezifizierte) Inhalt n mit Hilfe des öffentlichen Schlüssels von A verschlüsselt.

Im Gegensatz zur symmetrischen Verschlüsselung verwendet die asymmetrische Verschlüsselung zwei Schlüssel - den privaten und den öffentlichen Schlüssel. Dementsprechend findet sich in Scyther eine Implementierung von diesem Verhalten. Es wird dabei angenommen, dass eine Rolle einen Langzeit-Schlüsselpaar des privaten und öffentlichen Schlüssels besitzt [Cre14a]. Der private Schlüssel der Rolle A wird durch sk(A) für secretkey von A und der öffentliche Schlüssel durch pk(A) für publickey von A angegeben. Der Inhalt der Nachricht wird wie in Unterunterabschnitt 3.2.2.7 beschrieben, dargestellt. Lediglich der Schlüssel (k(A, B)) wird durch den öffentlichen Schlüssel (pk(A)) von A ersetzt, wie es in der Codedarstellung 3.8 beschrieben wird[47]. Diese Nachricht kann nur von Rollen entschlüsselt werden, die den korrespondierenden privaten Schlüssel (sk(A)) besitzen [Cre14a]. Dabei wird der Entschlüsselungsvorgang implizit vorgenommen. Das heißt, dass über die Eingabesprache von Scyther nicht definiert werden kann, dass die verschlüsselte Nachricht mittels des privaten Schlüssel entschlüsselt wird. Dies wird automatisch vom Werkzeug übernommen, sofern die Rolle den privaten Schlüssel besitzt.

3.2.2.9 Signatur

```
0 {n}sk(A)
```

Codedarstellung 3.9: Darstellung der Nutzung einer Signatur in Scyther. Es wird der (nicht näher spezifizierte) Inhalt n mit Hilfe des privaten Schlüssels von A signiert.

[46] Hierbei wird keine vollständige Nachricht dargestellt, um den Fokus auf die symmetrische Verschlüsselung des Inhalts zu legen.

[47] Hierbei wird keine vollständige Nachricht dargestellt, um den Fokus auf die asymmetrische Verschlüsselung des Inhalts zu legen.

Signaturen finden ebenfalls Anwendung in Scyther. Sie werden genutzt, um Inhalte von Nachrichten zu signieren. Dazu wird der pivate Schlüssel des Kommunikationspartners verwendet, welcher diesen Inhalt signieren möchte, wie es in der Codedarstellung 3.9 gezeigt wird. Hierbei signiert die Rolle A den Inhalt n mittels des privaten Schlüssels sk(A).

3.2.2.10 Benutzerdefinierte Typen

```
0 usertype String;
```

Codedarstellung 3.10: Darstellung der Nutzung eines benutzerdefinierten Typs in Scyther. Es wird ein Typ namens String definiert.

Benutzerdefinierte Typen können in Scyther verwendet werden, um dem Werkzeug mitzuteilen, dass Deklarationen von Variablen des Typs nur durch Terme des Typs instanziiert werden können und nicht durch andere Typen instanzierbar sind. Durch die Angabe von benutzerdefinierten Typen kann Scyther die Berechnungen performanter gestalten [Cre14a]. Benutzerdefinierte Typen werden in Scyther mit dem Schlüsselwort usertype eingeleitet. Daraufhin wird der Name des Typs festgelegt. Die Codedarstellung 3.10 stellt dieses Verhalten dar. Hierbei wird der Typ String definiert[48]. Die Instanziierung einer Variablen von diesem Typ erfolgt wie in Unterunterabschnitt 3.2.2.6 beschrieben, da sich benutzerdefinierte Typen wie vordefinierte Typen verhalten.

3.2.2.11 Makros

```
0 macro term = {B, key}pk(A);
```

Codedarstellung 3.11: Darstellung der Nutzung eines Makros in Scyther. Es wird ein Makro term definiert, welches als Abkürzung des dahinterstehenden Ausdrucks dient.

Die Verwendung von Makros in Scyther dient insbesondere der Schaffung der Übersichtlichkeit, da hiermit lange Ausdrücke abgekürzt werden können. Makros ersetzen das Vorkommen des Namens des Makros (hier: term) durch den eigentlichen Ausdruck des Makros. Die Makros sind für das gesamte Dokument gültig [Cre14a].

3.2.2.12 Hashfunktion

```
0 hashfunction H;
// ..
2   H(n);
// ..
```

Codedarstellung 3.12: Darstellung der Nutzung einer Hashfunktion in Scyther. Es wird eine Hashfunktion mit dem Namen H definiert.

[48] Die Definition eines benutzerdefinierten Typs führt nicht dazu, dass Scyther weiß, wie dieser Typ funktioniert.

Das Werkzeug Scyther bietet die Möglichkeit Hashfunktionen zu definieren, wie es in Zeile eins der Unterunterabschnitt 3.2.2.12 gezeigt. Diese dienen insbesondere dazu, einen Hash zu generieren, welcher von niemandem umgekehrt werden kann [Cre14a]. Damit alle Rollen diese Funktion nutzen können, wird die Definition außerhalb des Protokolls vorgenommen. Die Verwendung erfolgt dann, wie es in Zeile drei der Unterunterabschnitt 3.2.2.12 erkennbar ist. Hierbei wird der Term n als Parameter für die Hashfunktion verwendet.

3.3 Analyzing the IETF ACE-OAuth-Protocol

In seiner Veröffentlichung [Tsc18] analysiert H. Tschofenig das CTP. Dafür gibt der Autor zunächst einen Überblick über das besagte Protokoll und das Framework, in welchem dieses spezifiziert wurde. Insbesondere wird beschrieben, dass die Analyse von Sicherheitsprotokollen mit Werkzeugen, wie beispielsweise Scyther, eine gewisse Abstraktion voraussetzt. Sollte diese Abstraktion fehlerhaft sein, könnte dies das Auffinden von Fehlern verhindern.

Für seine Analyse trifft H. Tschofenig zunächst ein paar Annahmen. Hierin wird beschrieben, dass der Access Token in Klartext versendet wird und dieser als Identifizierer zu betrachten ist, welcher auf den Client zurückführt. Darüber hinaus wird angenommen, dass die Token Introspection verschlüsselt stattfindet. Ein weiterer Aspekt, welcher in dieser Analyse angenommen wird, ist, dass sowohl der Client als auch der Resource Server je einen Langzeit-Schlüssel mit dem Authorization Server besitzen.

H. Tschofenig stellt des Weiteren die beiden Testspezifikationen vor, die für das CTP entwickelt wurden. Eine Testspezifikation wurde für das Scyther-Werkzeug [Cre14b] erstellt und eine für das AVISPA-Werkzeug [ABB+05]. In seiner Veröffentlichung zeigt der Autor eine Verwundbarkeit des Protokolls auf, welches zurückgeführt wird auf eine mangelnde Authentifizierung des Clients gegenüber dem Authorization Server. Es wird darüber hinaus mittels eines, durch Scyther ausgegebenen, Angriffsgraphen dargestellt, wie einer der identifizierten Angriffe auf das Client-Token-Protokoll aussieht. Hierin wird dargelegt, dass der Client mit einem bösartigen Resource Server interagiert und nicht mit dem eigentlichen Resource Server.

In seiner Veröffentlichung fordert H. Tschofenig daher, entweder die grundlegende Überarbeitung oder die Entfernung des CTPs aus der ACE-OAuth-Spezifikation.

Aus dieser Veröffentlichung gehen Testspezifikationen für das CTP [SSW+17] des ACE-OAuth-Frameworks[49] hervor, die mittels AVISPA und Scyther analysiert werden. Hierbei zeigt sich eine Ähnlichkeit hinsichtlich des zugrunde liegenden Frameworks, da dieses ebenfalls die Grundlage für das C3DC-Profil [GBB18b] bildet. Dementsprechend kann die Scyther-Spezifikation genutzt werden, um zu prüfen, ob diese dem aktuellen ACE-OAuth-Framework [SSW+19] standhält und ob diese eine Erweiterung auf das zu untersuchende C3DC-Profil ermöglicht.

[49] Hierbei handelt es sich um die Versionsnummer -10 der ACE-OAuth-Spezifikation.

3.4 A Formal Model for Delegated Authorization of IoT Devices Using ACE-OAuth

Die Autoren L. Arnaboldi und H. Tschofenig nutzen das Tamarin-Werkzeug [MSCB13], um die symbolische Analyse des ACE-OAuth-Frameworks in ihrer Veröffentlichung [AT19] vorzunehmen. Um dieses Ziel zu erreichen, wird zunächst die ACE-OAuth-Spezifikation formalisiert. Hierbei werden die grundlegenden Sicherheitsanforderungen und Sicherheitsziele aus der Spezifikation extrahiert und formal definiert. Dies resultiert in einem ausführbaren Modell der ACE-OAuth-Spezifikation, welches als Grundlage dienen soll, um verschiedene weitere Konfigurationen der ACE-OAuth-Spezifikation testen zu können. Der Analyse mittels Tamarin liegt zugrunde, dass die Nachrichten durch Terme repräsentiert werden und dass das Angreifermodell dem Dolev-Yao-Modell entspricht. Tamarin nutzt ein Modell des Sicherheitsprotokolls, worin die jeweiligen Aktionen der Rollen definiert werden. Dies geschieht unter Verwendung einer formalen Sprache. Kommt das Tamarin-Werkzeug zur Terminierung, so gibt dieses entweder einen möglichen Angriff, oder einen Beweis für das Nichtvorhandensein eines Angriffes aus. Des Weiteren stellen die Autoren das ACE-OAuth-Framework vor. Die Autoren machen deutlich, dass für die Analyse eines Sicherheitsprotokolls ein gewisses Maß an Abstraktion vorgenommen werden muss, wobei dies mit Bedacht geschehen muss. Um das zuvor genannte Modell verifizieren zu können, müssen bestimmte Sicherheitsanforderungen getroffen werden. Diese werden von den Autoren, unter Berücksichtigung der Veröffentlichung von G. Lowe [Low97], beschrieben. Hierfür werden die Eigenschaften Aliveness, Weak Agreement, Non-injective Agreement[50] und Injective Agreement aufgezeigt. Unter Injective Agreement wird hierbei eine Erweiterung des Non-injective Agreements verstanden. Dies hat zur Folge, dass zusätzlich gefordert wird, dass für jeden Protokolldurchlauf einer Rolle eine eindeutige Instanz des Protokolldurchlaufs der anderen Rolle existiert, die zusammengehören. Mit diesen Eigenschaften wird es ermöglicht, eine Aussage darüber treffen zu können, ob der Protokolldurchlauf nicht verändert wurde und somit sich entsprechend der Spezifikation verhält.

Das ACE-OAuth-Framework wird daraufhin in drei Phasen aufgeteilt. Hierbei werden die Phasen des Token Grants, des Resource Requests und der Token Validierung unterschieden. In der Phase des Token Grants wird angemerkt, dass ein Token, welcher ungeschützt ist, einem Angreifer Zugriff auf die Resource gewähren würde. Dementsprechend wird dies als wichtiger Punkt betrachtet, den es effizient zu schützen gilt. Hierfür fordert das ACE-OAuth-Framework, dass die Kommunikation zwischen dem Authorization Server und dem Client geschützt wird hinsichtlich der Vertraulichkeit. Dementsprechend wird verlangt, dass für alle Nachrichten zwischen diesen beiden Rollen, ein Angreifer den Inhalt der Nachrichten nicht wissen kann, ohne die jeweiligen Schlüsselmaterialien zu besitzen. Zwischen dem Client und dem Resource Server existiert während des Resource Requests zunächst kein verschlüsselter Kanal. Daher wird die Kommunikation im formalen Modell entsprechend als ungeschützter Kanal modelliert. Für den Resource Request muss der Resource Server die Eigenschaft des Injective

[50] Die Eigenschaften Aliveness, Weak Agreement und Non-injective Agreement werden hier nicht näher aufgeführt, da diese unter Unterunterabschnitt 3.2.1.2 beschrieben werden.

Agreements an den Client garantieren. Die dritte Phase wird in der Veröffentlichung durch die Token Introspection beschrieben. Da hierbei der Authorization Server dem Resource Server bei der Authentifizierung unterstützt, ist es notwendig, dass dies geschützt geschieht. Hierbei ergeben sich dieselben Anforderungen, wie zuvor beschrieben.

Daraufhin wird ein Profil des ACE-OAuth-Frameworks definiert, anhand dessen die Analyse durchgeführt wird. Hierin wird einzig asymmetrische Verschlüsselung eingesetzt und es wird das Operation Bundle definiert, welches den eigentlichen Resource Request, einen Zeitstempel und den Access Token zu einem Bündel zusammenfügt. Ebenfalls wird in diesem Szenario angenommen, dass der Resource Server keine Introspection durchführt, sondern den Token lokal validieren kann. Die Autoren verdeutlichen darüber hinaus, dass nur eine gewisse Anzahl an Gesichtspunkten aus dem ACE-OAuth-Framework genutzt werden können, um eine detailierte Analyse durchführen zu können.

L. Arnaboldi und H. Tschofenig gehen darauf ein, dass das Finden eines Angriffs auf ein Protokoll nur ein einziges unsicheres Szenario benötigt, wohingegen die Verifikation wesentlich schwieriger ist, da hierbei alle möglichen Szenarien betrachtet werden müssen. Bei der Analyse gehen die Autoren erneut auf die Anforderungen ein, welche die Spezifikation erfüllen muss. Für den Token Grant ergibt sich hierbei die Geheimhaltung der Anfrage, sowie dass die Eigenschaft des Injective Agreements für diese Anfrage eingehalten wird. Zuletzt muss die Eigenschaft des Injective Agreements für den Token eingehalten werden. Bei dem Resource Request wird die Geheimhaltung der Resource, sowie die Eigenschaft des Injective Agreements für die Resource gefordert. Ausgehend von einer naiven Implementierung des beschriebenen Profils, ergeben sich bei der Analyse verschiedene Verwundbarkeiten. Die Ursachen werden von den Autoren auf Entscheidungen in der Spezifikation zurückgeführt, die sich einfach beheben lassen. Des Weiteren wird die Aussage getroffen, dass jedes Profil Unterschiede hervorbringt, die jeweils unterschiedliche Verwundbarkeiten zu diesem Framework hinzufügen könnten. Um dies zu vermeiden, müsse eine gewisse Menge an Anforderungen für jedes dieser Profile spezifiziert werden. Die Autoren diskutieren darüber hinaus, warum der Token obgleich der vorgenommenen Anpassungen, dennoch einem Angreifer offenbart werden kann, da sich dies auf eine Verwundbarkeit im Resource Request zurückführen ließe.

Abschließend wird das Ergebnis der Veröffentlichung betrachtet. Hierbei geht hervor, dass die Autoren einen formalen Beweis erbracht haben, dass das ACE-OAuth-Framework sicher ist, sofern dies entsprechend der Spezifikation beziehungsweise der Anforderungen implementiert wird. Darüber hinaus wird vorgeschlagen, das entwickelte Modell als Grundlage für die Analyse der Erweiterungen der ACE-OAuth-Architektur zu nutzen.

Für diese Thesis wird die Veröffentlichung von L. Arnaboldi und H. Tschofenig als Möglichkeit angesehen, die Ergebnisse in Vergleich mit den Ergebnissen dieser Thesis zu setzen.

3.5 Avoiding Gaps in Authorization Solutions for the Internet of Things

S. Gerdes et al. beschreiben in ihrer Veröffentlichung [GBB18a] verschiedene Aufgaben, welche die teilnehmenden Akteure eines Protokolls auszuführen haben, um geschützt miteinander kommunizieren zu können. Darüber hinaus wird beschrieben, welche Konsequenzen aus der fehlerhaften Ausführung entstehen können. Nach S. Gerdes et al. können Lücken im Autorisierungsprozess auftreten, wenn Anforderungen an ausgetausche Daten oder Ähnliches vorhanden sind, die in der Spezifikation nicht explizit definiert sind. Ohne eine entsprechende Richtlinie würde ein Entwickler dazu gezwungen sein, eigene Annahmen der Sachverhalte treffen zu müssen, die zu Verwundbarkeiten führen können, wenn dieser Prozess fehlerhaft gestaltet wird. Des Weiteren könnte der Autorisierungsprozess kompromittiert werden, wenn die ausgetauschten Informationen nicht in jedem Schritt geschützt werden.

Dementsprechend wird in dieser Veröffentlichung eine Checkliste vorgestellt, welche die notwendigen Schritte darstellt, die zur Sicherung der Kommunikation von den jeweiligen Akteuren ausgeführt werden müssen, um im Interesse der jeweiligen Overseeing Principal zu arbeiten und deren Sicherheitsziele zu verfolgen. Zunächst muss dafür ein Akteur feststellen können, welche Berechtigungen ein anderer Akteur besitzt, bevor Nachrichten ausgetauscht werden können. Wenn dieser Schritt ausgelassen werden würde, könnte dies nach S. Gerdes et al. dazu führen, dass unautorisierte Daten akzeptiert werden und vertrauliche Daten an unautorisierte Akteure gelangen könnten.

Ein weiterer Teil der Checkliste ist der Integritätsschutz der Informationen. Hierbei darf es einem Angreifer nicht möglich sein, Änderungen an diesen Informationen vornehmen zu können. Daraufhin folgt die Aufgabe zur Aktualität der Berechtigungen. Diese Berechtigungen müssen regelmäßig erneuert werden, da Berechtigungen nicht unendlich lang gültig sein dürfen. Werden diese Berechtigungen nicht regelmäßig erneuert, kann dies dazu führen, dass gegebenenfalls entscheidende Informationen fehlen, wenn eine Entscheidung getroffen werden muss. Im vierten Schritt der Checkliste müssen die erhaltenen Informationen evaluiert werden. Darüber hinaus wird eine Aufgabe beschrieben, indem ein Akteur feststellen muss, ob die Informationen durch einen Akteur ausgestellt wurden, der vom jeweiligen Overseeing Principal autorisiert ist.

Eine weitere Aufgabe ist die Feststellung, ob diese Informationen überhaupt an den jeweiligen Akteur gerichtet sind. Wird diese Aufgabe nicht durchgeführt, so könnte ein Angreifer den Akteur dazu bewegen, Informationen anzunehmen, die für einen anderen Akteur bestimmt sind. Der siebte Schritt der Checkliste besteht darin, dass der Akteur prüfen muss, ob diese Informationen von einem Akteur stammt, mit dem derzeit kommuniziert wird. Die letzte Aufgabe der Checkliste beschreibt die Notwendigkeit, alle erhaltenen Informationen zu einer Angelegenheit evaluieren zu müssen.

Nach der Vorstellung der Checkliste, zeigen S. Gerdes et al. die Anwendung dieser Liste auf das DTLS-Profil des ACE-OAuth-Frameworks. Hierbei wird untersucht,

ob dieses Profil die oben genannten Aufgaben ausführt. Zunächst wird betrachtet, ob der Authorization Server diese Aufgaben in Bezug auf den Resource Server ausführt, bevor dies auf den Client angewendet wird.

Zuerst muss der Authorization Server überprüfen, ob der Resource Server autorisiert ist, den Access Token zu erhalten. Der Access Token wird als vertraulich behandelt, da dieser gegebenenfalls den symmetrischen Schlüssel enthalten kann. Dementsprechend fordert die Autorenschaft, dass das DTLS-Profil explizit den Schutz der Vertraulichkeit des Access Tokens vorschreibt, um die Ausführung dieser Aufgabe zu ermöglichen.

Für die darauffolgende Aufgabe beschreiben S. Gerdes et al., dass das ACE-OAuth-Framework den Integritätsschutz des Access Tokens fordert, jedoch das DTLS-Profil nicht spezifiziert, wie dies erreicht wird. Daher sollten Änderungen vorgenommen werden, die diese Unklarheiten beseitigen.

Damit der Resource Server die Validität des Access Tokens überprüfen kann, stellt das ACE-OAuth-Framework verschiedene Mechanismen zur Verfügung. Darunter zu finden, ist die Angabe eines Datums, ab welchem Zeitpunkt dieser Access Token ungültig wird. Hierbei wird jedoch vorausgesetzt, dass der Authorization Server und der Resource Server zeitlich synchronisiert sind. Nach der Autorenschaft wird hierbei jedoch nicht deutlich, welcher Token der aktuellste Token ist, sollten mehrere Token vorhanden sein. Eine weitere Möglichkeit für den Resource Server ist die Nutzung der Introspection. Dieser Ansatz bringt zwei Probleme hervor. Zum einen wird durch das ACE-OAuth-Framework nicht spezifiziert, wie der Resource Server feststellt, ob die erhaltenen Informationen aktuell sind und zum anderen ermöglicht die Nutzung der Introspection eine Verwundbarkeit des Resource Servers. Der letzte Ansatz hierfür beschreibt die Verwendung der Nutzung von aufeinanderfolgenden Sequenznummern. Jedoch wird dieser Ansatz durch die Autorenschaft als unbenutzbar beschrieben.

Für die vierte Aufgabe der Checkliste verdeutlichen S. Gerdes et al., dass das ACE-OAuth-Framework deutlicher spezifizieren muss, wie der Resource Server auf Access Tokens reagieren muss, die nicht den Anforderungen der Spezifikation entsprechen. Hierbei wird insbesondere darauf eingegangen, dass fehlende Informationen zu Verwundbarkeiten führen können, wenn diese in Implementierungen falsch umgesetzt werden. Darüber hinaus wird angemerkt, dass der Resource Server ungeschützte Tokens nicht annehmen darf und dies in dem ACE-OAuth-Framework spezifiziert werden sollte.

Der Resource Server muss prüfen, ob der Authorization Server autorisiert ist, dem Resource Server die jeweiligen Informationen zur Verfügung zu stellen. Das ACE-OAuth-Framework nimmt dafür an, dass der Resource Server beim Authorization Server registriert wurde und gegenseitig Schlüsselmaterialien ausgetauscht wurden. Nach S. Gerdes et al. macht das Framework jedoch keine Aussagen darüber, ob dieses Schlüsselmaterial den Resource Server in die Lage versetzt, zu prüfen, ob ein Access Token vom Authorization Server ausgestellt wurde.

Um die sechste Aufgabe zu erfüllen, muss der Resource Server sicherstellen, dass dieser der angedachte Empfänger des Access Tokens ist. Dafür spezifziert das ACE-OAuth-Framework, dass der Resource Server jeden Token ablehnen muss,

für den dieser nicht der angedachte Empfänger ist, wodurch der Resource Server dieser Aufgabe nachkommt.

Des Weiteren muss der Resource Server prüfen, ob der Client der Besitzer des Access Tokens ist. Die Autorenschaft sagt hierzu aus, dass das ACE-OAuth-Framework nicht spezifziert, dass diese Prüfung durchgeführt werden muss und weisen daraufhin, dass dies behoben werden sollte. Für das DTLS-Profil jedoch wird beschrieben, dass diese Aufgabe in dem Profil ausgeführt wird, da die Anfragen des Clients nur dann autorisiert sind, wenn diese über eine geschützte Verbindung versendet werden.

Für den Client kann der Authorization Server zusätzliche Informationen an die Antwort des Token Requests anhängen, die als RS Information[51] bezeichnet werden. Das ACE-OAuth-Framework nimmt an, dass der Client beim Authorization Server registriert wurde und dass dabei gegenseitig Schlüsselmaterialien ausgetauscht wurden. Dennoch wird in dem Framework nicht erwähnt, ob der Authorization Server vom Resource Owner Informationen darüber erhalten hat, ob der Client autorisiert ist.

In der Anfrage an den Authorization Server spezifiziert der Client, welchen Resource Server dieser ansprechen möchte. Allerdings wird im ACE-OAuth-Framework nicht definiert, wie der Resource Server identifiziert wird. Daher macht die Autorenschaft deutlich, dass das Framework definieren muss, wie der Client den Resource Server in seiner Anfrage spezifziert, da der Client sonst möglicherweise mit dem falschen Resource Server kommuniziert.

Des Weiteren beschreiben S. Gerdes et al., dass die Spezifizierung des angedachten Empfängers durch den Authorization Server geschehen kann, indem die RS Information zunächst mit dem Schlüsselmaterial des Clients verschlüsselt wird, bevor der Authorization Server diese mittels eines Integritätsschutzes versieht. Geschieht dies nicht in der beschriebenen Reihenfolge, so wäre die Angabe des angedachten Empfängers nicht geschützt und könnte modifiziert werden.

Der Client muss sicherstellen, dass die Informationen zur Kommunikation mit dem Resource Server aktuell gehalten werden, um Schutz bieten zu können. Allerdings spezifiziert das ACE-OAuth-Framework nicht, wie dies erreicht wird. Darüber hinaus wird die Aussage getroffen, dass das Framework nicht spezifiziert, wie der Client alte von neuen RS Information unterscheidet und somit ein Angreifer den Client dazu bewegen könnte, veraltete Informationen zu nutzen.

Um die fünfte Aufgabe ausführen zu können, muss der Client überprüfen, ob der Authorization Server autorisiert ist, die RS Information auszustellen. Das ACE-OAuth-Framework schlägt hierfür eine fest implementierte Liste von autorisierten Authorization Servern vor, welches durch die Autorenschaft jedoch nicht als empfehlenswert angesehen wird, da die Autorisierungs-Informationen unendlich lange gültig sind, sollten diese nicht aktualisiert werden. Darüber hinaus definiert dieses Framework nicht, dass diese Liste von der Requesting Party spezifiziert werden müsste. Der Autorenschaft zu Folge sollte der Client in der Lage sein,

[51] Ausgehend von der Erklärung in dieser Veröffentlichung [GBB18a] entsprechen die RS Information den Access Information, wie in Unterabschnitt 2.1.10 beschrieben.

diese Aufgabe zu bewältigen, wenn die zuvor beschriebenen Probleme behoben werden.

Eine weitere Aufgabe des Clients ist die Überprüfung, ob der Client der angedachte Empfänger der RS Information ist. Wie zuvor beschrieben, kann dies nur dann festgestellt werden, wenn diese Informationen zunächst verschlüsselt und dann mit einem Integritätsschutz versehen werden. Nach S. Gerdes et al. darf der Client diese Informationen nur dann annehmen, wenn diese Anforderungen erfüllt werden. Dies wird im ACE-OAuth-Framework jedoch nicht deutlich dargelegt.

Um die letzte Aufgabe bewältigen zu können, muss der Client überprüfen, ob der Resource Server das Schlüsselmaterial nutzen kann. Es wird hierbei nicht deutlich, wie der Client feststellt, ob der Resource Server autorisiert ist.

In ihrer abschließenden Diskussion beschreibt die Autorenschaft, dass die Behebung der identifizierten Lücken dazu beitragen kann, dass die Interessen der Overseeing Principals eingehalten werden. Dabei wird die Aussage getroffen, dass wichtige Änderungen getroffen werden müssen hinsichtlich der RS Information, um diese zur sicheren Authentifizierung des Resource Servers gestalten zu können. Im Internet of Things sollen Geräte autonom agieren können und sollen sich nicht auf den Eingriff der Overseeing Principals verlassen müssen, um zu entscheiden, auf welche anderen Geräte diese zugreifen dürfen. Allerdings können eingeschränkte Geräte auch nicht gewaltige Mengen an Regeln zur Autorisierung speichern.

Daher wird von der Autorenschaft die Verwendung einer Architektur vorgeschlagen, die es den eingeschränkten Geräten ermöglicht, ein weniger eingeschränktes Gerät als Helfer zu verwenden.

S. Gerdes et al. beschreiben abschließend, dass einige der vorgestellten Lücken durch eine Überarbeitung des ACE-OAuth-Frameworks und des DTLS-Profils geschlossen werden können, sowie das manche Teile der Spezifikationen dazu führen, dass diese Spezifikation nur auf bestimmte Szenarien anwendbar ist.

Die Untersuchung des DTLS-Profils in dieser Veröffentlichung stellt Aufgaben vor, die hinsichtlich einer sicheren Kommunikation ausgeführt werden müssen. Hierbei kann davon ausgegangen werden, dass diese auch in dem C3DC-Profil berücksichtigt werden müssen. Daher bietet diese Veröffentlichung Annahmen und Ergebnisse, die in dieser Thesis betrachtet werden sollten.

4 Entwurf und Implementierung einer Scyther-Spezifikation

In den folgenden Sektionen wird zunächst diskutiert, ob die Implementierung des CTPs in Scyther der aktuellen Spezifikation entspricht. Daraufhin werden die Annahmen und Erwartungen für die Analyse des C3DC-Profils getroffen, bevor die Implementierung des Profils vorgestellt wird. Abschließend werden die Ergebnisse evaluiert und Lösungsansätze vorgeschlagen.

4.1 Implementierung des CTPs in Scyther

Obowhl das CTP [SSW+17] in der elften Version[52] der ACE-OAuth-Spezifikation [SSW+18] entfernt wurde, lassen sich Vergleiche mit dem Protokolfluss des aktuellen ACE-OAuth-Frameworks anstellen. Dafür wird das Beispiel der ACE-OAuth-Spezifikation [SSW+19, Seite 72 ff.] herangezogen, welches ähnliche Charakteristiken aufweist. Hierbei wird sowohl in dem CTP, als auch in dem Beispiel der ACE-OAuth-Spezifikation davon ausgegangen, dass es dem Client nicht möglich ist, den Authorization Server zum Zeitpunkt des Requests zu erreichen, aufgrund einer limitierten Verbindung. Darüber hinaus wird zur Validierung des Access Tokens die Token Introspection genutzt.

```
0 usertype SessionKey;

2 protocol clienttoken (C, RS, AS)
  {
4 }
```

Codedarstellung 4.1: Implementierung des CTPs in Scyther nach H. Tschofenig [Tsc18].

Die Implementierung des CTPs nach H. Tschofenig [Tsc18] definiert zunächst einen benutzerdefinierten Typ namens SessionKey, wie in der Codedarstellung 4.1 gezeigt. Dieser wird benutzt, um das auszutauschende Schlüsselmaterial zwischen den Rollen zu repräsentieren. Anschließend werden die drei Rollen definiert, die an dem Protokoll teilnehmen. Dies sind der Client (C), der Resource Server (RS) und der Authorization Server (AS), wie in Abschnitt 2.2 und Abschnitt 2.5 beschrieben.

Da die aktuelle Spezifikation des ACE-OAuth-Frameworks [SSW+19, Seite 72 ff.] hier dieselben Rollen definiert, wie im CTP beschrieben, stimmt die Scyther-Spezifikation, wie in der Codedarstellung 4.1 beschrieben, überein.

[52] Hierbei handelt es sich um die Versionsnummer -10 der ACE-OAuth-Spezifikation.

4.1.1 Rolle: Client

```
role C    % C: Client
{
  var key: SessionKey;

  send_1(C, RS, C);
  recv_4(RS, C, {key}k(C, AS));

  claim_C1(C, Secret, key);
  claim_C3(C, Alive);
  claim_C4(C, Weakagree);
  claim_C5(C, Niagree);
  claim_C6(C, Nisynch);
}
```

Codedarstellung 4.2: Implementierung der Rolle des Clients des CTPs in Scyther nach H. Tschofenig [Tsc18].

Die Rolle des Clients in der Codedarstellung 4.2 wird zunächst beschrieben durch die Deklarierung der Variable key, die im weiteren Verlauf des Protokolls den Client-Token beziehungsweise das Schlüsselmaterial beinhalten wird. Im ersten Schritt sendet der Client den Access Token, den der Client in einem vorherigen Schritt mit dem Authorization Server ausgehandelt hat, an den Resource Server. Hierbei wird dieser Access Token durch C dargestellt und nicht verschlüsselt. Der vierte Schritt besteht darin, dass der Client vom Resource Server den Client-Token beziehungsweise das Schlüsselmaterial erhält. Diese Nachricht wurde zwischen dem Client und dem Authorization Server verschlüsselt.

Während im CTP davon ausgegangen wird, dass der Access Token für den Client schon vor dem eigentlichen Request an den Resource Server ausgehandelt wurde, muss dieser Schritt für die aktuelle Version in der Implementierung in Scyther explizit berücksichtigt werden. Dies ist notwendig, da die aktuelle Version das Schlüsselmaterial[53] für den Client nicht über den Resource Server an den Client weitergibt, sondern dieser direkt zwischen dem Client und dem Authorization Server ausgehandelt und ausgetauscht wird [SSW+19]. Dementsprechend ist die Scyther-Spezifikation des CTPs hier nicht korrekt. Um dies zu modellieren, müsste also zunächst eine Anfrage an den Authorization Server gerichtet werden, um den Access Token anzufragen. Darüber hinaus kann der Client angeben, welchen Resource Server er mit seiner Anfrage adressieren möchte [SSW+19, Seite 65]. Als Antwort darauf müsste der Authorization Server den Access Token, sowie das Schlüsselmaterial für den Client, an den Client senden. Der Request an den Resource Server kann dann ähnlich erfolgen, wie in dem send_1-Event der Analyse des CTPs [Tsc18]. Hierbei muss jedoch in der aktuellen Version des ACE-OAuth-Frameworks beachtet werden, dass der Access Token zwischen dem Authorization Server und dem Resource Server verschlüsselt werden muss, sodass nur der Resource Server in der Lage ist, diesen zu entschlüsseln [SSW+19]. Zusätzlich macht es die aktuelle Version notwendig, dass der Authorization Server den angedachten Empfänger, hier also den Resource Server, in dem Access Token angeben muss

[53] Im CTP entspricht dies dem Client-Token.

[SSW+19, Seite 41]. Der letzte Schritt (recv_4) stimmt dementsprechend nicht mit der Scyther-Spezifikation in der Codedarstellung 4.2 überein und müsste angepasst werden. Dies insofern, als dass der Client kein Schlüsselmaterial, also insbesondere keinen Client-Token, über den Resource Server erhält.

Hinsichtlich der Claims lassen sich keine Differenzen feststellen. Hierbei muss weiterhin gelten, dass das ausgetauschte Schlüsselmaterial geheim bleibt, sowie, dass die Beteiligten ebenfalls einen Durchlauf des Protokolls durchgeführt haben. Darüber hinaus müssen die weiteren Claims eingehalten werden, um die Einhaltung der Nachrichten, sowie die Verständigung über die ausgetauschten Daten vornehmen zu können.

4.1.2 Rolle: Resource Server

```
0  role RS   %R: Resource Server
   {
2    var key: SessionKey;

4    recv_1(C, RS, C);
     send_2(RS, AS, {C}k(RS, AS));
6    recv_3(AS, RS, {key, {key}k(C, AS)}k(RS, AS));
     send_4(RS, C, {key}k(C, AS));
8
     claim_RS1(RS, Secret, key);
10   claim_RS3(RS, Alive);
     claim_RS4(RS, Weakagree);
12 }
```

Codedarstellung 4.3: Implementierung der Rolle des Resource Servers des CTPs in Scyther nach H. Tschofenig [Tsc18].

Da der Resource Server ebenfalls den Client-Token beziehungsweise das Schlüsselmaterial erhält, muss dieser auch eine Variable key deklarieren. Im ersten Schritt der Implementierung des CTPs in der Codedarstellung 4.3 erhält der Resource Server den Access Token unverschlüsselt von dem Client. Daraufhin sendet der Resource Server diesen Access Token verschlüsselt an den Authorization Server, um die Token Introspection zu nutzen. Der dritte Schritt zeigt, dass der Resource Server das notwendige Schlüsselmaterial für sich und den Client-Token für den Client verschlüsselt erhält. Anschließend sendet der Resource Server den Client-Token, welcher durch den Authorization Server verschlüsselt wurde, an den Client.

Ausgehend von der zuvor beschriebenen Änderung der Anfrage eines Access Tokens beim Authorization Server, würde sich hier zunächst die Reihenfolge beziehungsweise die Nummerierung der Events ändern. Dabei würde der Access Token im ersten Receive-Event, wie zuvor beschrieben, geändert werden müssen. Hierbei enthält der Access Token das Schlüsselmaterial nicht selbst, sondern nur einen Identifizierer, der das Schlüsselmaterial referenziert [SSW+19, Seite 72 ff.]. Ebenso würde der Introspection-Request an den Authorization Server den geänderten Access Token enthalten müssen, um mit der aktuellen Version übereinzustimmen. In der Antwort vom Authorization Server auf den Introspection-Request müsste

dann das Schlüsselmaterial für den Client entfernt werden, welches im CTP den Client-Token darstellt, da dieser nicht mehr über den Resource Server an den Client gegeben wird. Das Schlüsselmaterial für den Resource Server würde hierbei weiterhin ausgetauscht werden [SSW+19, Seite 74]. Ausgehend von den beschriebenen Sachverhalten, sollte die Entfernung des Client-Tokens ausreichen, um die von H. Tschofenig beschriebene Scyther-Spezifikation in der Codedarstellung 4.3 auf die aktuelle Version des ACE-OAuth-Frameworks zu aktualisieren. Das letzte Send-Event würde, ausgehend von der aktuellen ACE-OAuth-Spezifikation, in der Scyther-Spezifikation nicht auftreten, da das Schlüsselmaterial nicht zum Client weitergegeben wird.

Da die definierten Claims weiterhin gültig sind, muss hier keine Veränderung vorgenommen werden.

4.1.3 Rolle: Authorization Server

```
0  role AS   %A: Authorization Server
   {
2    fresh key: SessionKey;

4    recv_2(RS, AS, {C}k(RS, AS);
     send_3(AS, RS, {key, {key}k(C, AS)}k(RS, AS));
6  }
```

Codedarstellung 4.4: Implementierung der Rolle des Authorization Servers des CTPs in Scyther nach H. Tschofenig [Tsc18].

Der Authorization Server beteiligt sich an dem CTP, indem dieser zunächst die benötigten Schlüsselmaterialien erzeugt. Dies geschieht mittels des Schlüsselwortes fresh, wie in Unterunterabschnitt 3.2.2.6 beschrieben. Als erster Schritt (recv_2) erhält der Authorization Server den verschlüsselten Access Token durch den Resource Server, wie in der Codedarstellung 4.4 zu sehen. Infolgedessen sendet der Authorization Server die jeweiligen Schlüsselmaterialien verschlüsselt an den Resource Server als Antwort auf den Token Introspection Request.

Der Authorization Server erzeugt weiterhin das Schlüsselmaterial, wie in der Scyther-Spezifikation in der Codedarstellung 4.4 dargestellt. Hinsichtlich der Anfrage des Access Tokens beim Authorization Server, muss der Access Token, wie zuvor beschrieben, abgeändert werden. Zunächst würde also ein Receive-Event korrespondierend zu dem beschriebenen Send-Event des Clients auftreten, gefolgt von der Versendung des beschriebenen Access Tokens. Das Empfangen des Access Tokens durch den Resource Server, sowie die Antwort an den Resource Server erfolgt korrespondierend zu den beschriebenen Events des Resource Servers.

4.1.4 Bewertung der Weiterverwendung

Die Scyther-Spezifikation des CTPs, welche von H. Tschofenig [Tsc18] entwickelt wurde, kann für die aktuelle Version des ACE-OAuth-Frameworks nicht weiterverwendet werden, da diese keine Übereinstimmung zur aktuellen Version liefert. Die Analyse der bestehenden Scyther-Spezifikation hat gezeigt, dass das

CTP lediglich geringe Ähnlichkeiten zu der aktuellen Version aufweist. Die dargelegten Änderungen können jedoch die Grundlage für die Verwendung in der Scyther-Spezifikation des C3DC-Profils bilden.

4.2 Entwurf einer Scyther-Spezifikation für das C3DC-Profil

Damit die Analyse des C3DC-Profils vorgenommen werden kann, müssen zunächst einige Annahmen getroffen werden, die sich hinsichtlich des ACE-OAuth-Frameworks und des C3DC-Profils ergeben. Darüber hinaus sollen Kriterien beziehungsweise Erwartungen geschaffen werden, die einen Vergleich mit den Ergebnissen der Analyse ermöglichen.

Zunächst werden die Erwartungen, die an das C3DC-Profil gestellt werden, definiert.

1. **Geheimhaltung des ausgetauschten Schlüsselmaterials:**
   Eine wichtige Eigenschaft beim Austausch von Schlüsselmaterialien ist die Geheimhaltung ebendieser. Einem Angreifer darf es nicht ermöglicht werden, sich Zugang zu diesen Schlüsselmaterialien zu beschaffen. Das ACE-OAuth-Framework beschreibt Mechanismen zur Authentifizierung und Autorisierung von Geräten. Diese Vorgänge werden allgemein als vetraulich angesehen. Dementsprechend wird die Erwartungshaltung getroffen, dass der Austausch der Schlüsselmaterialien so ausgeführt wird, dass die Schlüsselmaterialien vor einem Angreifer geheimgehalten wird und daher der Vertraulichkeit nachgekommen wird.
2. **Ausführung des Protokolls:**
   Es wird darüber hinaus erwartet, dass das Protokoll wie spezifiziert, ausgeführt wird. Hierunter wird verstanden, dass die jeweiligen Nachrichten wie im C3DC-Profil spezifiziert, ausgeführt werden und dabei der Inhalt der Nachrichten mit der Spezifikation übereinstimmt. Darüber hinaus wird erwartet, dass das Protokoll auch ohne die optionalen Schritte den Sicherheitseigenschaften gerecht wird.

Des Weiteren wurden verschiedene Annahmen für die Analyse des C3DC-Profils getroffen, die im Folgenden erläutert werden:

1. **Optionale Schritte im C3DC-Profil:**
   Die optionalen Schritte des Unauthorized Resource Requests und der AS Information werden in der Analyse nicht betrachtet, da angenommen wird, dass die Schritte zum Entdecken der jeweiligen Teilnehmer bereits stattgefunden haben und der Client bereits weiß, welcher Resource Server angesprochen wird.
2. **Authorized Resource Request:**
   Hinsichtlich des Authorized Resource Requests wird die Annahme getroffen, dass dieser in der Scyther-Spezifikation nicht stattfindet. Dies wird

damit begründet, dass der Request bereits davon ausgeht, das Schlüsselmaterial beziehungsweise den Access Token bereits ausgetauscht und eine DTLS-Sitzung aufgebaut zu haben [GBB18b, Seite 6 f.]. Dementsprechend wird im letzten Schritt, anstelle des Authorized Resource Requests, der Austausch des Access Tokens vorgenommen. Dies geht einher mit der DTLS-ClientKeyExchange Message [GBB+18c, Seite 4].

3. **Aufbau der TLS-/DTLS-Sitzungen:**
   Es wird angenommen, dass die DTLS-Sitzungen zwischen dem Client und dem Client Authorization Server, sowie zwischen dem Resource Server und dem Authorization Server bereits aufgebaut sind. Bei der TLS-/DTLS-Sitzung zwischen dem Client Authorization Server und dem Authorization Server wird ebenfalls davon ausgegangen, dass die Sitzung bereits etabliert wurde, da dies kein eigentlicher Bestandteil des C3DC-Profils ist.

4. **Access Request des Clients an den Client Authorization Server:**
   Da weder das C3DC-Profil noch das ACE-OAuth-Framework spezifizieren, wie der Access Request definiert ist, wird hier angenommen, dass dieser sich ähnlich zum Token Request des Clients verhält. Dies bedeutet, dass der Client die benötigten Informationen an den Client Authorization Server weitergibt. Insbesondere wird angenommen, dass der Access Request den Resource Server spezifiziert, welchen der Client mit seinem Request adressieren möchte, wie es in dem Token Request optional möglich ist.

5. **Schlüsselverteilung:**
   Hinsichtlich des auszutauschenden Schlüsselmaterials wird angenommen, dass der Access Token das Schlüsselmaterial selbst und nicht nur eine Referenz enthält. Dementsprechend wird keine Introspection genutzt, damit der Resource Server dieses Schlüsselmaterial erhält.

   Darüber hinaus wird angenommen, dass der Client mit dem Client Authorization Server, sowie der Client Authorization Server mit dem Authorization Server und der Resource Server mit dem Authorization Server jeweils einen Langzeit-Schlüssel für die Kommunikation miteinander besitzen. Diese Annahme wird ausgehend vom Aufbau der TLS- beziehungsweise DTLS-Sitzungen, sowie ausgehend von der ACE-OAuth-Spezifikation getroffen, worin beschrieben wird, dass bestimmte Techniken zum Austausch dieser Schlüsselmaterialien angewendet werden [SSW+19, Seite 12]. Ebenfalls wird dies durch L. Arnaboldi et al. in ihrer Analyse des ACE-OAuth-Frameworks angenommen [AT19, Seite 6].

6. **Access Token und Access Information:**
   Um mit dem Werkzeug Scyther eine Testspezifikation modellieren zu können, müssen bestimmte Informationen wegfallen, da diese durch Scyther nicht modelliert beziehungsweise berücksichtigt werden können. Hierbei entfallen unter anderem die Informationen für die Zugangsberechtigungen des Clients im Access Token.

7. **Nutzung eines MAC-Algorithmus:**
   Aus der Spezifikation des ACE-OAuth-Frameworks geht hervor, dass die

Nutzung eines AEAD[54]-Algorithmus einem MAC[55]-Algorithmus zum Integritätsschutz vorzuziehen ist. Da jedoch keine Implementierung bekannt ist, die dieses Verhalten in Scyther modelliert, wird sich hier für die Verwendung eines MAC-Algorithmus entschieden. Hierbei wird dieser nach H. Yang et al. [YOP16] modelliert.

8. **Definition des Begriffs securely:**
   Aus der Spezifikation des C3DC-Profils geht nicht hervor, welche Bedeutung der Begriff securely in Bezug auf die Spezifikation hat. Da dies ebenfalls nicht aus dem ACE-OAuth-Framework hervorgeht, wird hier die folgende Annahme getroffen. Es wird angenommen, dass unter securely verstanden wird, dass die Kommunikation sowohl verschlüsselt als auch einem Integritätsschutz unterzogen wird. Dies geht einher mit der Information, dass die Kommunikation zwischen dem Authorization Server und dem Client hinsichtlich der Vertraulichkeit und der Integrität geschützt ist [SSW+19, Seite 42]. Hierbei wird angenommen, dass ein MAC, wie für den Access Token, genutzt wird, da dies nicht aus der Spezifikation des C3DC-Profils hervorgeht.

4.3 Implementierung des C3DC-Profils in Scyther

Ausgehend von den in Abschnitt 4.2 beschriebenen Annahmen wurde die folgende Implementierung des C3DC-Profils vorgenommen. Die vollständige Implementierung des C3DC-Profils in Scyther wird im Anhang im Abschnitt A.2 dargestellt. Hierbei wird sich auf die Beschreibung der Send-Events beschränkt, da sich die Receive-Events korrespondierend verhalten und lediglich den Empfang der jeweiligen Nachricht darstellen. Die Scyther-Spezifikation wird in den einzelnen Rollen beschrieben.

```
usertype SessionKey;

hashfunction MAC;

macro AT = {RS, {key}k(RS, AS)}k(RS, AS), MAC({RS, {key}k(RS, AS
    )}k(RS, AS), k(RS, AS));
macro AI = key;

protocol c3dc (C, CAS, RS, AS)
{
}
```

Codedarstellung 4.5: Implementierung des C3DC-Profils in Scyther.

Die Scyther-Spezifikation in der Codedarstellung 4.5 definiert ein Protokoll mit vier Rollen. Dies sind der Client (C), der Client Authorization Server (CAS), der Resource Server (RS) und der Authorization Server (AS), wie in dem C3DC-Profil spezifiziert [GBB18b, Seite 2 ff.]. Zuvor werden ein benutzerdefinierter Typ,

[54] Die Authenticated Encryption with Associated Data (AEAD) wird hier nicht näher betrachtet.
[55] Der Message Authentication Code (MAC) wird hier nicht näher betrachtet.

eine Hashfunktion, sowie zwei Makros definiert. Der benutzerdefinierte Typ SessionKey wird entsprechend der Analyse des CTPs [Tsc18] definiert und ebenso verwendet. Dies ist notwendig, um im weiteren Verlauf den Schlüssel modellieren zu können. Die Hashfunktion wird definiert, um einen MAC-Algorithmus zu modellieren.

Mittels der beiden Makros werden der Access Token (AT) und die Access Information (AI) dargestellt. Der Access Token wurde definiert als Teil einer Nachricht, bestehend aus dem Schlüssel (key) und dem angedachten Empfänger des Access Tokens, dem Resource Server. Bezugnehmend auf die Spezifikation des C3DC-Profils wird das Schlüsselmaterial (key) an den Resource Server weitergereicht [GBB18b, Seite 3]. Dies wird ebenfalls durch das ACE-OAuth-Framework bestätigt, worin beschrieben wird, dass der Resource Server das durch den Access Token erhaltene Schlüsselmaterial nutzt, um eine sichere Verbindung mit dem Client einzugehen [SSW+19, Seite 14]. In dem ACE-OAuth-Framework wird darüber hinaus ausgesagt, dass ein symmetrischer Schlüssel, welcher im Access Token direkt angegeben ist, verschlüsselt sein muss [SSW+19, Seite 41]. Dementsprechend wird der Schlüssel zunächst zwischen dem Resource Server und dem Authorization Server verschlüsselt, wie es in Zeile vier der Codedarstellung 4.5 zu sehen ist. Das Hinzufügen des angedachten Empfängers im Access Token wird durch die ACE-OAuth-Spezifikation definiert [SSW+19, Seite 41]. Hierbei wird vor dem Schlüssel der Resource Server angehängt. Der Access Token wird wiederum zwischen dem Resource Server und dem Authorization Server verschlüsselt, da aus der ACE-OAuth-Spezifikation hervorgeht, dass ein unverschlüsselter Access Token einem Angreifer Informationen offenbaren könnte. Zusätzlich wird ein MAC-Algorithmus benutzt, um den Inhalt des Access Tokens einem Integritätsschutz zu unterziehen. Hierbei wird die Annahme hinsichtlich der Nutzung eines MAC-Algorithmus angewendet, wie in Abschnitt 4.2 beschrieben. Dem MAC-Algorithmus wird der symmetrische Schlüssel des Resource Servers und des Authorization Servers hinzugefügt, wie von C. Paar und J. Pelzl beschrieben [PP10, Seite 320 ff.]. In dem ACE-OAuth-Framework wird hierzu beschrieben, dass die Nutzung einer digitalen Signatur beziehungsweise eines MAC-Algorithmus verwendet werden sollte, um den Inhalt des Access Tokens vor bestimmten Gefahren zu schützen [SSW+19, Seite 41].

Die Access Information enthalten lediglich den Schlüssel (key). Dies geht aus der Spezifikation des C3DC-Profils hervor, worin beschrieben wird, dass der Client mit Informationen ausgestattet wird, welche das Schlüsselmaterial enthalten [GBB18b, Seite 3]. Ebenfalls lässt sich dies aus der ACE-OAuth-Spezifikation schließen, in welcher ausgesagt wird, dass der Client das Schlüsselmaterial aus den Access Information erhält [SSW+19, Seite 14]. Dementsprechend wird die Zeile fünf der Codedarstellung 4.5 spezifiziert.

4.3.1 Rolle: Client

```
role C
{
  var key: SessionKey;

  send_1(C, CAS, {RS}k(C, CAS));
  recv_4(CAS, C, {AI, AT}k(C, CAS), MAC({AI, AT}k(C, CAS), k(C,
   CAS)));
  send_5(C, RS, AT);

  claim_C1(C, Secret, key);
  claim_C2(C, Alive);
  claim_C3(C, Weakagree);
}
```

Codedarstellung 4.6: Implementierung der Rolle des Clients des C3DC-Profils in Scyther.

In der Codedarstellung 4.6 wird die Rollen-Spezifikation des Clients in Scyther vorgenommen. Hierbei wird zunächst die Deklarierung der Variable key vorgenommen, die im späteren Verlauf das Schlüsselmaterial darstellen wird und geschieht, wie in der Analyse des CTPs [Tsc18] beschrieben. Dies wird vorgenommen, da es dem Werkzeug Scyther damit möglich wird, die Analyse des Schlüsselaustauschs durchführen zu können.

Das erste Send-Event (send_1) wird genutzt, um den Access Request des Clients zu modellieren [GBB18b, Seite 6]. Diese nicht optionale Kommunikation findet zwischen dem Client und dem Client Authorization Server statt. Wie in Abschnitt 4.2 beschrieben, wird weder in der ACE-OAuth-Spezifikation noch in dem C3DC-Profil spezifiziert, wie der Access Request ausgeführt wird. Entsprechend der in Abschnitt 4.2 beschriebenen Annahme wird hier durch den Client spezifiziert, auf welchen Resource Server zugegriffen werden soll. Dies wurde aus der ACE-OAuth-Spezifikation abgeleitet. Hierbei wird beschrieben, dass der Client im Token Request optional angeben kann, welchen Resource Server dieser Request adressieren soll [SSW+19, Seite 65]. Darüber hinaus erfolgt die Kommunikation zwischen dem Client und dem Client Authorization Server verschlüsselt über den aufgebauten DTLS-Kanal [GBB18b, Seite 6 f.]. Hier wird kein MAC-Algorithmus genutzt, um die Nachricht einem Integritätsschutz zu unterziehen, da der Client eingeschränkt ist und aus dem C3DC-Profil, sowie aus der ACE-OAuth-Spezifikation nicht hervorgeht, ob dieser Client in der Lage ist eine solche Aufgabe auszuführen.

Damit der Authorized Resource Request geschützt stattfinden kann, wie in dem C3DC-Profil [GBB18b, Seite 7] beschrieben, muss zunächst die DTLS-Sitzung zwischen dem Client und dem Resource Server aufgebaut werden. Hierfür muss der Client den Access Token an den Resource Server senden. Dies geschieht, indem der Access Token als Inhalt der ClientKeyExchange Message des DTLS-Handshakes genutzt wird, wie im CoAP-DTLS-Profil beschrieben [GBB+18c, Seite 4, 11]. Hierbei wird die in Abschnitt 4.2 beschriebene Annahme angewendet. Zu sehen, ist

dieser Sachverhalt im send_5-Event in der Codedarstellung 4.6. Hierbei handelt es sich um einen Request, welcher keinen zusätzlichen Vertraulichkeitsschutz erfährt, da zum einen der Access Token zwischen dem Resource Server und dem Authorization Server verschlüsselt ist und zum anderen kein Schlüsselmaterial zwischen dem Client und dem Resource Server etabliert ist.

Um die Spezifikation auf bestimmte Sicherheitseigenschaften zu testen, werden die folgenden Forderungen formuliert.

Zunächst wird die Forderung des Clients spezifiziert, dass das ausgetauschte Schlüsselmaterial geheimgehalten und nicht kompromittiert wurde. Damit ist gemeint, dass der Angreifer keine Möglichkeit hat das Schlüsselmaterial in irgendeiner Weise abzufangen und zu nutzen. Diese Eigenschaft geht aus dem Sinn des ACE-OAuth-Frameworks und des C3DC-Profils hervor. Dementsprechend wird die Zeile acht in der Codedarstellung 4.6 definiert. Eine weitere Forderung, die vom Client gestellt wird, ist, dass die Rollen mit denen der Client kommuniziert das Aliveness-Claim, wie in Unterunterabschnitt 3.2.1.2 beschrieben, erfüllen. Demnach müssen die Rollen je einen Durchlauf des Protokolls mit dem Client durchgeführt haben [Low97, Seite 2]. Die Weak Agreement-Forderung, wie in Zeile zehn der Codedarstellung 4.6 spezifiziert, dient der Erweiterung des Aliveness-Claims, wie in Unterunterabschnitt 3.2.1.2 beschrieben. Diese beiden Forderungen werden gestellt, um sicherzustellen, dass der Client auch tatsächlich mit den Rollen kommuniziert hat.

4.3.2 Rolle: Client Authorization Server

```
0  role CAS
   {
2    var key: SessionKey;

4    recv_1(C, CAS, {RS}k(C, CAS));
     send_2(CAS, AS, {RS, C}k(CAS, AS), MAC({RS, C}k(CAS, AS), k(
      CAS, AS)));
6    recv_3(AS, CAS, {AI, AT}k(CAS, AS), MAC({AI, AT}k(CAS, AS), k(
      CAS, AS)));
     send_4(CAS, C, {AI, AT}k(C, CAS), MAC({AI, AT}k(C, CAS), k(C,
      CAS)));
8
     claim_CAS1(CAS, Secret, key);
10   claim_CAS2(CAS, Alive);
     claim_CAS3(CAS, Weakagree);
12   claim_CAS4(CAS, Niagree);
     claim_CAS5(CAS, Nisynch);
14 }
```

Codedarstellung 4.7: Implementierung der Rolle des Client Authorization Servers des C3DC-Profils in Scyther.

Wie bei dem Client muss in der Implementierung des Client Authorization Servers die Deklarierung der Variable key angegeben werden, wie es in der Codedarstellung 4.7 gezeigt wird. Dies ist notwendig, damit der Transport des Schlüsselmate-

rials an den Client und damit an den Resource Server stattfinden kann.

Der Token Request wird in der Scyther-Spezifikation mittels des zweiten Send-Events (send_2) modelliert. Hierbei sendet der Client Authorization Server eine Nachricht an den Authorization Server. Diese Nachricht enthält zunächst einmal den Resource Server, den der Client als den Empfänger seines Requests spezifiziert hat. Es wird davon ausgegangen, dass der Client Authorization Server sich wie der Client verhält, wenn dieser einen Token Request beim Authorization Server ausführt. Dementsprechend wird angenommen, dass der Resource Server durch den Client Authorization Server in seiner Nachricht an den Authorization Server adressiert wird, wie es optional im Token Request durch den Client geschehen kann [SSW+19, Seite 65]. Hierbei wird die in Abschnitt 4.2 beschriebene Annahme angewendet. Darüber hinaus werden in dieser Nachricht Informationen über den Client an den Authorization Server versendet. Dies geht aus der Spezifikation des C3DC-Profils hervor, in der beschrieben wird, dass der Authorization Server Informationen von dem Resource Owner einholt, um festzustellen, ob der Client Authorization Server berechtigt ist, Informationen und Schlüsselmaterialien über den Client herauszugeben [GBB18b, Seite 6]. Es wird hierbei angenommen, dass der Client Authorization Server Informationen über den Client an den Authorization Server weitergibt, welche durch das C dargestellt werden. Zwischen dem Client Authorization Server und dem Authorization Server ist eine TLS- oder eine DTLS-Sitzung aufgebaut [GBB18b, Seite 6]. Darüber hinaus wird in dem ACE-OAuth-Framework beschrieben, dass es dem Profil obliegt zu definieren, wie die Kommunikation geschützt wird [SSW+19, Seite 22, 68, 73]. In dem C3DC-Profil wird dazu ausgesagt, dass die Kommunikation zwischen dem Client Authorization Server und dem Authorization Server geschützt erfolgt [GBB18b, Seite 6]. Dargestellt wird dies in der Codedarstellung 4.7 mittels der Angabe des zu verwendenden Schlüssels in Zeile fünf. Es wird darüber hinaus angenommen, dass ein MAC zum Integritätsschutz der Nachricht angewendet wird, wie in Abschnitt 4.2 definiert.

In dem vierten Send-Event (send_4) wird der Token Transfer durch den Client Authorization Server vorgenommen. Der Token Transfer wird wie im C3DC-Profil definiert an den Client gerichtet [GBB18b, Seite 6]. Dies lässt sich an Zeile sieben der Codedarstellung 4.7 erkennen. Der Inhalt des Transfers entspricht der Weiterleitung des zuvor erhaltenen Access Tokens, sowie der Access Information, wie in dem C3DC-Profil beschrieben [GBB18b, Seite 6]. Hierbei wird jedoch die Verschlüsselung zwischen dem Authorization Server und dem Client Authorization Server aufgehoben. Hinzukommend wird durch das C3DC-Profil gefordert, dass der Versand des Token Transfers geschützt geschieht [GBB18b, Seite 7]. Entsprechend wird der Inhalt der Nachricht durch das jeweilige Schlüsselmaterial erweitert, wie dies der Zeile sieben der Codedarstellung 4.7 entnommen werden kann. Darüber hinaus findet die Angabe eines MACs statt, um der Annahme über den Begriff securely, wie in Abschnitt 4.2 beschrieben, nachzukommen.

Die für den Client Authorization Server gestellten Forderungen entsprechen denen, die von H. Tschofenig in seiner Analyse des CTPs [Tsc18, Seite 5] an den

Client gestellt wurden. Dies wird so modelliert, da angenommen wird, dass der Client Authorization Server sich wie der Client verhält hinsichtlich des Requests beziehungsweise des Austauschs eines Access Tokens mit dem Authorization Server. Hierbei fordert der Client Authorization Server zunächst die Geheimhaltung des Schlüsselmaterials (key), wie es aus dem ACE-OAuth-Framework und dem C3DC-Profil hervorgeht [SSW+19, GBB18b]. Außerdem wird gefordert, dass die Kommunikationspartner des Client Authorization Servers den Aliveness-Claim erfüllen und dementsprechend einen Durchlauf des Protokolls durchführen. Die Aliveness-Forderung wird durch die Weak Agreement-Forderung erweitert, wie in Unterunterabschnitt 3.2.1.2 beschrieben. Damit soll erreicht werden, dass die jeweiligen Rollen sich der Kommunikation mit dem jeweiligen Partner sicher sein können. Um die Einhaltung der Reihenfolge der versendeten Nachrichten und die Übereinstimmung der Daten zu fordern, werden in Zeile 12 und 13 der Codedarstellung 4.7 die Niagree- und Nisynch-Claims spezifziert.

4.3.3 Rolle: Resource Server

```
0 role RS
  {
2   var key: SessionKey;

4   recv_5(C, RS, AT);

6   claim_RS1(RS, Secret, key);
    claim_RS2(RS, Alive);
8   claim_RS3(RS, Weakagree);
  }
```

Codedarstellung 4.8: Implementierung der Rolle des Resource Servers des C3DC-Profils in Scyther.

Zum Erhalt des Schlüsselmaterials ist es ebenfalls für den Resource Server notwendig, dass die Variable key deklariert wird, wie in der Codedarstellung 4.8 gezeigt.

Da außer dem Erhalt des Access Tokens keine weitere Aktion des Resource Servers in der Spezifikation des C3DC-Profils erfolgt, da die optionalen Schritte des Unauthorized Resource Requests und der AS Information nicht berücksichtigt werden, muss hier mit Ausnahme der Forderungen nichts weiter modelliert werden [GBB18b, Seite 6].

Die Spezifizierung der Forderungen für den Resource Server wird ebenfalls wie in der Analyse des CTPs von H. Tschofenig [Tsc18] vorgenommen. Dies lässt sich darauf zurückführen, dass der Resource Server hier lediglich das Schlüsselmaterial beziehungsweise den Access Token empfängt. Für den Resource Server wird die Forderung definiert, dass das erhaltene Schlüsselmaterial (key) einem Angreifer nicht offenbart wurde. Darüber hinaus wird die Aliveness-Forderung, sowie die Weak Agreement-Forderung gestellt, um sicherzustellen, dass der ausgetauschte Access Token auch tatsächlich durch den Client an den Resource Server versendet wurde.

4.3.4 Rolle: Authorization Server

```
0 role AS
  {
2   fresh key: SessionKey;

4   recv_2(CAS, AS, {RS, C}k(CAS, AS), MAC({RS, C}k(CAS, AS), k(
     CAS, AS)));
    send_3(AS, CAS, {AI, AT}k(CAS, AS), MAC({AI, AT}k(CAS, AS), k(
     CAS, AS)));
6 }
```

Codedarstellung 4.9: Implementierung der Rolle des Authorization Servers des C3DC-Profils in Scyther.

Um das Schlüsselmaterial zu erzeugen, wird in der Scyther-Spezifikation in der Codedarstellung 4.9 das Schlüsselwort fresh für die Variable key benutzt.

Der Token Grant des Authorization Servers an den Client Authorization Server wird im dritten Send-Event dargestellt. Zunächst erzeugt der Authorization Server den in Abschnitt 4.3 beschriebenen Access Token, sowie die Access Information [GBB18b, Seite 6]. Für das Senden des Token Grants spezifiziert das C3DC-Profil, dass der Authorization Server dafür Sorge tragen muss, dass der Client Authorization Server den Access Token und die Access Information geschützt erhält [GBB18b, Seite 6]. Ebenso geht aus der ACE-OAuth-Spezifikation hervor, dass jede Kommunikation mit dem Authorization Server verschlüsselt geschehen muss [SSW+19, Seite 15] Dementsprechend wird, wie es in der Codedarstellung 4.9 in Zeile 5 gezeigt wird, das Schlüsselmaterial zwischen dem Client Authorization Server und dem Authorization Server genutzt, um diese Anforderung zu modellieren. Wie in Abschnitt 4.2 beschrieben, wird der Begriff securely im C3DC-Profil nicht definiert. Entsprechend der Annahme wird zusätzlich zur Verschlüsselung die Verwendung eines MAC-Algorithmus vorgenommen.

C. Cremers und S. Mauw beschreiben in ihrer Veröffentlichung, dass die jeweiligen Rollen in Scyther nur eine lokale Sicht auf den Zustand des Systems haben, ausgehend von den erhaltenen Nachrichten [CM12, Seite 37]. Dementsprechend kann der Authorization Server nicht feststellen, ob das ausgetauschte Schlüsselmaterial kompromittiert wurde oder nicht. Dies gilt ebenfalls für die anderen Claims. Daraus resultierend werden keine Forderungen gestellt.

4.4 Auswertung der Ergebnisse des C3DC-Profils

Die Auswertung der Analyse der zuvor implementierten Spezifikation des C3DC-Profils soll im Folgenden betrachtet werden. Hierfür stellt die Abbildung 4.1 die Ergebnisse der Analyse mittels des Scyther-Werkzeugs zusammengefasst dar. Aus dieser Zusammenfassung wird ersichtlich, dass Scyther ausgehend von der Implementierung der Spezifikation mindestens elf Angriffe auf das Protokoll beziehungsweise auf die spezifizierten Forderungen identifizieren konnte. Obwohl das Scyther-Werkzeug tatsächliche Angriffe auf das Protokoll darstellen kann, werden hier vor allem jene Schritte aufgezeigt, die zur Verletzung des jeweiligen Claims führen.

Claim				Status		Comments
c3dc	C	c3dc,C1	Secret key	Fail	Falsified	At least 1 attack.
		c3dc,C2	Alive	Fail	Falsified	At least 1 attack.
		c3dc,C3	Weakagree	Fail	Falsified	At least 1 attack.
	CAS	c3dc,CAS1	Secret key	Fail	Falsified	At least 1 attack.
		c3dc,CAS2	Alive	Fail	Falsified	At least 1 attack.
		c3dc,CAS3	Weakagree	Fail	Falsified	At least 1 attack.
		c3dc,CAS4	Niagree	Fail	Falsified	At least 1 attack.
		c3dc,CAS5	Nisynch	Fail	Falsified	At least 1 attack.
	RS	c3dc,RS1	Secret key	Fail	Falsified	At least 1 attack.
		c3dc,RS2	Alive	Fail	Falsified	At least 1 attack.
		c3dc,RS3	Weakagree	Fail	Falsified	At least 1 attack.

Abbildung 4.1: Darstellung der Ergebnisse der Analyse des C3DC-Profils mittels Scyther.

4.4.1 Claims für den Client

Die Claims des Clients konnte Scyther allesamt falsifizieren. Dies geht aus dem Status der Abbildung 4.1 hervor.

In Abbildung 4.2 wird ein Teil des Angriffsgraphs für das Secret-Claim des Clients gezeigt. Der vollständige Angriffsgraph für das Secret-Claim des Clients wird in Abbildung A.1 dargestellt. Aus dem Graphen gehen vier Instanzen des Protokolls hervor. Die erste (Run #1) Instanz wird vom Client und die zweite (Run #2) vom Authorization Server ausgeführt. Der dritte (Run #3) und vierte (Run #4) Protokolldurchlauf wird vom Client Authorization Server ausgeübt. Auffällig hierbei ist die vierte Instanz, die in Abbildung 4.2 dargestellt wird. Hierbei wird dargestellt, dass Eve als Client Authorization Server betrachtet wird und mittels des initialen Wissen das Schlüsselmaterial zwischen dem Client und dem Client Authorization Server etablieren und dadurch die erste Nachricht konstruieren kann. Der Client Authorization Server nimmt hierbei an mit Eve als Client zu

kommunizieren. Entsprechend wird der zwischen dem Client Authorization Server und dem Authorization Server ausgehandelte Access Token, sowie die Access Information an Eve versendet. Daraus resultierend wird das Claim falsifiziert.

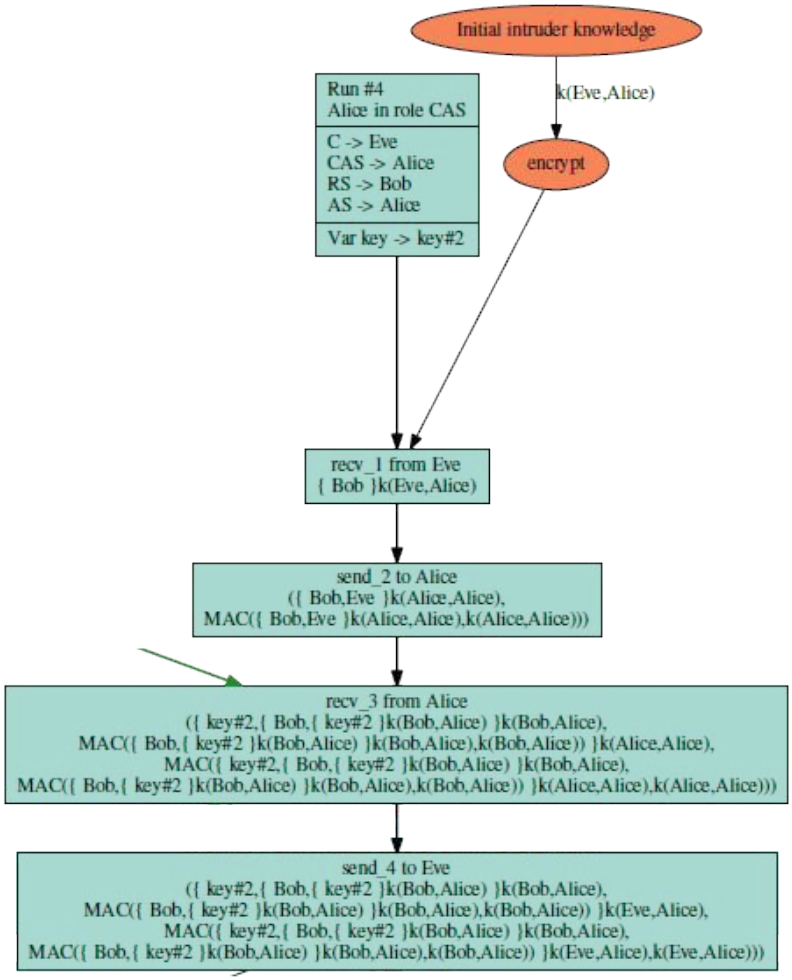

Abbildung 4.2: Darstellung eines Ausschnitts des Angriffsgraphen des Secret-Claims des Clients, welcher mittels Scyther generiert wurde. Dieser Teil des Angriffsgraphs wurde strukturell verändert, um eine Verbesserung der Lesbarkeit zu erzielen.

Der von Scyther ausgegebene Angriffsgraph zeigt, dass der Client Authorization Server einem nicht vertrauenswürdigen Client die Informationen weitergibt. Dieses Verhalten wird als unwahrscheinlich betrachtet, da sich der Client Authorization Server in derselben Sicherheitsdomäne wie der Client befindet [GBB18b, Seite 3 ff.]. Als Autorisierungs-Manager des Clients sollte der Client Authorization Server in der Lage sein die eigenen Clients von nicht vertrauenswürdigen Clients zu unterscheiden. Hier fehlt es an Einschränkungen für den Client Authorization Server, wann dieser den Access Token und die Access Information an einen Client weitergibt. Diese fehlen aufgrund der notwendigen Abstraktion des Protokolls. Ausgehend von dem Handbuch des Scyther-Werkzeugs [Cre14a] ist keine Möglichkeit bekannt, solche Einschränkungen zu modellieren. Dennoch kann eine Aussage über die Sicherheit des Schlüsselmaterials getroffen werden. Dies insofern, als dass es keine Indizien dafür gibt, dass eine Offenbarung des

Schlüsselmaterials an einen Angreifer andersweitig erfolgt. Da die Überprüfung der Autorisierung im C3DC-Profil stattfindet, kann angenommen werden, dass der Client Authorization Server eine Kommunikation mit einem nicht vertrauenswürdigen Akteur unterlässt. Somit deutet dies daraufhin, dass das Protokoll wie spezifiziert erfolgt und der Schlüsselaustausch somit geschützt ist.

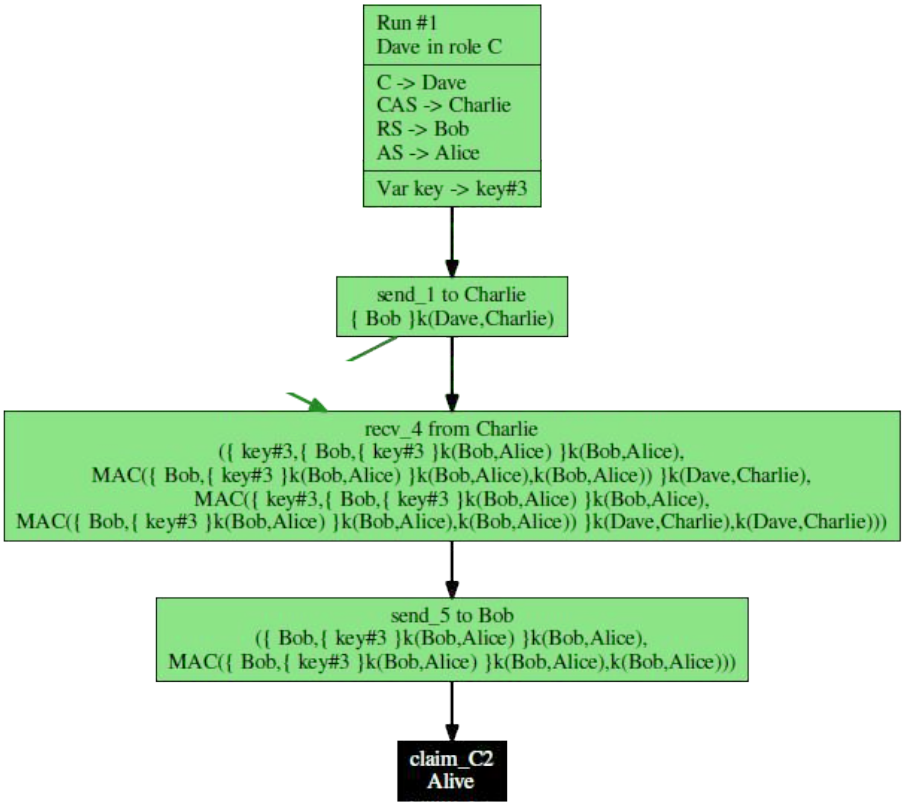

Abbildung 4.3: Darstellung eines Ausschnitts des Angriffsgraphen des Aliveness-Claims des Clients, welcher mittels Scyther generiert wurde. Dieser Teil des Angriffsgraphs wurde strukturell verändert, um eine Verbesserung der Lesbarkeit zu erzielen.

Die Abbildung 4.3 stellt einen Ausschnitt des Angriffsgraphen für den Aliveness-Claim des Clients dar, wohingegen in Abbildung A.2 der gesamte Angriffsgraph dargestellt wird. Diese Angriffsgraphen stellen ebenfalls die Verletzung des Weak Agreement-Claims dar[56]. In dem Angriffsgraphen in Abbildung A.2 werden drei Instanzen beziehungsweise Durchläufe des Protokolls gezeigt. Der erste Durchlauf (Run #1) wird durch den Client ausgeübt, hierbei wird der Client mit Dave bezeichnet. Die zweite Instanz (Run #2) wird durch den Client Authorization Server und die dritte Instanz durch den Authorization Server ausgeführt. Hierbei wird von den jeweiligen Rollen angenommen mit den zuvor beschriebenen Akteuren zu kommunizieren. Mittels der grünen Pfeile lässt sich darüber hinaus feststellen, dass die Inhalte der gesendeten Nachrichten den Inhalten der erhaltenen Nachrichten entsprechen. Auffällig ist jedoch, dass keine Instanz des Resource

[56]Dementsprechend wird dieser Graph nicht zusätzlich im Anhang hinzugefügt, da dieser keinen Mehrwert für diese Arbeit schafft.

Servers ausgeführt wird. Dementsprechend kommt es zur Falsifizierung des Aliveness-Claims. Erkennbar ist dies am fünften Send-Event (send_5 to Bob) der ersten Instanz (Run #1), wie dies in Abbildung 4.3 gezeigt wird. Der Resource Server Bob erhält die Nachricht jedoch nicht zwangsläufig, da der Resource Server keine Instanz des Protokolls ausführt. Entsprechend erfolgt auch die Falsifizierung des Weak Agreement-Claims, da keine Einigung über die Teilnahme am Protokoll stattfindet.

Die Falsifizierung der beiden Claims lässt sich auf die Passivität des Resource Servers zurückführen, welcher dieser in der Abstraktion des Protokolls einnimmt. Hierbei ist gemeint, dass der Resource Server lediglich eine Nachricht empfängt, aber selber keine sendet. Dies wirft die Frage auf, ob dies eine Schwäche im Design des Protokolls oder ob dies ein Fehler in der Abstraktion des Protokolls ist. Unter Betrachtung der optionalen Schritte wird ersichtlich, dass der Resource Server in der Antwort auf den Unauthorized Resource Request eine Nonce versenden kann, die daraufhin im Token Request, sowie im Access Token vorhanden sein muss [SSW+19, Seite 17 ff.]. Hierdurch würde der Resource Server aktiv an dem Protokollverlauf teilnehmen. Da diese beiden Schritte jedoch optional sind für das C3DC-Profil [GBB18b, Seite 6], werden diese nicht berücksichtigt. Ein weiterer Punkt, der hier berücksichtigt werden muss, ist der DTLS-Handshake. In der Abstraktion wird lediglich davon ausgegangen, dass die DTLS-ClientKeyExchange Message versendet wird, da hierbei der Access Token ausgetauscht wird. Allerdings würden hier auch weitere Nachrichten vom Resource Server folgen, würde dieser Handshake vollständig ausgeführt werden. Da dies jedoch nicht eindeutig aus dem C3DC-Profil hervorgeht, ob dies beziehungsweise wie dies geschieht, wird davon ausgegangen, dass die Testspezifikation diesbezüglich korrekt ist. Entsprechend wird die Falsifizierung auf das Design des Protokolls zurückgeführt.

4.4.2 Claims für den Client Authorization Server

Für den Client Authorization Server werden die gestellten Forderungen von Scyther, wie in Abbildung 4.1 dargestellt, falsifiziert.

In Abbildung 4.4 wird ein Ausschnitt des Angriffsgraphen für das Secret-Claim des Client Authorization Servers dargestellt. Die Abbildung A.3 zeigt den vollständigen Angriffsgraphen für das Secret-Claim des Client Authorization Servers. In dieser Abbildung werden vier Instanzen gezeigt, wobei sich auf den vierten Protokolldurchlauf (Run #4) in Abbildung 4.4 fokussiert wird. Dieser wird von Alice als Client Authorization Server ausgeführt. Alice kommuniziert mit einem nicht vertrauenswürdigen Akteur Eve als Client. Entsprechend führt dies dazu, dass der Client Authorization Server Eve den Access Token und die Access Information zukommen lässt, wie dies am vierten Send-Event (send_4 to Eve) an Eve zu sehen ist. Damit wird das Secret-Claim durch Scyther falsifiziert.

Hierbei zeigt sich, dass der Client Authorization Server einem nicht vertrauenswürdigen Client die Informationen weitergibt. Wie für das Secret-Claim des Clients in Unterabschnitt 4.4.1 beschrieben, wird dieses Verhalten als unwahrscheinlich betrachtet, da sich der Client Authorization Server in derselben Sicherheitsdomäne

wie der Client befindet [GBB18b, Seite 3 ff.]. Dementsprechend sollte der Client Authorization Server in der Lage sein die eigenen Clients von nicht vertrauenswürdigen Clients zu unterscheiden. Es kann somit aus dem Graphen gedeutet werden, dass das Protokoll seinen Sicherheitszielen bezüglich des Austauschs des Schlüsselmaterials nachkommt, wie in Unterabschnitt 4.4.1 beschrieben.

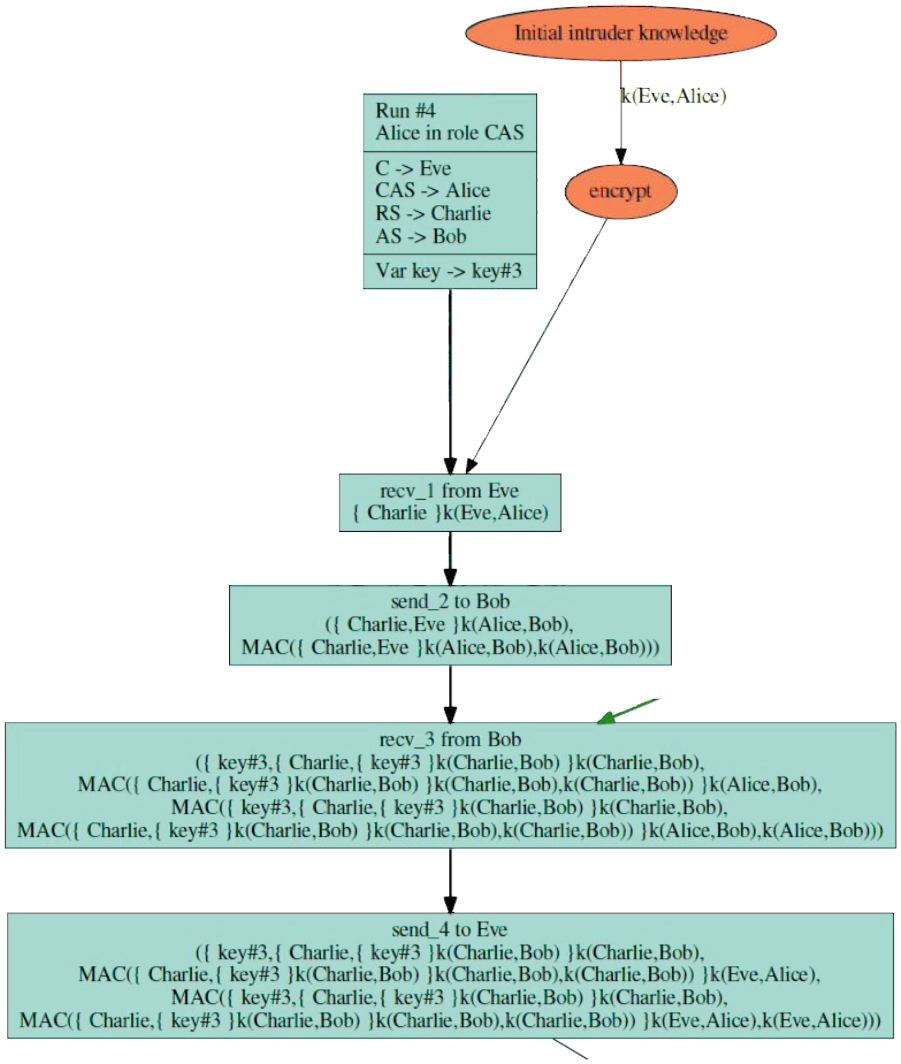

Abbildung 4.4: Darstellung eines Ausschnitts des Angriffsgraphen des Secret-Claims des Client Authorization Servers, welcher mittels Scyther generiert wurde. Dieser Teil des Angriffsgraphs wurde strukturell verändert, um eine Verbesserung der Lesbarkeit zu erzielen.

Die Abbildung 4.5 zeigt einen Teil des Angriffsgraphen für das Aliveness- und Weak Agreement-Claim des Client Authorization Servers. In Abbildung A.4 lässt sich der vollständige Angriffsgraph für die Claims Aliveness und Weak Agreement des Client Authorization Servers auffinden. Auch in diesem Fall sind die Angriffsgraphen[57] gleich, was darauf zurückzuführen ist, dass das Weak Agreement-Claim das Aliveness-Claim erweitert. Aus der Abbildung geht hervor, dass

[57] Der Angriffsgraph unterscheidet sich lediglich in der Bezeichnung des Claims, sowie in der Identifikationsnummer.

der Client eine Instanz des Protokolls ausführt, jedoch kein Schlüsselmaterial erhält. Dies lässt sich am Kasten mit der Beschriftung Run #2 erkennen, wie in Abbildung 4.5 erkennbar.

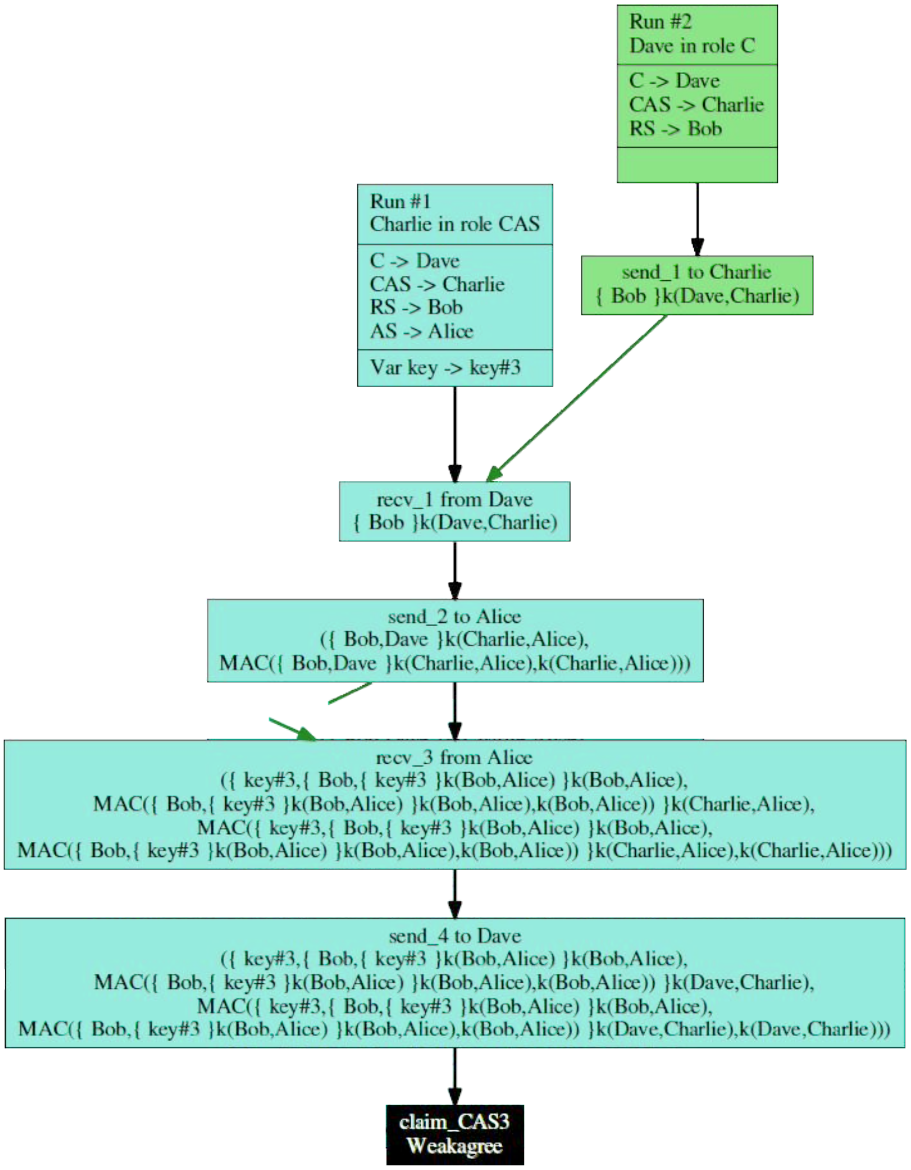

Abbildung 4.5: Darstellung eines Ausschnitts des Angriffsgraphen des Weak Agreement-Claims des Client Authorization Servers, welcher mittels Scyther generiert wurde. Dieser Teil des Angriffsgraphs wurde strukturell verändert, um eine Verbesserung der Lesbarkeit zu erzielen.

Hierbei ist das untereste Feld leer, woraus resultiert, dass der Client das Schlüsselmaterial nicht erhalten hat. Wird der Graph weiter betrachtet, so kann aus der ersten Instanz (Run #1) geschlossen werden, dass die Nachricht, welche den Access Token und die Access Information enthält, vom Client Authorization Server versendet wird. Allerdings muss der Client diese Nachricht nicht zwangsläufig erhalten, wie es in Abbildung A.4 gezeigt wird. Dementsprechend führt dies dazu, dass der Client keinen kompletten Durchlauf des Protokolls ausführt und das Aliveness-Claim wird entsprechend falsifiziert. Aufgrunddessen kommt es zur

Falsifizierung des Weak Agreement-Claims, da keine Einigung über die gegenseitige Teilnahme am Protokoll erfolgt.

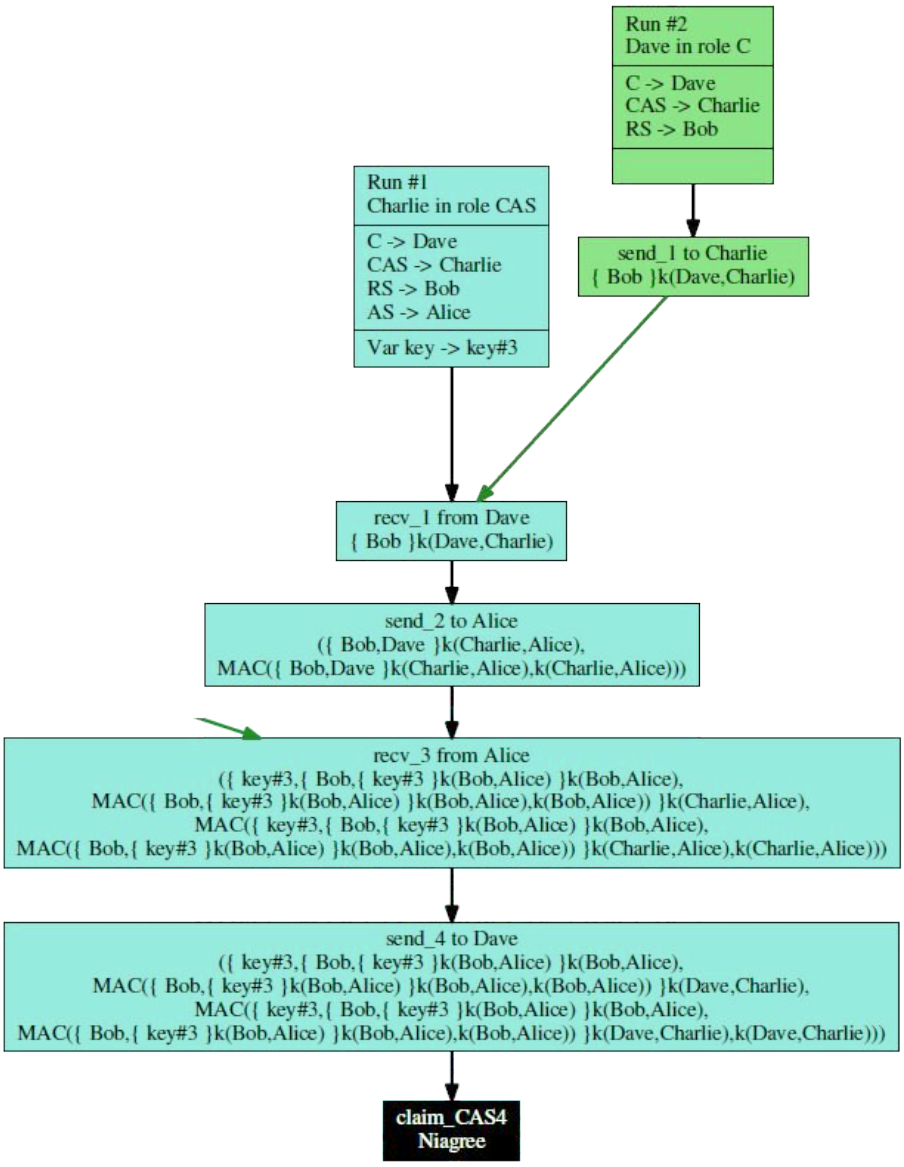

Abbildung 4.6: Darstellung eines Ausschnitts des Angriffsgraphen des Niagree-Claims des Client Authorization Servers, welcher mittels Scyther generiert wurde. Dieser Teil des Angriffsgraphs wurde strukturell verändert, um eine Verbesserung der Lesbarkeit zu erzielen.

Für das Claim Niagree des Client Authorization Servers wurde der in Abbildung A.5 dargestellte Angriffsgraph erzeugt. Dieser zeigt fünf Instanzen des Protokolls. Zwei dieser Instanzen werden vom Client ausgeführt, zwei vom Client Authorization Server, sowie eine weitere vom Authorization Server. Auffällig hierbei sind die beiden Instanzen des Clients. Aus der zweiten Instanz (Run #2) geht hervor, dass der Client sich am Protokoll insofern beteiligt, als dass dieser das erste Send-Event (send_1 to Charlie) versendet, wie erkennbar in dem Ausschnitt des Angriffsgraphen für das Niagree-Claim des Client Authorization Servers in Abbildung 4.6. Dies ergibt sich ebenfalls für den Client im fünften Protokolldurchlauf

(Run #5). Beiden Instanzen fehlt das Schlüsselmaterial. Unter Berücksichtigung des zuvor genannten Aspekts lässt sich in der ersten Instanz (Run #1) daraufhin feststellen, dass das vierte Send-Event (send_4 to Dave) an den Client erfolgt, allerdings der Client diese Nachricht nicht erhält. Entsprechend kommt es zur Falsifizierung des Niagree-Claims, da der Client und der Client Authorization Server keine Einigung über den Erhalt der Informationen erzielen.

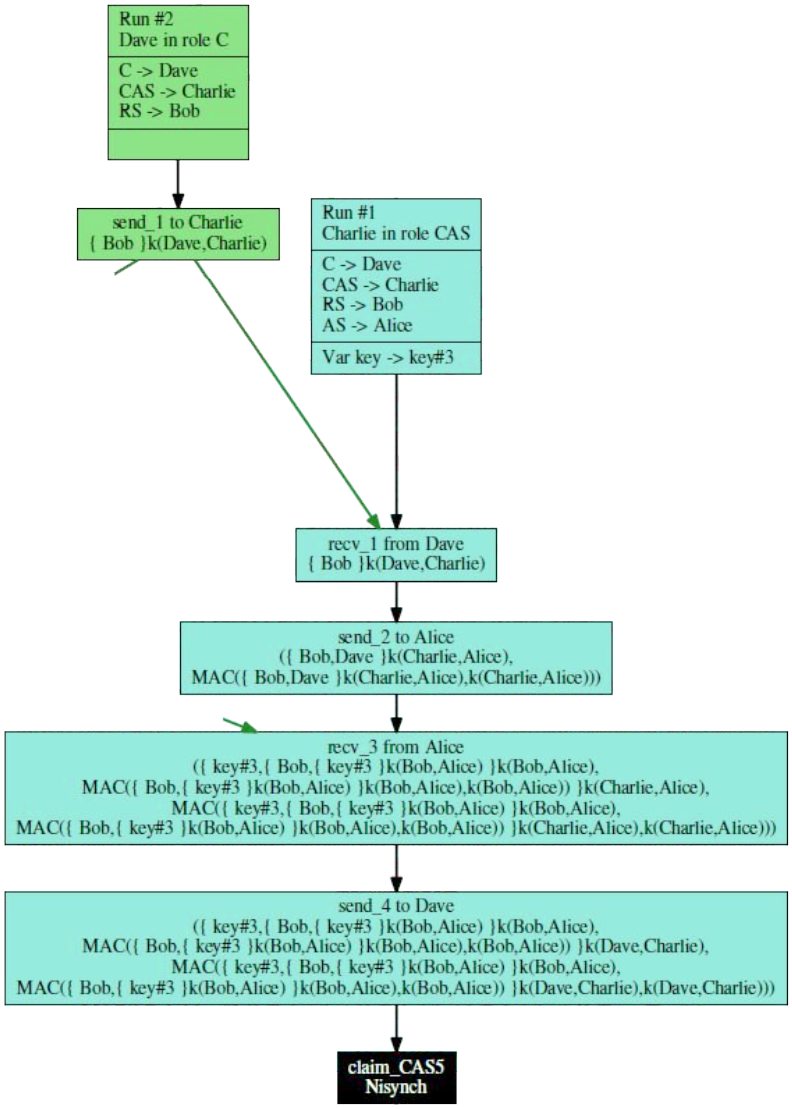

Abbildung 4.7: Darstellung eines Ausschnitts des Angriffsgraphen des Nisynch-Claims des Client Authorization Servers, welcher mittels Scyther generiert wurde. Dieser Teil des Angriffsgraphs wurde strukturell verändert, um eine Verbesserung der Lesbarkeit zu erzielen.

Die Abbildung A.6 zeigt den vollständigen Angriffsgraphen für das Nisynch-Claim. Hierbei werden vier Instanzen des Protokolls dargestellt. Die zweite Instanz (Run #2) wird vom Client ausgeführt. Hierbei wird aus der Abbildung 4.7 ersichtlich, dass der Client das erste Send-Event (send_1 to Charlie) versendet und daraufhin keine weitere Beteiligung am Protokoll erfolgt. Ebenfalls lässt sich erkennen, dass der Client kein Schlüsselmaterial erhält. Unter Betrachtung der

vierten Nachricht des Client Authorization Servers (send_4 to Dave) in der ersten Instanz (Run #1) wird deutlich, dass dieser die Nachricht an den Client sendet und der Client die Nachricht jedoch nicht erhält. Daraus resultierend kommt keine Einigung über die ausgetauschten Daten, sowie über die Reihenfolge der Nachrichten zustande und das Nisynch-Claim wird falsifiziert.

Ähnlich zum Resource Server nimmt der Client bei der Aushandlung des Access Tokens eine eher passive Rolle ein. Dementsprechend wird von Scyther die Falsifizierung der Claims vorgenommen. Hierbei bleibt offen, ob dies ein Fehler im Design des Protokolls ist oder ob die Abstraktion des Protokolls fehlerhaft ist. Mittels der optionalen Schritte würde eine Nonce über den Client an den Authorization Server gesendet werden, welcher als Inhalt für den Access Token genutzt werden muss [SSW+19, Seite 17 ff.]. Dies geschieht über den Client und eröffnet die Möglichkeit, dass dieser aktiv teilnimmt. Da das C3DC-Profil jedoch auch ohne diese optionalen Schritte auskommen soll, wird dies nicht weiter berücksichtigt. Ausgehend von der Spezifikation des C3DC-Profils werden zwischen dem Client und dem Client Authorization Server zwei Nachrichten ausgetauscht. Dies sind der Access Request und der Token Transfer. Hierbei wird der Access Request in der Spezifikation des C3DC-Profils nicht genau definiert, wie in Abschnitt 4.2 beschrieben. Dementsprechend kann hier nicht ausgeschlossen werden, ob die Modellierung des Protokolls gegebenenfalls falsch vorgenommen wurde. Da die Implementierung jedoch nach den vorhandenen Informationen der Spezifikation erfolgt ist, wird dies als korrekt angenommen.

4.4.3 Claims für den Resource Server

Aus dem Status der Abbildung 4.1 geht hervor, dass die Claims für den Resource Server falsifiziert wurden.

Hierfür wird in Abbildung 4.8 dargestellt, warum das Secret-Claim falsifiziert wird. Wird der Graph betrachtet, so lässt sich erkennen, dass es zwei Instanzen gibt, wobei die erste (Run #1) durch den Resource Server und die zweite (Run #2) durch den Authorization Server ausgeführt wird. Ebenso geht hervor, dass der Authorization Server mit einem nicht vertrauenswürdigen Client Authorization Server kommuniziert. Entsprechend wird die zweite Nachricht des Protokolls von Eve (recv_2 from Eve) erzeugt, woraufhin der Authorization Server den Access Token und die Access Information an Eve ausstellt. Somit wird das Secret-Claim falsifiziert, da Eve die Nachricht entschlüsseln kann und somit Zugriff auf das Schlüsselmaterial erhält.

Hierbei muss angemerkt werden, dass dieses Verhalten als unwahrscheinlich betrachtet wird. Dies lässt sich dahingehend begründen, dass diese beiden Rollen nach der Spezifikation des C3DC-Profils eine gegenseitige Authentifizierung durchführen [GBB18b, Seite 6]. Darüber hinaus geht aus dem Profil hervor, dass die beiden Rollen Informationen von ihren Overseeing Principals erhalten, mit welchem Authorization Server beziehungsweise Client Authorization Server sie kommunizieren dürfen [GBB18b, Seite 6]. Dies sollte die beiden Rollen in die Lage

versetzen, vertrauenswürdige Akteure von nicht vertrauenswürdigen Akteuren zu unterscheiden.

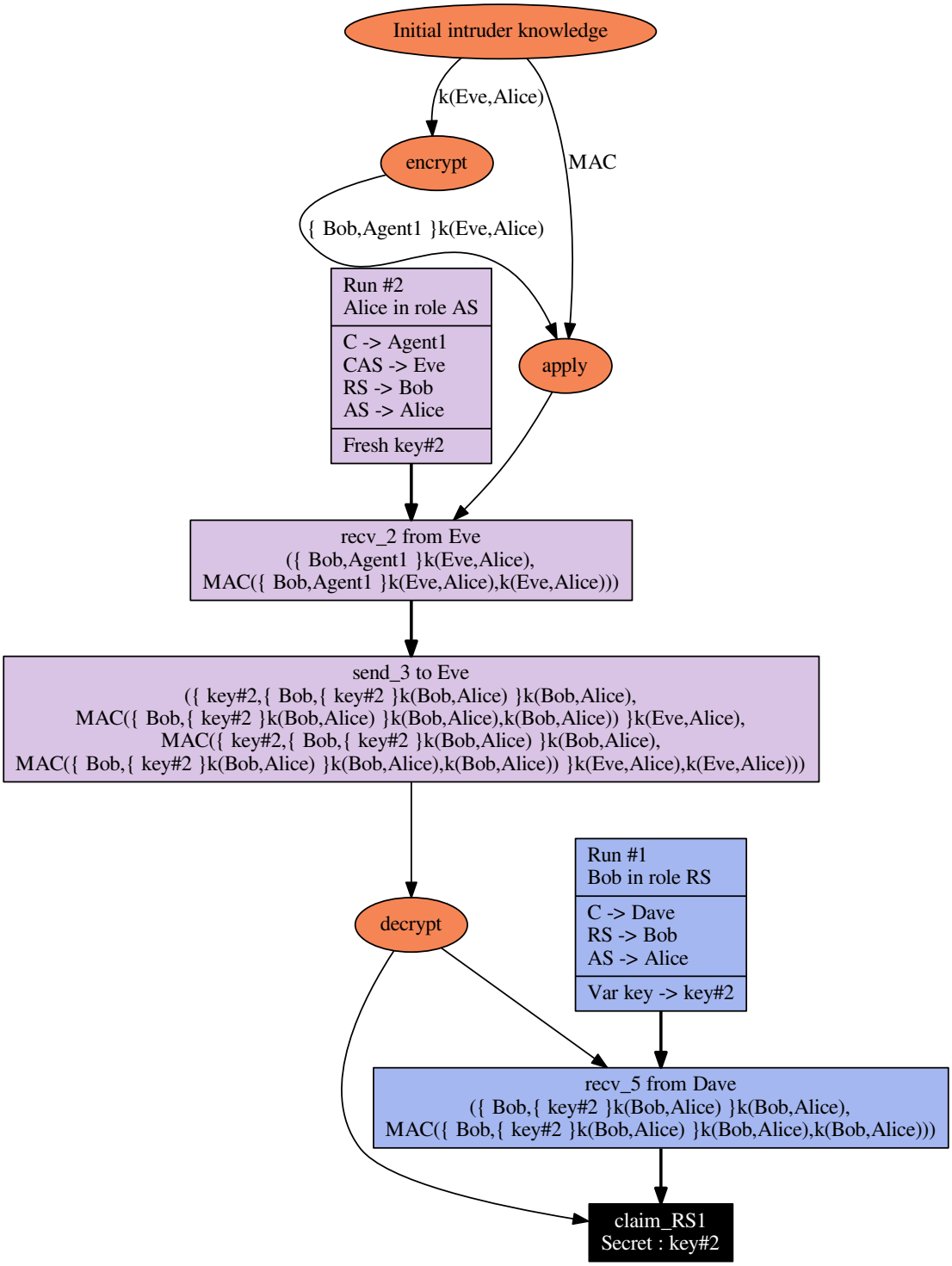

Abbildung 4.8: Darstellung des Angriffsgraphen des Secret-Claims des Resource Servers, welcher mittels Scyther generiert wurde.

Obwohl diese Einschränkungen beziehungsweise die Überprüfung der Autorisierung von Eve im Scyther-Werkzeug nicht modelliert werden kann, kann hier eine Aussage über das Schlüsselmaterial getroffen werden. Ausgehend davon, dass

der Client Authorization Server und der Authorization Server in der Lage sind, zu unterscheiden mit welchen Akteuren sie kommunizieren und eine Überprüfung der Autorisierung im C3DC-Profil stattfindet, deutet dies daraufhin, dass der Austausch des Schlüsselmaterials wie spezifiziert erfolgt und somit geschützt ist.

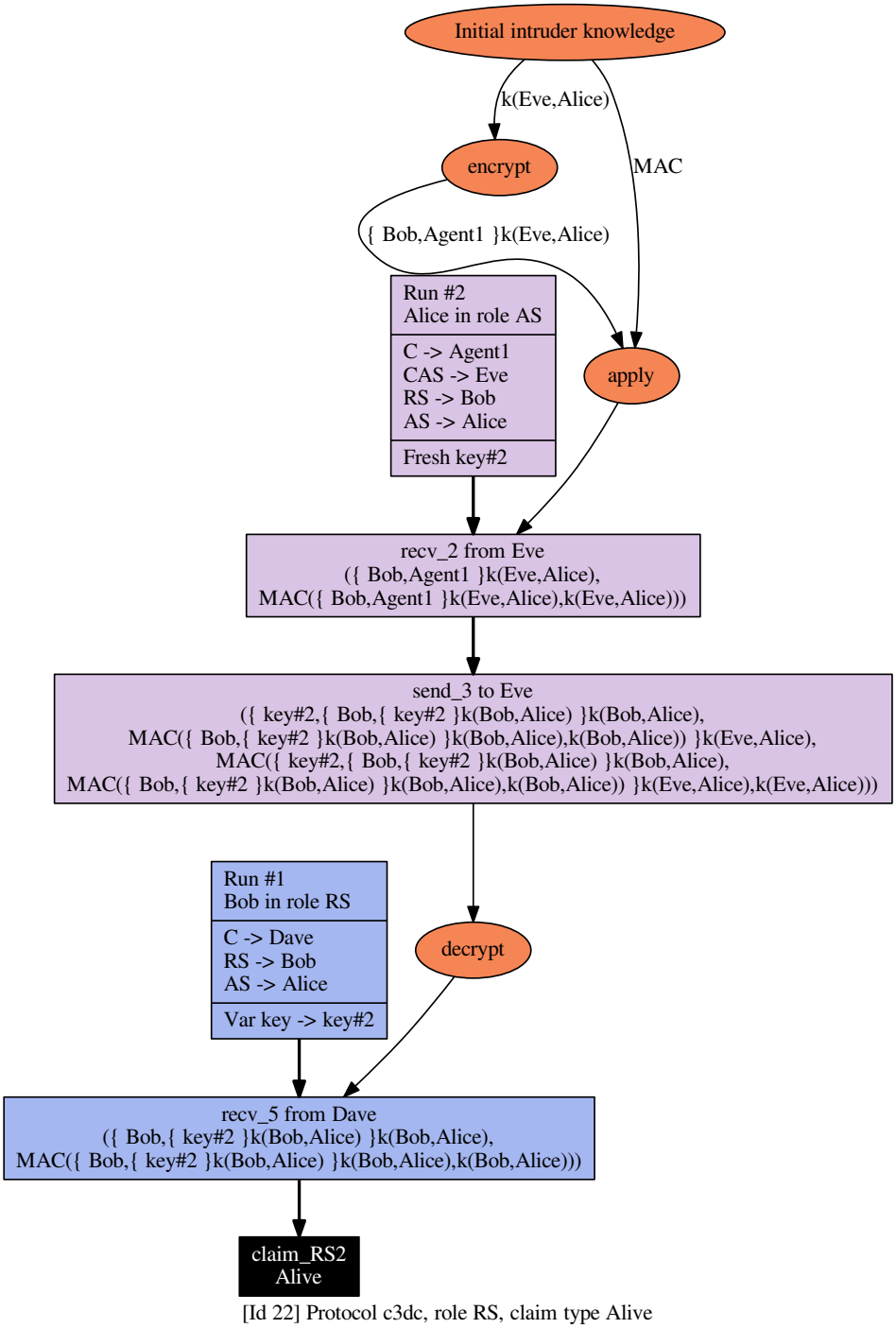

Abbildung 4.9: Darstellung des Angriffsgraphen des Aliveness-Claims des Resource Servers, welcher mittels Scyther generiert wurde.

Scyther falsifiziert darüber hinaus das Aliveness-, sowie das Weak Agreement-Claim des Resource Servers. Dies wird in der Abbildung 4.9 durch einen Angriffs-

graphen dargestellt. Hierbei wird derselbe Graph für das Aliveness-, als auch für das Weak Agreement-Claim generiert[58]. In dem Graphen werden zwei Instanzen des Protokolls aufgezeigt. Ausgeführt wird die erste Instanz (Run #1) von Bob als Resource Server, wohingegen Alice den Authorization Server in der zweiten Instanz (Run #2) ausübt. Deutlich wird hierbei, dass der Client und der Client Authorization Server beide nicht am Protokoll teilnehmen. Dies führt zunächst zur Falsifizierung des Aliveness-Claims. Hierbei lässt sich insbesondere anmerken, dass der Resource Server annimmt von dem Client Dave die fünfte Nachricht (recv_5 from Dave) zu erhalten, allerdings diese Nachricht von Eve konstruiert wird. Weiterführend wird das Weak Agreement-Claim falsifiziert, da die Einigung über die Teilnahme am Protokoll dementsprechend nicht erfolgt.

In dem Graphen in Abbildung A.8 wird vorausgesetzt, dass es der nicht vertrauenswürdigen Eve möglich ist, sich als Client Authorization Server auszugegeben und sich so Zugriff auf das Schlüsselmaterial zu verschaffen. Da sowohl der Client Authorization Server als auch der Authorization Server mit Informationen von ihren Overseeing Principals ausgestattet werden, die eine gegenseitige Authentifizierung nach sich ziehen, wird dieses Verhalten als unwahrscheinlich betrachtet.

4.4.4 Weitere Ergebnisse

Die Ergebnisse, die von Scyther generiert wurden, geben Auskünfte darüber, dass der Client und der Resource Server eher passiv am Protokoll teilnehmen, wodurch die zweite Erwartung aus Abschnitt 4.2 verletzt wird. Ebenfalls geht hervor, dass die im Zuge dieser Arbeit erstellte Spezifikation die Prüfung der Autorisierung zwischen den Akteuren nicht berücksichtigt und dementsprechend die Spezifikation den Sicherheitszielen nicht komplett gerecht wird. Dies kann jedoch auf das benutzte Werkzeug zurückgeführt werden, da der Sachverhalt in diesem Werkzeug nicht modelliert werden kann.

Die Analyse des ACE-OAuth-Frameworks durch L. Arnaboldi et al. [AT19] zeigt die Verifizierung des Frameworks. Daraus geht hervor, dass der Schlüsselaustausch geschützt erfolgt. Für das C3DC-Profil lässt sich jedoch nur vermuten, dass dieses ebenfalls sicher ist, da die Angriffsgraphen dies lediglich andeuten. Somit kann nicht eindeutig festgestellt werden, ob die erste Erwartung aus Abschnitt 4.2 gehalten werden kann.

Hinsichtlich der identifizierten Unklarheiten lässt sich dies in Vergleich setzen mit der Veröffentlichung von S. Gerdes et al. [GBB18a]. Hierin werden Unklarheiten im ACE-OAuth-Framework, sowie im CoAP-DTLS-Profil beschrieben. Diese lassen sich darauf zürückführen, dass nicht genügend Informationen für bestimmte Sachverhalte vorhanden sind, wie dies im C3DC-Profil auch der Fall ist. Entsprechend werden die folgenden Lösungsansätze vorgeschlagen.

[58] Dementsprechend wird dieser Graph nicht zusätzlich im Anhang hinzugefügt, da dieser keinen Mehrwert für diese Arbeit schafft.

4.5 Lösungsansätze für das C3DC-Profil

Die Auswertung der Ergebnisse der Analyse des C3DC-Profils mittels Scyther hat Schwachstellen identifiziert. Für diese Schwachstellen sollen im folgenden Lösungsansätze beschrieben werden. Hierbei wird sich zunächst auf Unklarheiten bezogen, die im C3DC-Profil auftreten, bevor Ansätze hinsichtlich der Aktivität des Resource Servers und des Clients beschrieben werden.

1. **Access Request:**
   In Abschnitt 4.2 wird beschrieben, dass weder das C3DC-Profil noch das ACE-OAuth-Framework spezifizieren, wie der Access Request des Clients an den Client Authorization Server erfolgt beziehungsweise welche Informationen dieser beinhaltet. Dementsprechend wird in dieser Arbeit die Annahme getroffen, dass diese Anfrage sich wie der Access Token Request verhält. Um Missverständnisse, beispielsweise bei der Implementierung des Protokolls, zu vermeiden, sollte die Autorenschaft Ergänzungen diesbezüglich treffen. Diese Ergänzungen sollten den Access Request näher beschreiben und die Informationen darlegen, die zwischen dem Client und dem Client Authorization Server ausgetauscht werden. Zur Verdeutlichung sei hier unter anderem der Unterabschnitt 5.6.1 der ACE-OAuth-Spezifikation [SSW+19, Seite 22 ff.] genannt, worin der Client-to-AS Request beschrieben wird. Hierbei wird zum Ende des Abschnitts hin eine beispielhafte Anfrage dargestellt. Eine ähnliche Umsetzung wird für das C3DC-Profil als sinnvoll erachtet, hierbei sei dies nicht auf den Access Request beschränkt, sondern auf das gesamte Profil.

2. **Definition des Begriffs securely:**
   Das C3DC-Profil verwendet den Begriff securely, um zu beschreiben, dass das ausgetauschte Schlüsselmaterial geschützt versendet werden muss. Hieraus wird jedoch nicht ersichtlich, wie der Begriff zu verstehen ist beziehungsweise inwiefern das Schlüsselmaterial geschützt wird. Die Frage, die offen bleibt, ist, ob hierunter der Vertraulichkeitsschutz, der Integritätsschutz oder beides verstanden wird. Daher wird, wie in Abschnitt 4.2 beschrieben, eine Annahme getroffen, wie dieser Begriff in dieser Arbeit verstanden wird. Um auch hier Missverständnisse zu vermeiden, sollte das C3DC-Profil explizit definieren, was unter securely verstanden wird. Eine mögliche Umsetzung wird im ACE-OAuth-Framework unter anderem im Unterabschnitt 6.2 beschrieben. Hier wird explizit ausgesagt, welcher Schutz angewendet werden muss, um den Inhalt des Access Tokens zu schützen [SSW+19, Seite 42]. Dies sollte auch für die Kommunikation zwischen dem Client Authorization Server und dem Authorization Server, sowie zwischen dem Client Authorization Server und dem Client explizit definiert werden.

3. **Beteiligung des Clients im C3DC-Profil:**
   Ausgehend von der Auswertung in Unterabschnitt 4.4.1 hat sich gezeigt, dass das Aliveness-, sowie das Weak Agreement-Claim des Client Authorization Servers falsifiziert wurden. Zurückgeführt wurde dies auf eine mangelnde Beteiligung des Clients an diesem Prozess und auf die Unklarheiten bezüglich des Access Requests und des DTLS-Handshakes. Sollten

diese Unklarheiten behoben werden, könnte dies gegebenenfalls bereits zur Behebung führen.
Wird die mangelnde Beteiligung des Clients betrachtet, so könnte hier andernfalls ein ähnlicher Ansatz verwendet werden, wie in der ACE-OAuth-Spezifikation beschrieben. Hierbei werden für die optionalen Schritte ein Client-Nonce-Parameter verwendet [SSW+19, Seite 19]. Dieser dient dem Resource Server dazu, nur aktuelle Token zu akzeptieren [SSW+19, Seite 19]. Da dieser Parameter über die ganze Aushandlung hinweg in den Nachrichten beziehungsweise im Access Token benutzt werden muss, kann angenommen werden, dass die Rolle damit aktiv am Protokoll teilnehmen würde. Entsprechend könnte ein ähnlicher Ansatz gewählt werden, wodurch der Client aktiv an diesem Prozess teilnimmt.

4. **Beteiligung des Resource Servers im C3DC-Profil:**
Die Beteiligung vom Resource Server in der Abstraktion des C3DC-Profils ist passiv. Da nicht beschrieben wird, inwieweit der DTLS-Handshake im Zuge des C3DC-Profils ausgeführt wird, wurde lediglich die Versendung des Access Tokens durch den Client berücksichtigt. Entsprechend wird vorgeschlagen, dass dieser Handshake ausführlicher im C3DC-Profil beschrieben wird.
Alternativ lässt sich hier ebenfalls der Ansatz des Client-Nonce-Parameter betrachten, welcher in den optionalen Schritten verwendet wird. Hierbei müsste überlegt werden, ob ein ähnlicher Ansatz genutzt werden könnte, um die aktive Teilnahme des Resource Servers am Protokoll zu gewährleisten. Ein zusätzlicher Schritt müsste dafür im C3DC-Profil hinzugefügt werden.

In ihrer Veröffentlichung beschreiben S. Gerdes et al. [GBB18a] ebenfalls Unklarheiten hinsichtlich des untersuchten Profils und heben die Wichtigkeit der Behebung dieser hervor. Entsprechend sollten die hier identifizierten Unklarheiten ausgebessert werden. Ebenso sollten Überlegungen angestellt werden, wie der Resource Server und der Client aktiver am Protokoll teilnehmen können unter Berücksichtigung der zuvor beschriebenen Ansätze.

5 Fazit

Im Zuge dieser Arbeit sollte eine formale Analyse des C3DC-Profils des ACE-OAuth-Frameworks durchgeführt werden. Hierbei sollte insbesondere festgestellt werden, ob dieses Profil den Annahmen entsprechend sicher ist und keine mögliche Sicherheitslücken aufweist. Um dieses Ziel zu erreichen, sollte eine Testspezifikation entwickelt werden, die eine Abstraktion des C3DC-Profils darstellt. Diese Spezifikation sollte für das Scyther-Werkzeug entwickelt werden, mit dessen Hilfe anschließend die Analyse durchgeführt wurde. Das Werkzeug wurde gewählt, da es bereits eine ähnliche Analyse für ein Protokoll des unterliegenden ACE-OAuth-Frameworks gibt. Diese geht aus der Arbeit von H. Tschofenig, wie in Abschnitt 3.3 beschrieben, hervor. Hierbei sollte zunächst festgestellt werden, ob diese Grundlage dem aktuellen Framework entspricht und ob diese eine Erweiterung auf das C3DC-Profil zulässt. Zum Abschluss sollte die Analyse ausgewertet und Lösungsansätze für mögliche Probleme vorgestellt werden.

Die Entwicklung der Testspezifikation ging aus den vorliegenden Spezifikationen des ACE-OAuth-Frameworks und des C3DC-Profils hervor. Hierbei wurden die Informationen der Spezifikationen genutzt, um den Verlauf des Protokolls modellieren zu können. Anschließend wurden bestimmte Eigenschaften gefordert, die das Protokoll erfüllen muss, um den Anforderungen an die Sicherheit gerecht zu werden.

Bei der anschließenden Auswertung wurden die vom Scyther-Werkzeug generierten Diagramme untersucht. Diese geben Aufschluss darüber, wie die identifizierten Angriffe zustande kommen. Hieraus lassen sich die jeweiligen Probleme und mögliche Lösungsansätze ableiten.

Die Betrachtung der Analyse des Client-Token-Protokolls hat gezeigt, dass diese Implementierung nicht mehr der aktuellen Version des ACE-OAuth-Frameworks entspricht. Dementsprechend wurden Vorschläge unterbreitet, die beschreiben, welche Änderungen vorgenommen werden müssten, um der aktuellen Version zu entsprechen. Eine Erweiterung der Analyse des Client-Token-Protokolls auf das C3DC-Profil war dementsprechend nicht direkt möglich.

Die Analyse des C3DC-Profils in dieser Arbeit ergab, dass es Designschwächen hinsichtlich der Aktivität des Clients und des Resource Servers gibt. Darüber hinaus hat sich in der Analyse gezeigt, dass die Prüfung der Autorisierung der Akteure in der Spezifikation nicht berücksichtigt wird und somit die Angriffsgraphen Verwundbarkeiten hinsichtlich des Schlüselmaterials zeigen. Unter genauer Betrachtung der Angriffsgraphen konnte jedoch festgestellt werden, dass diese Graphen daraufhin deuten, dass das Schlüsselmaterial geschützt ausgetauscht

wird. Zurückgeführt wurde dieser Fehler auf eine fehlende Modellierungsmöglichkeit seitens des benutzten Werkzeugs Scyther.

Aus dieser Arbeit gehen Lösungsansätze hervor, die darauf abzielen die Verständlichkeit der Spezifikation zu verbessern, um die identifizierten Unklarheiten im C3DC-Profil zu beseitigen. Außerdem werden Lösungsansätze vorgestellt, die es dem Client und dem Resource Server ermöglichen sollen, aktiv an dem Protokoll teilzunehmen.

Das Ziel dieser Arbeit wurde insofern erreicht, als dass eine Testspezifikation für das C3DC-Profil erstellt und eine Analyse des Protokolls durchgeführt wurde. Hierbei muss jedoch berücksichtigt werden, dass die Überprüfung der Autorisierung der Akteure fehlt und somit die Spezifikation nicht komplett abgebildet wurde. Dennoch lässt die Analyse Rückschlüsse auf mögliche Lösungsansätze zu. Darüber hinaus wurde gezeigt, dass das CTP nach H. Tschofenig nicht der aktuellen Version des ACE-OAuth-Frameworks entspricht.

Für das weitere Vorgehen mit dieser Arbeit wird vorgeschlagen, dass eine weitere Analyse des C3DC-Profils vorgenommen wird. Dies wird für notwendig gehalten, da diese Arbeit die Überprüfung der Autorisierung der Akteure nicht berücksichtigen konnte. Um dementsprechend eine vollständige Analyse des Protokolls zu erhalten, wird vorgeschlagen eine Analyse mit einem mächtigeren Werkzeug vorzunehmen. Hierbei könnte die in Abschnitt 3.4 vorgestellte Arbeit von L. Arnaboldi et al. Berücksichtigung finden, da diese eine Grundlage für das ACE-OAuth-Framework unter Nutzung des Tamarin-Werkzeugs angefertig haben.

Für zukünftige Arbeiten wird es darüber hinaus als sinnvoll erachtet, frühzeitig die Einschränkungen der benutzten Werkzeuge in Erfahrung zu bringen, um entsprechend alle Aspekte des Protokolls modellieren zu können.

Darüber hinaus sollten die identifizierten Probleme, sowie die beschriebenen Lösungsansätze von der Autorenschaft des C3DC-Profils betrachtet werden. Hierbei sollte insbesondere diskutiert werden, ob die genannten Aspekte kritisch sind, ob eine Überarbeitung des Profils notwendig ist und diese gegebenenfalls durchzuführen.

A Anlagen

A.1 Implementierung des CTPs in Scyther

```
usertype SessionKey;

protocol clienttoken (C, RS, AS)
{
  role C    %C: Client
  {
    var key: SessionKey;

    send_1(C, RS, C);
    recv_4(RS, C, {key}k(C, AS));

    claim_C1(C, Secret, key);
    claim_C3(C, Alive);
    claim_C4(C, Weakagree);
    claim_C5(C, Niagree);
    claim_C6(C, Nisynch);
  }
  role RS   %R: Resource Server
  {
    var key: SessionKey;

    recv_1(C, RS, C);
    send_2(RS, AS, {C}k(RS, AS));
    recv_3(AS, RS, {key, {key}k(C, AS)}k(RS, AS));
    send_4(RS, C, {key}k(C, AS));

    claim_RS1(RS, Secret, key);
    claim_RS3(RS, Alive);
    claim_RS4(RS, Weakagree);
  }
  role AS   %A: Authorization Server
  {
    fresh key: SessionKey;

    recv_2(RS, AS, {C}k(RS, AS);
    send_3(AS, RS, {key, {key}k(C, AS)}k(RS, AS));
  }
}
```

Codedarstellung A.1: Implementierung des CTPs in Scyther nach H. Tschofenig [Tsc18].

A.2 Implementierung des C3DC-Profils in Scyther

```
0  // Usertype as defined in the foundation for
   // this work [SSW+16, p. 5].
2  usertype SessionKey;

4  // Message Authentication Code for protecting
   //  the content of the Access Token and for
6  //  integrity protecting messages.
   hashfunction MAC;
8
   // Access Token
10 //  containing the key (self-contained, meant for RS)
   //  [GBB18b, p. 3, SSW+19, p. 68]
12 //  and the intended recipient (RS) [SSW+19, p. 41].
   //  Since the AT may contain a symmetric key, the key
14 //  must be encrypted [SSW+19, p. 41].
   //  AT is encrypted between RS and AS and integrity
16 //  protected using a digital signature or
   //  a keyed message digest (MAC) [SSW+19, p. 41, 46].
18 //  Established keying material between the AS
   //  and the RS allows the AS to apply cryptographic
20 //  protection to the AT [SSW+19, p. 12].
   macro AT = {RS, {key}k(RS, AS)}k(RS, AS), MAC({RS, {key}k(RS, AS
      )}k(RS, AS), k(RS, AS));
22 // Access Information
   //  containing the key (meant for C) [GBB18b, p. 3].
24 macro AI = key;

26 // C: Client
   // CAS: Client Authorization Server
28 // RS: Resource Server
   // AS: Authorization Server
30 protocol c3dc (C, CAS, RS, AS)
   {
32 role C
   {
34   var key: SessionKey;

36   // Send: Access Request
     //  It is assuemd that C specifies which RS the
38   //  request will target as he would do when request-
     //  ing an AT at the AS directly [SSW+19, p. 65].
40   //  Since there is an established DTLS Session between
     //  C and CAS it is assumed that the communication is
42   //  encrypted [GBB18b, p.6].
     send_1(C, CAS, {RS}k(C, CAS));
44   // Receive: Token Transfer
     recv_4(CAS, C, {AI, AT}k(C, CAS), MAC({AI, AT}k(C, CAS), k(C,
      CAS)));
46   // Send: DTLS KeyExchangeMessage
     //  containing AT [GBB18b, p. 7, GBB+18c, p. 4, 10].
```

```
48    //  Since there is no established DTLS Session nor
      //  keying material between C and RS, the communication
50    //  is not encrypted.
      send_5(C, RS, AT);
52
      // Key has to be secret.
54    claim_C1(C, Secret, key);
      // Every Role has to complete a run of the
56    // protocol with C [Low97, p. 2].
      claim_C2(C, Alive);
58    // Every Role has to agree that they were running
      // the protocol with C [Low97, p. 3].
60    claim_C3(C, Weakagree);
  }
62  role CAS
  {
64    var key: SessionKey;

66    // Receive: Access Request
      recv_1(C, CAS, {RS}k(C, CAS));
68    // Send: Token Request
      //  It is assumed that CAS is acting like C in SSW+19.
70    //  CAS specifies which RS the request
      //  will target [SSW+19, p. 65].
72    //  CAS provides Information about C
      //  [GBB18b, p. 6, GBB+18c, p. 5].
74    //  Since there is an established TLS-/DTLS-Session
      //  between CAS and AS, the message is encrypted.
76    //  Since CAS and AS are meant to communicate securely,
      //  it is assumed they are using encryption and
78    //  integrity protection (MAC) to achieve this.
      send_2(CAS, AS, {RS, C}k(CAS, AS), MAC({RS, C}k(CAS, AS), k(
       CAS, AS)));
80    // Receive: Token Grant
      recv_3(AS, CAS, {AI, AT}k(CAS, AS), MAC({AI, AT}k(CAS, AS), k(
       CAS, AS)));
82    // Send: Token Transfer
      //  containing the AT and the AI [GBB18b, p. 7].
84    //  Since there is an established DTLS Session
      //  between CAS and C, the transfer of the AT
86    //  and the AI is encrypted [GBB18b, p. 7].
      //  The transfer of the AI and the AT to C
88    //  is encrypted and signed [GBB18b, p. 6].
      //  It is assumed that the term securely [GBB18b, p. 6]
90    //  means using encryption and integrity protection (MAC).
      send_4(CAS, C, {AI, AT}k(C, CAS), MAC({AI, AT}k(C, CAS), k(C,
       CAS)));
92
      // Key has to be secret.
94    claim_CAS1(CAS, Secret, key);
      // Every Role has to complete a run of the
96    // protocol with CAS [Low97, p. 2].
```

```
    claim_CAS2(CAS, Alive);
98  // Every Role has to agree that they were running
    // the protocol with CAS [Low97, p. 3].
100 claim_CAS3(CAS, Weakagree);
    // The Roles have to agree on the data
102 // of the exchanged messages [CM12, p. 47-50].
    claim_CAS4(CAS, Niagree);
104 // The send and receive events have to occur in
    // the specified order and have to contain the
106 // specified content {CM12, p. 47-50].
    claim_CAS5(CAS, Nisynch);
108 }
    role RS
110 {
    var key: SessionKey;
112
    // Receive: DTLS KeyExchangeMessage
114 recv_5(C, RS, AT);

116 // Key has to be secret.
    claim_RS1(RS, Secret, key);
118 // Every Role has to complete a run of the
    // protocol with RS [Low97, p. 2].
120 claim_RS2(RS, Alive);
    // Every Role has to agree that they were running
122 // the protocol with RS [Low97, p. 3].
    claim_RS3(RS, Weakagree);
124 }
    role AS
126 {
    fresh key: SessionKey;
128
    // Receive: Token Request
130 recv_2(CAS, AS, {RS, C}k(CAS, AS), MAC({RS, C}k(CAS, AS), k(
     CAS, AS)));
    // Send: Token Grant
132 //  AS creating AT and adding AI to
    //  the response [GBB18b, p. 6].
134 //  The transfer of the AI and the AT to CAS
    //  is encrypted and integrity protected (MAC) [GBB18b, p. 6].
136 //  It is assumed that the term securely [GBB18b, p. 6]
    //  means using encryption and integrity protection.
138 send_3(AS, CAS, {AI, AT}k(CAS, AS), MAC({AI, AT}k(CAS, AS), k(
     CAS, AS)));
    }
140 }
```

Codedarstellung A.2: Implementierung des C3DC-Profils in Scyther.

A.3 Angriffsgraphen der Analyse des C3DC-Profils

A.3.1 Claim Secret für den Client

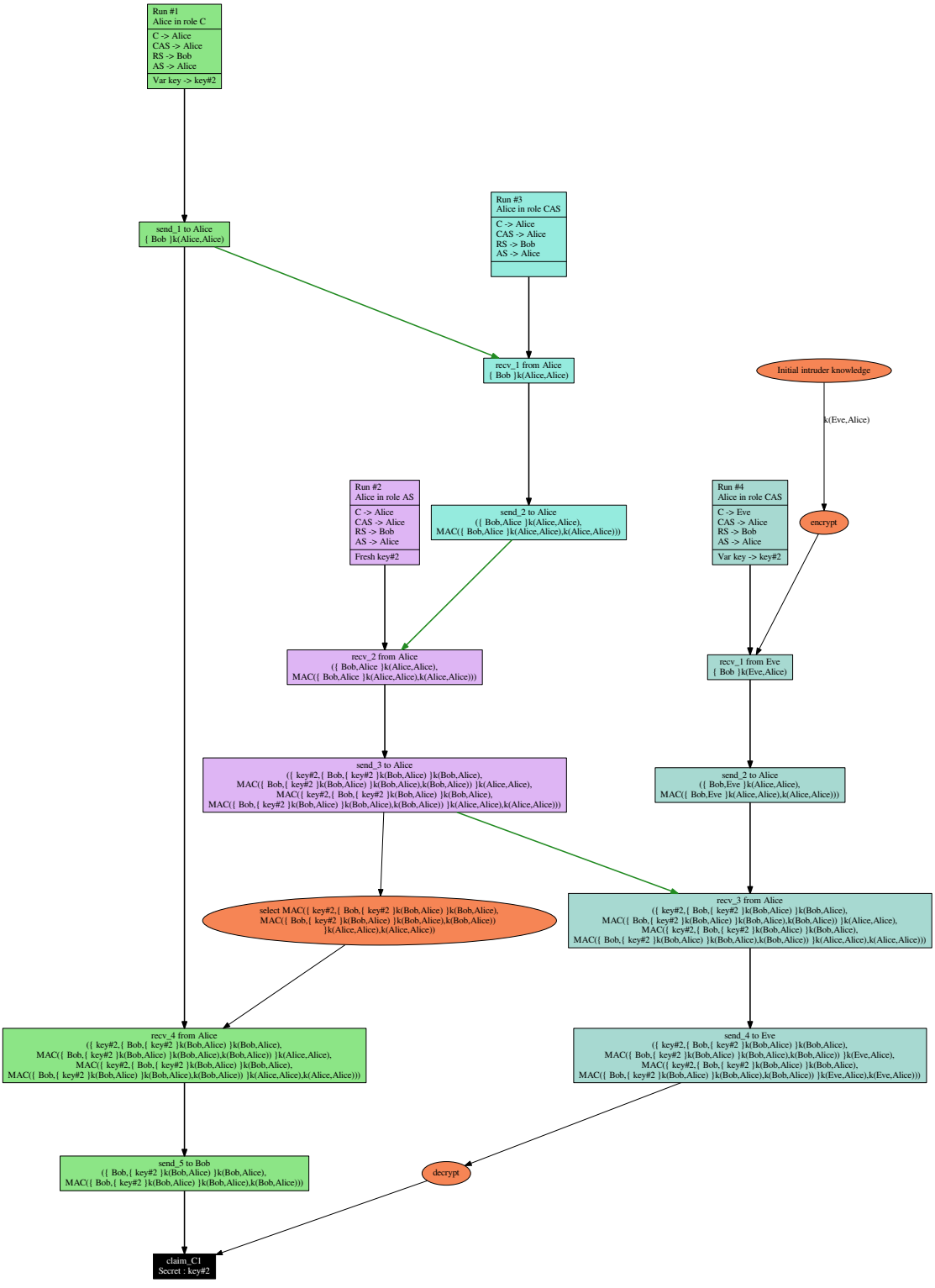

Abbildung A.1: Darstellung des Angriffsgraphen des Secret-Claims des Clients, welcher mittels Scyther generiert wurde.

A.3.2 Claim Alive für den Client

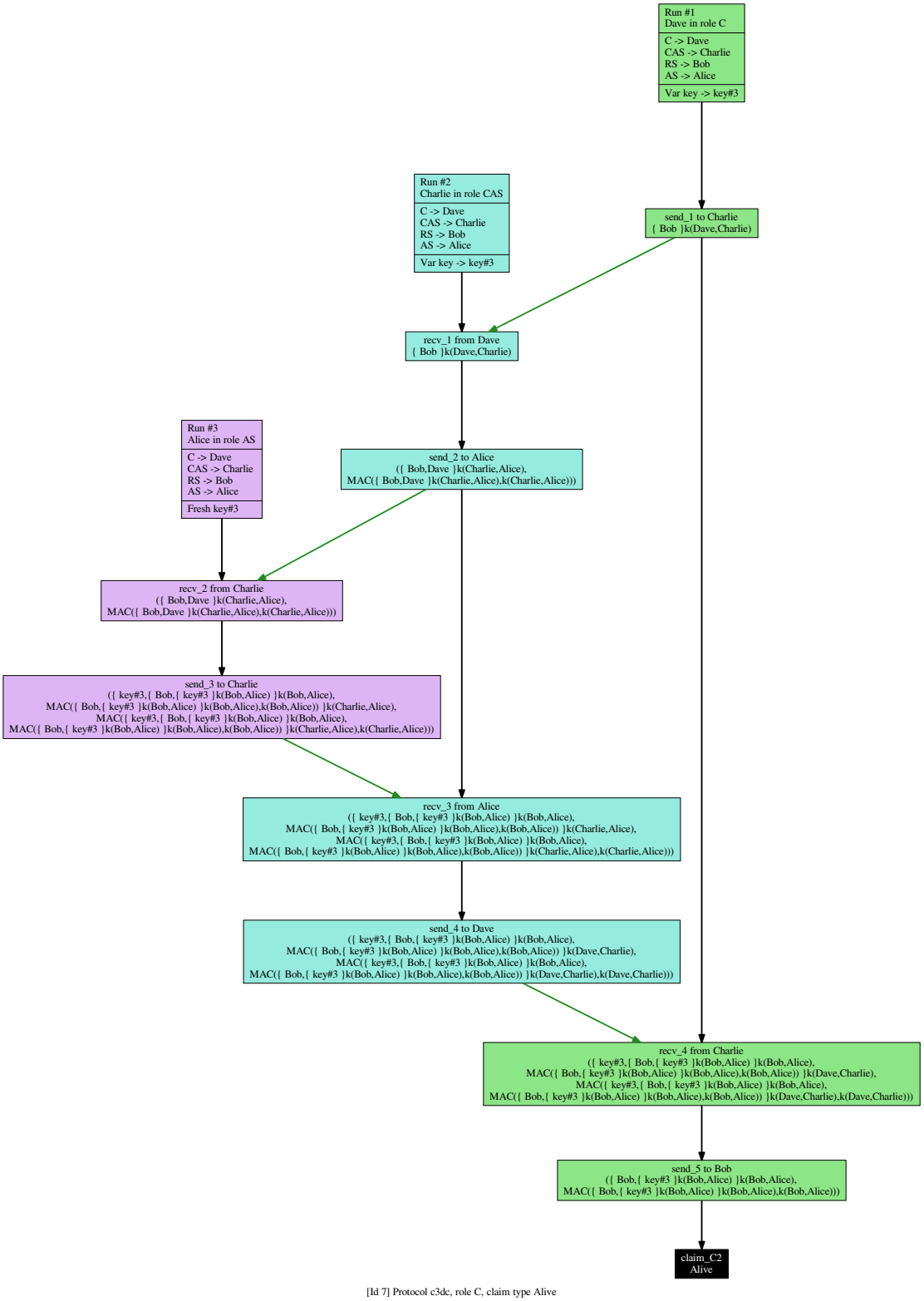

Abbildung A.2: Darstellung des Angriffsgraphen des Aliveness-Claims des Clients, welcher mittels Scyther generiert wurde.

A.3.3 Claim Secret für den Client Authorization Server

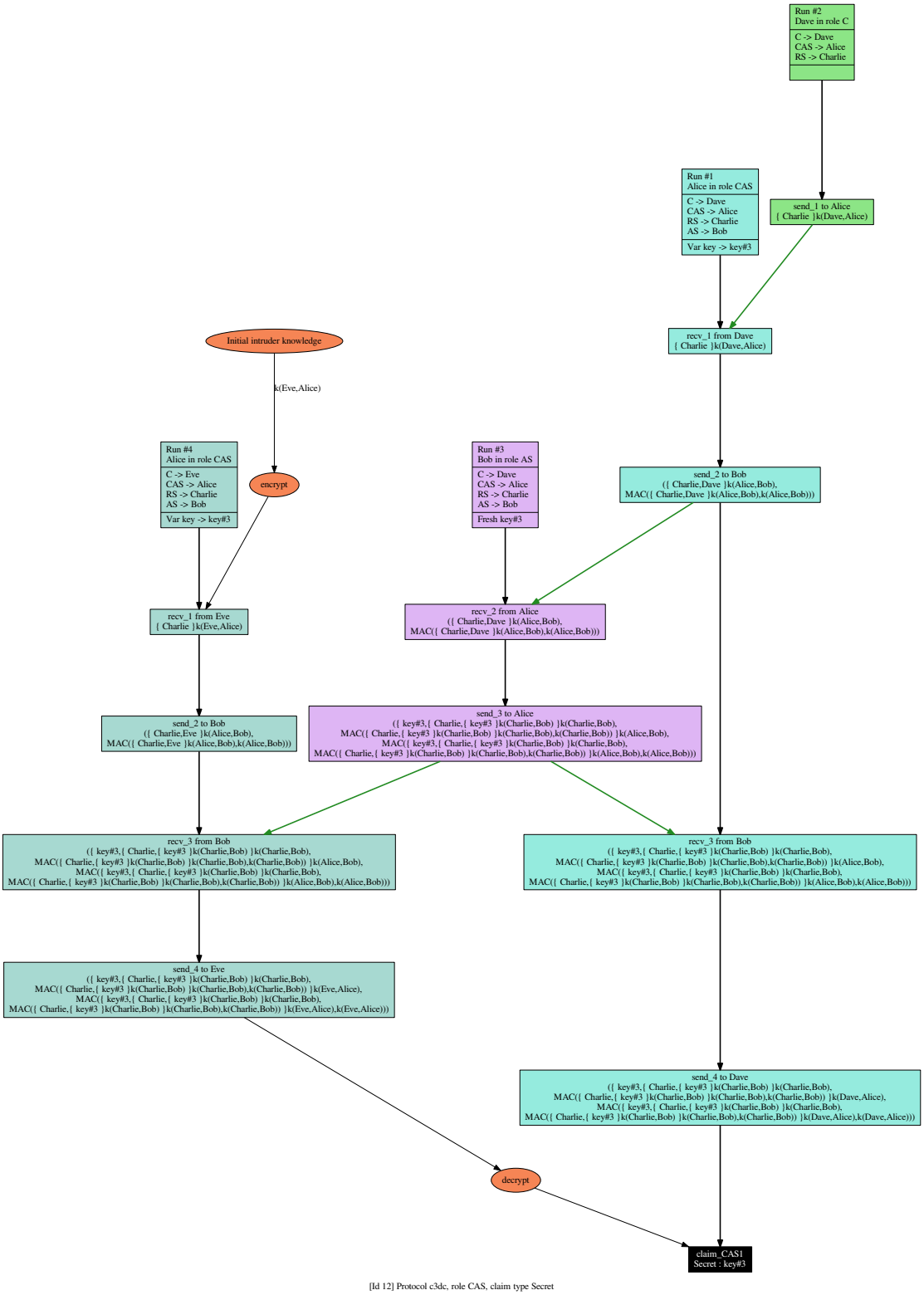

Abbildung A.3: Darstellung des Angriffsgraphen des Secret-Claims des Client Authorization Servers, welcher mittels Scyther generiert wurde.

A.3.4 Claim Weakagree für den Client Authorization Server

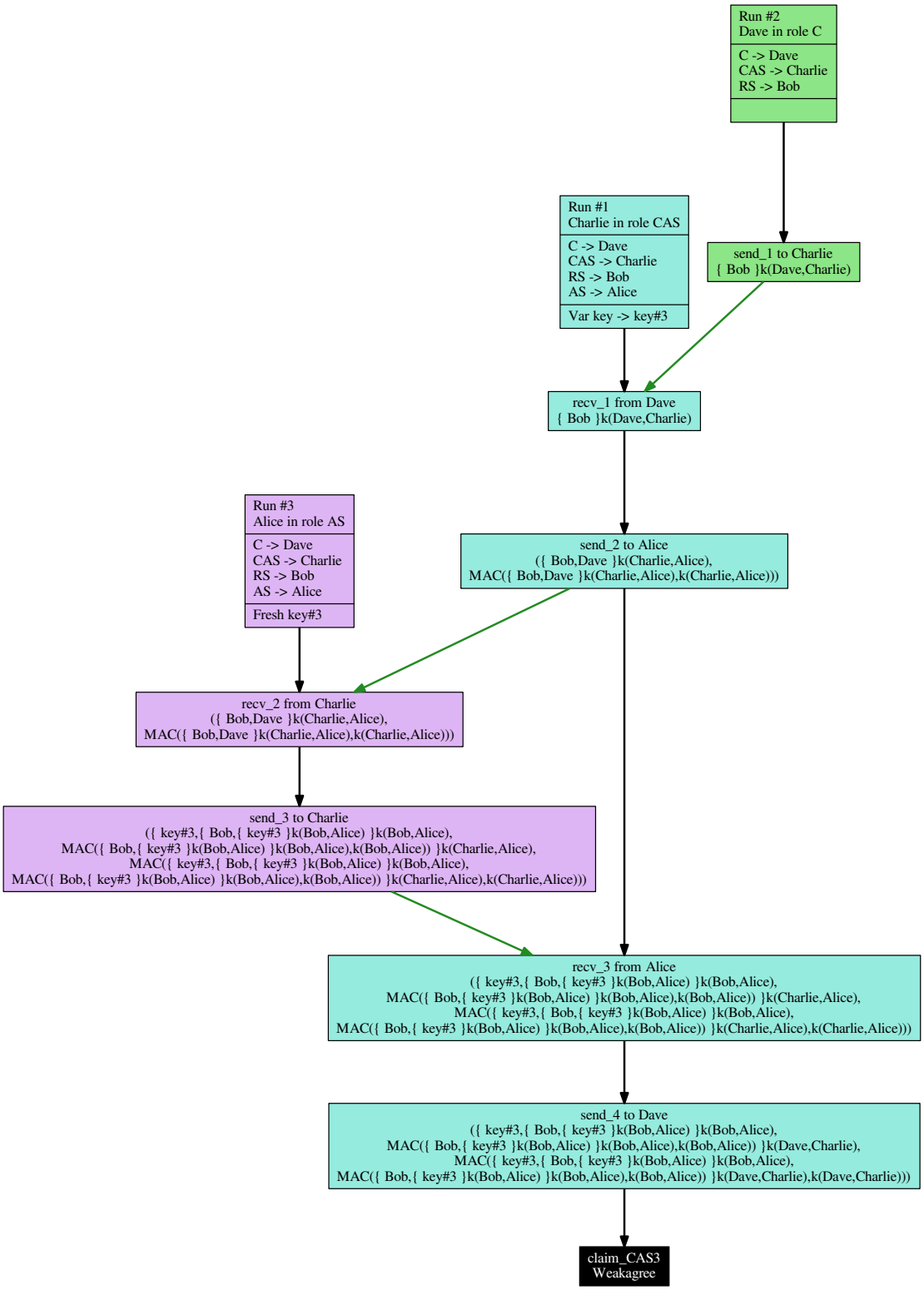

Abbildung A.4: Darstellung des Angriffsgraphen des Weak Agreement-Claims des Client Authorization Servers, welcher mittels Scyther generiert wurde.

A.3.5 Claim Niagree für den Client Authorization Server

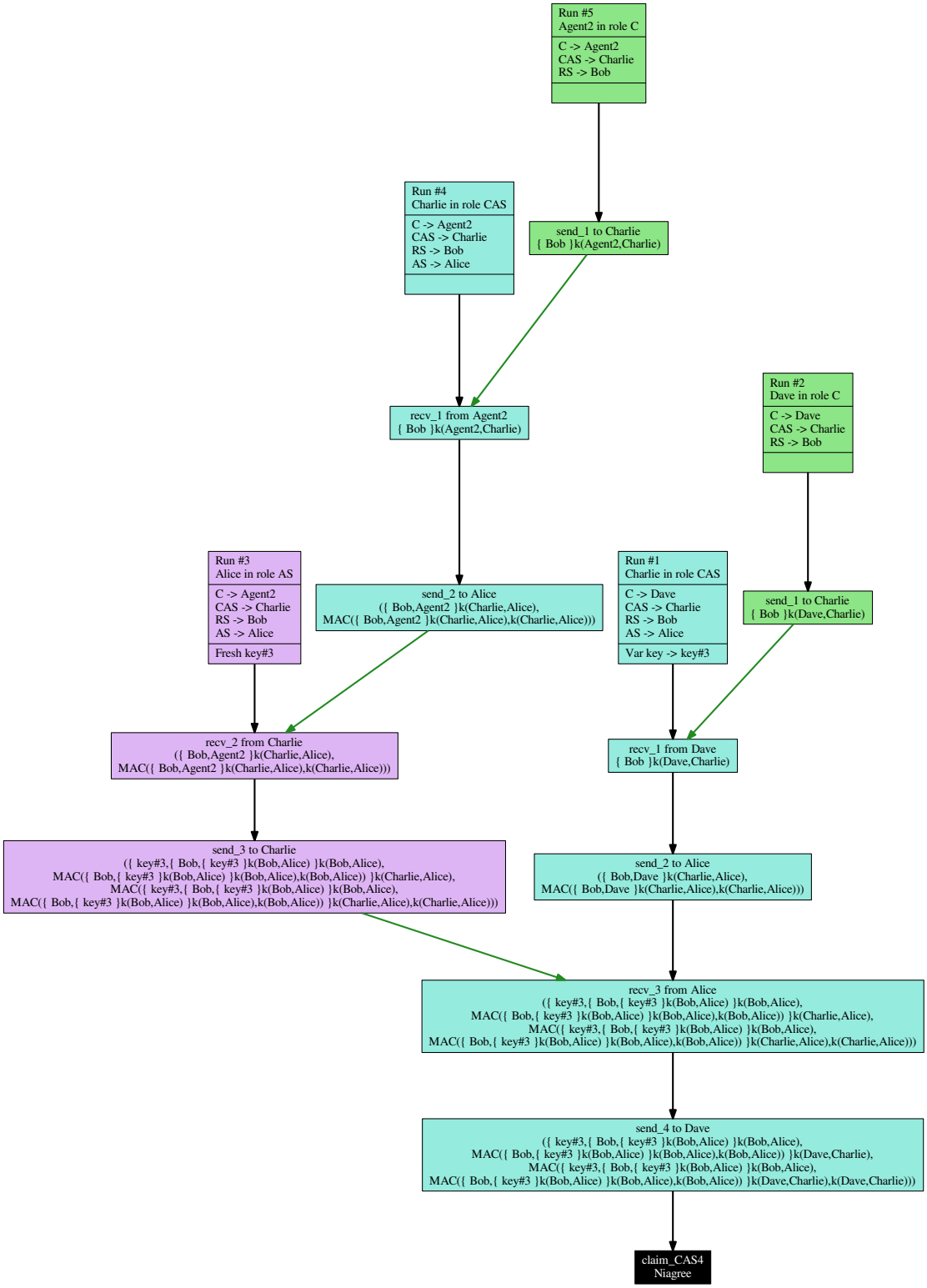

Abbildung A.5: Darstellung des Angriffsgraphen des Non-injective Agreement-Claims des Client Authorization Servers, welcher mittels Scyther generiert wurde.

A.3.6 Claim Nisynch für den Client Authorization Server

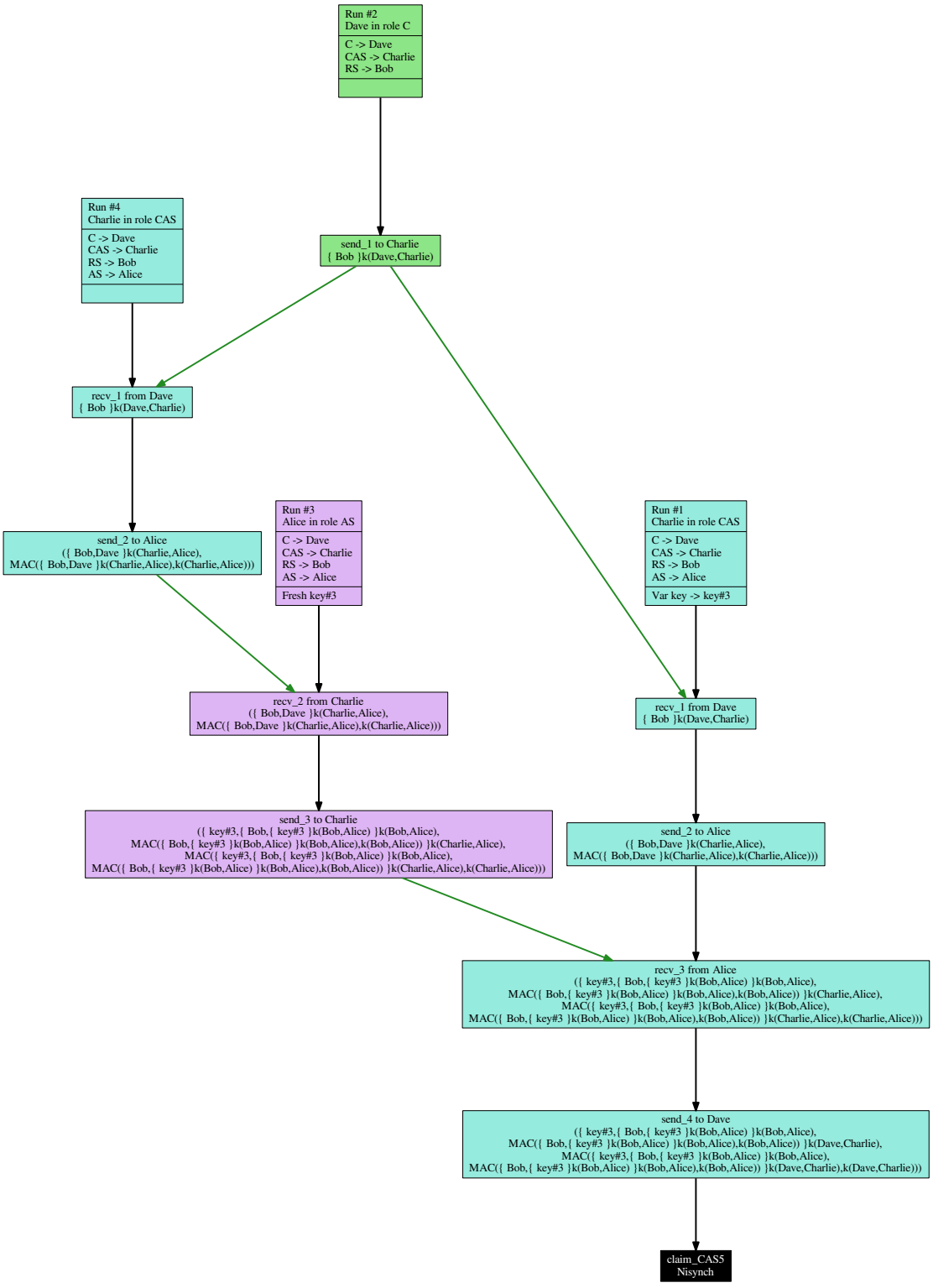

Abbildung A.6: Darstellung des Angriffsgraphen des Non-injective Synchronisation-Claims des Client Authorization Servers, welcher mittels Scyther generiert wurde.

A.3.7 Claim Secret für den Resource Server

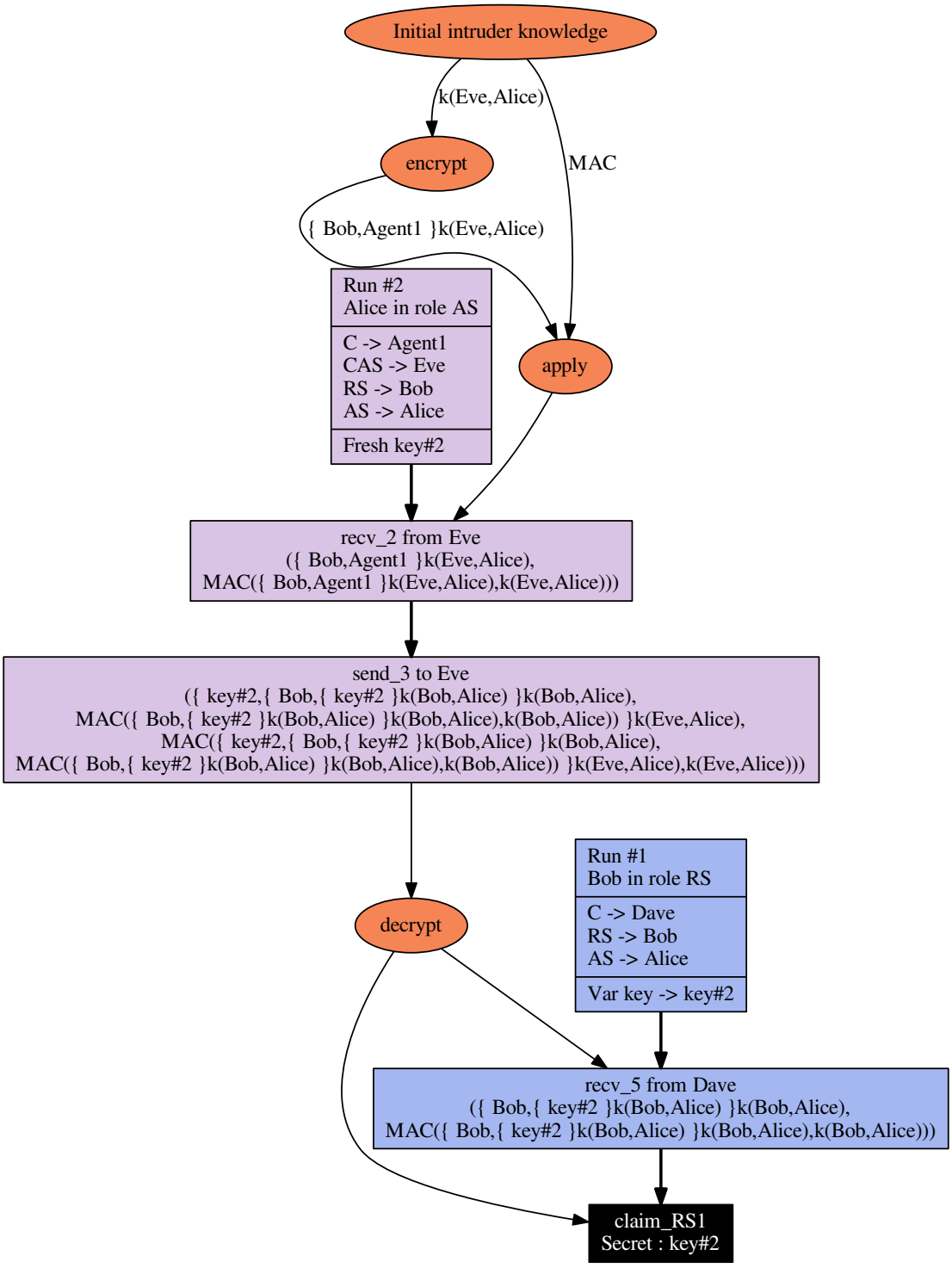

Abbildung A.7: Darstellung des Angriffsgraphen des Secret-Claims des Resource Servers, welcher mittels Scyther generiert wurde.

A.3.8 Claim Aliveness für den Resource Server

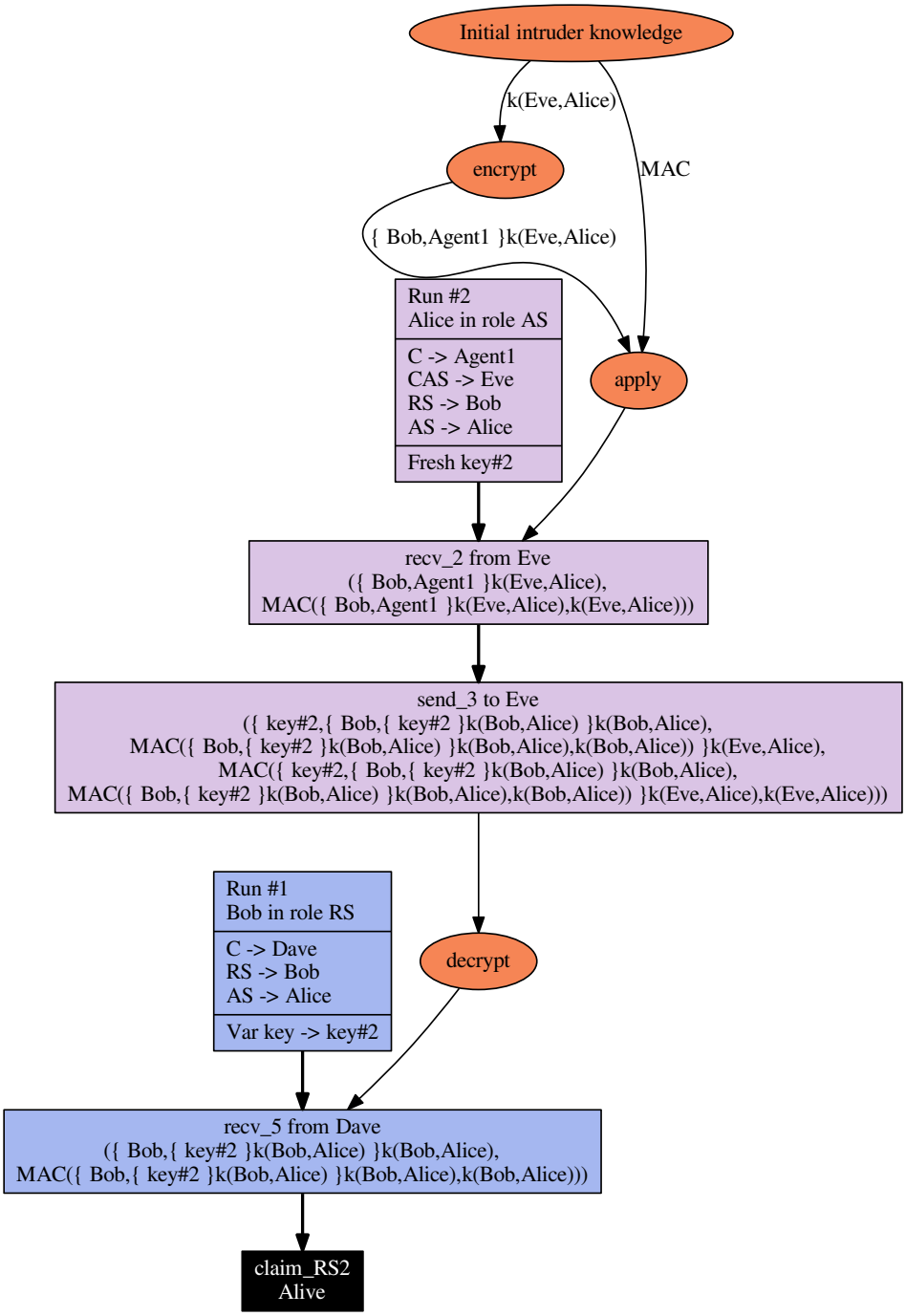

Abbildung A.8: Darstellung des Angriffsgraphen des Aliveness-Claims des Resource Servers, welcher mittels Scyther generiert wurde.

B Literaturverzeichnis

[ABB+05] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P. C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, und L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In K. Etessami und S. K. Rajamani, editors, *Computer Aided Verification*, pages 281–285, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[AT19] L. Arnaboldi und H. Tschofenig. A Formal Model for Delegated Authorization of IoT Devices Using ACE-OAuth. https://sec.uni-stuttgart.de/_media/events/osw2019/slides/arnaboldi_tschofenig-ace_auth_final.pdf, 2019.

[BCM18] D. Basin, C. Cremers, und C. Meadows. *Model Checking Security Protocols*, pages 727–762. Springer International Publishing, Cham, 2018. https://doi.org/10.1007/978-3-319-10575-8_22.

[CM12] C. Cremers und S. Mauw. *Operational Semantics and Verification of Security Protocols*. Springer Berlin Heidelberg, 2012.

[Cre08a] C. J. F. Cremers. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In A. Gupta und S. Malik, editors, *Computer Aided Verification*, pages 414–418, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Cre08b] C. J.F. Cremers. Unbounded Verification, Falsification, and Characterization of Security Protocols by Pattern Refinement. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 119–128, New York, NY, USA, 2008. ACM. http://doi.acm.org/10.1145/1455770.1455787.

[Cre14a] C. Cremers. Scyther User Manual. https://github.com/cascremers/scyther/blob/master/gui/scyther-manual.pdf, 2014.

[Cre14b] C. Cremers. The Scyther Tool. https://people.cispa.io/cas.cremers/scyther/, 2014.

[ET05] P. Eronen und H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279, RFC Editor, December 2005. https://tools.ietf.org/html/rfc4279.

[GBB15] S. Gerdes, O. Bergmann, und C. Bormann. Delegated CoAP Authentication and Authorization Framework (DCAF). Internet-Draft draft-gerdes-ace-dcaf-authorize-04, Internet Engineering Task Force, October 2015. Work in Progress.

[GBB18a] S. Gerdes, O. Bergmann, und C. Bormann. Avoiding Gaps in Authorization Solutions for the Internet of Things. In *Proceedings of the NDSS Workshop on Decentralized IoT Security and Standards (DISS)*, 2018. DOI 10.14722/diss.2018.23006, https://dx.doi.org/10.14722/diss.2018.23006.

[GBB18b] S. Gerdes, O. Bergmann, und C. Bormann. C3DC – Constrained Client/Cross-Domain Capable Authorization Profile for Authentication and Authorization for Constrained Environments (ACE). Internet-Draft draft-gerdes-ace-c3dc-00, Internet Engineering Task Force, October 2018. Work in Progress.

[GBB+18c] S. Gerdes, O. Bergmann, C. Bormann, G. Selander, und L. Seitz. Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE). Internet-Draft draft-ietf-ace-dtls-authorize-05, Internet Engineering Task Force, October 2018. Work in Progress.

[GSSB18] S. Gerdes, L. Seitz, G. Selander, und C. Bormann. An architecture for authorization in constrained environments. Internet-Draft draft-ietf-ace-actors-07, Internet Engineering Task Force, October 2018. Work in Progress.

[KKSL17] M. Kurkowski, A. Kozakiewicz, und O. Siedlecka-Lamch. Some Remarks on Security Protocols Verification Tools. In A. Grzech, J. Świątek, Z. Wilimowska, und L. Borzemski, editors, *Information Systems Architecture and Technology: Proceedings of 37th International Conference on Information Systems Architecture and Technology – ISAT 2016 – Part II*, pages 65–75, Cham, 2017. Springer International Publishing.

[Low96] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. In T. Margaria und B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[Low97] G. Lowe. A Hierarchy of Authentication Specifications. In *Proceedings of the 10th IEEE Workshop on Computer Security Foundations*, CSFW '97, pages 31–43, Washington, DC, USA, 1997. IEEE Computer Society.

[MSCB13] S. Meier, B. Schmidt, C. Cremers, und D. Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In Natasha Sharygina und Helmut Veith, editors, *Computer Aided Verification*, pages 696–701, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-39799-8_48.

[NS78] R. M. Needham und M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM*, 21(12):993–999, December 1978.

[PP10] C. Paar und J. Pelzl. *Message Authentication Codes (MACs)*, pages 319–330. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[SHB14] Z. Shelby, K. Hartke, und C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.

[SSW+16] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, und H. Tschofenig. Authentication and Authorization for Constrained Environments (ACE). Internet-Draft draft-ietf-ace-oauth-authz-02, Internet Engineering Task Force, June 2016. Work in Progress.

[SSW+17] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, und H. Tschofenig. Authentication and Authorization for Constrained Environments (ACE). Internet-Draft draft-ietf-ace-oauth-authz-09, Internet Engineering Task Force, November 2017. Work in Progress.

[SSW+18] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, und H. Tschofenig. Authentication and Authorization for Constrained Environments (ACE). Internet-Draft draft-ietf-ace-oauth-authz-10, Internet Engineering Task Force, February 2018. Work in Progress.

[SSW+19] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, und H. Tschofenig. Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth). Internet-Draft draft-ietf-ace-oauth-authz-24, Internet Engineering Task Force, October 2019. Work in Progress.

[Tsc18] H. Tschofenig. Analyzing the IETF ACE-OAuth Protocol. http://st.fbk.eu/sites/st.fbk.eu/files/osw2018-ace.pdf, 2018.

[YOP16] H. Yang, V. Oleshchuk, und A. Prinz. Verifying group authentication protocols by Scyther. 7:3–19, 2016.

C Stichwortverzeichnis