HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

ita Institute for Internet Technologies and Applications

Password Authenticated Connection Establishment

Studienarbeit Frühjahrssemester 2013

Autoren: Reto Schelbert, Simon Schreiber
Betreuer: Prof. Dr. Andreas Steffen

Ausgabe: Montag, 18. Februar 2013
Abgabe: Freitag, 31. Mai 2013

Aufgabenstellung

Password Authenticated Connection Establishment with IKEv2

Studenten: Reto Schelbert & Simon Schreiber
Betreuer: Prof. Dr. Andreas Steffen
Ausgabe: Montag, 18. Februar 2013
Abgabe: Freitag 31. Mai 2013

Einführung

Der Internet Standard RFC 6631 definiert PACE, ein neuartiges Authentisierungsverfahren, das zur Familie der Zero Knowledge Password Proof (ZKPP) Protokolle gehört, die es erlauben relativ schwache Passwörter beim Aufbau von IKEv2-basierten VPN Verbindungen zu verwenden. Die erstaunlich hohe Sicherheit wird dadurch erreicht, dass völlig zufällige Daten mit dem User Passwort verschlüsselt werden, so dass ein Angreifer nicht rausfinden kann, wann er auf das richtige Passwort gestossen ist.

Dieses Verfahren soll für die Open Source strongSwan VPN Lösung in der Programmiersprache C implementiert werden.

Aufgabenstellung

- Einarbeitung in den PACE RFC 6631 und die relevanten Teile des IKEv2 RFC 5996.
- Bekanntmachen mit der strongSwan Software Architektur und den Coding Rules.
- Implementation der PACE Funktionalität und Integration in die strongSwan Software.
- Im Minimum zu implementierende Funktionalität:
  - 4.1 Encrypting the Nonce: Unterstützung aller Verschlüsselungsalgorithmen mit Ausnahme von AEAD-basierten Verfahren (z.B. AES-GCM, AES-CCM).
- Optional zu implementierende Funktionalität mit abnehmender Wichtigkeit:
  - 3.5 Creating a Long-Term Secret
  - 3.6 Using the Long-Term Secret
  - 4.1 Encrypting the Nonce: Verschlüsselung mit dem entsprechendem Counter-Mode bei AEAD-basierten Verfahren.
  - 6.9 Long-Term Credential Storage of weak SPwd
- Nicht zu implementierende Funktionalität:
  - 5.7 PACE and Session Resumption
  - Appendix B. Password Salting

Links

- D. Kuegler, Y. Scheffer, *Password Authenticated Connection Establishment for IKEv2*, RFC 6631, June 2012,
  http://tools.ietf.org/html/rfc6631

- T. Kivinen, *Secure Password Framework for IKEv2*, RFC 6467, December 2011, **http://tools.ietf.org/html/rfc6467**
- C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen, Internet Key Exchange Protocol Version 2 (IKEv2), RFC 5996, September 2010, **http://tools.ietf.org/html/rfc5996**
- strongSwan Wiki **http://wiki.strongswan.org/**
- strongSwan Software Repository **http://git.strongswan.org/**

Rapperswil, 18. Februar 2013

Prof. Dr. Andreas Steffen

Erklärung über die eigenständige Arbeit

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- das wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Ort, Datum: Rapperswil, 31.05.2013

Reto Schelbert

Ort, Datum: Rapperswil, 31.05.2013

Simon Schreiber

Abstract

Abteilung	Informatik
Namen der Studierenden	Reto Schelbert Simon Schreiber
Studienjahr	FS 2013
Titel der Studienarbeit	Password Authenticated Connection Establishment with IKEv2
Examinator	Prof. Dr. Andreas Steffen
Themengebiet	Internet-Technologien und -Anwendung
Institut	Institute for Internet Technologies and Applications

Ziel der Studienarbeit war es, Password Authenticated Connection Establishment (PACE) für die Authentisierung im Internet Key Exchange Version 2 (IKEv2) in die OpenSource VPN-Lösung strongSwan zu implementieren. PACE stellt ein neues symmetrisches passwortbasiertes Authentisierungsverfahren dar, das gegenüber Pre-Shared Key (PSK) den Vorteil hat, dass die Sicherheit nicht von der kryptografischen Stärke des Passworts abhängt. Dies wird erreicht, indem mit dem Passwort völlig zufälliges Material verschlüsselt und übertragen wird. In einem zweiten Schritt bilden beide Kommunikationspartner einen neuen Diffie-Hellman (DH) Generator mit Hilfe dieses Zufallsmaterials und leiten daraus ein neues Shared Secret ab, das zur Authentisierung verwendet wird. PACE ist in RFC6631 definiert und zählt zu einer Reihe von Verfahren, die als Secure Password Method zusammengefasst werden. Damit sich zwei Kommunikationspartner auf ein solches Verfahren einigen können, ist eine Erweiterung von IKEv2 notwendig, die in RFC6467 als Secure Password Framework for IKEv2 beschrieben ist. Das Framework definiert zudem mit Generic Secure Password Method (GSPM) einen neuen Payload Type, der es ermöglicht, für Secure Password Methods spezifische Daten auszutauschen.
Die im Rahmen dieser Studienarbeit erfolgte Implementierung des GSPM Frameworks bietet eine Grundlage dafür, strongSwan um weitere secure password methods zu erweitern.
PACE wurde als Plugin für strongSwan in C realisert. Damit ist strongSwan in der Lage, PACE zur Authentisierung zu verwenden. Das Erzeugen des DH Shared Secret kann mit dem modularen Diffie-Hellman Verfahren (MODP DH) erfolgen. Mögliche Fortsetzungen nach Abschluss dieser Studienarbeit wären die Nutzung von Elliptic Curve Diffie-Hellman (ECDH) für das Shared Secret sowie das in RFC6631 vorgeschlagene Generieren und Speichern eines Long-Term Shared Secret, das die lokale Speicherung des schwachen initialen Passworts überflüssig machen würde.

Management Summary

Ausgangslage

Kurze Passwörter sind praktisch, weil einfach zu erinnern und zu übermitteln, stellen aber eine Gefahrenquelle dar: Es ist für einen Angreifer in relativ kurzer Zeit möglich, das Passwort zu erraten, wenn er den damit verschlüsselten Verkehr mitschneidet. Er braucht nur alle möglichen Passwörter darauf anzuwenden: Sobald einzelne Zeichen im entschlüsselten Klartext häufiger vorkommen als andere, ist der richtige Schlüssel gefunden. Passwort Authenticated Connection Establishment (PACE) verhindert dies, indem mit dem Passwort eine völlig zufällige Zeichenfolge verschlüsselt wird. Damit kann der Angreifer mit obigem Verfahren nicht mehr entscheiden, wann er auf das richtige Passwort gestossen ist: Die Zeichen im Klartext bleiben beim richtigen und falschen Schlüssel gleich häufig. Weil die zufällige Zeichenfolge somit trotz schwachem Passwort sicher übertragen werden kann, dient diese als Grundlage für neue Schlüssel, mit denen sich zwei Kommunikationspartner sicher authentisieren können. Ziel dieser Studienarbeit war die Implementierung dieses Verfahrens zur Authentisierung in die OpenSource VPN-Lösung strongSwan.

Vorgehen, Technologien

Damit sich zwei Kommunikationspartner auf das PACE-Verfahren zur Authentisierung einigen können, ist eine Änderung im Ablauf des in Internet Key Exchange Version 2 (IKEv2) notwendig. Auch die ausgetauschten Meldungen unterscheiden sich von den herkömmlichen Verfahren. Nach erfolgreicher Implementierung dieser Erweiterungen in strongSwan wurde eine Erweiterung entwickelt, die die Authentisierung zweier Kommunikationspartner mittels PACE bewerkstelligt. Das Projekt wurde in der Programmiersprache C realisiert.

Ergebnisse

Nach Abschluss unserer Studienarbeit steht für strongSwan eine Erweiterung zu Verfügung, mit der sich eine sichere Authentisierung auch mit einem schwachen Passwort erfolgreich durchführen lässt. Das Passwort wird dabei aus der Konfiguration gelesen. Mit unserer Implementierung ist auch die Grundlage für die Anbindung weiterer Verfahren geschaffen, die ein schwaches Passwort erlauben (sogenannte „Secure Password Methods“).

Ausblick

Ein mögliches Folgeprojekt wäre das Generieren und Speichern eines starken und wiederverwendbaren Passworts mittels PACE. Damit müsste das initiale schwache Passwort nicht mehr in der Konfiguration gesichert werden. Unsere Lösung lässt eine Erweiterung dahingehend zu, dass zur Ableitung der Authentisierungs-Schlüssel ein weiteres Verfahren (Elliptic Curve Diffie-Hellman) zum Einsatz kommt.

Inhalt

1 Einleitung

Im Wesentlichen ist das Ziel dieser Studienarbeit, den IKEv2 Daemon `charon` von strongSwan um die Fähigkeiten des Secure Password Frameworks zu erweitern und eines der Secure Password Verfahren, nämlich Password Authenticated Connection Establishment (PACE), zu realisieren. Nachfolgend möchten wir unseren Technischen Bericht zu dieser Implementierung vorstellen und auf ein paar Besonderheiten aufmerksam machen.

1.1 Aufbau der Dokumentation

Die vorliegende Dokumentation ist in folgende Kapitel gegliedert:

Grundlagen	In diesem Kapitel werden die Vorkenntnisse vermittelt, die für das Verständnis der Arbeit relevant sind.
Umsetzungskonzept	In diesem Kapitel erläutern wir unsere Vorgehensweise zur Umsetzung der Aufgabenstellung
Design / Implementation	In diesem Kapitel beschreiben wir das konkrete Softwaredesign unserer Arbeit und gehen auf einige Besonderheiten im Detail ein.
Tests	In diesem Kapitel stellen wir Testfälle vor und unser Vorgehen beim Testen wird dokumentiert.
Projektstand	In diesem Kapitel halten wir den Stand der Arbeit fest, analysieren Resultate und zeigen mögliche Weiterentwicklungen auf.
Projektmanagement	In diesem Kapitel werden spezifische Einzelheiten zur Projektdurchführung und Management festgehalten.

1.2 Darstellung

In der Implementation werden einzelne Code-Fragmente (Listings) dargestellt um die Erklärungen zu vervollständigen oder wichtige Passagen hervor zu heben.
Es wird nicht aller Code eins zu eins übernommen, sondern unnötige Codephrasen werden mit **[…]** im Listing ausgelassen um eine bessere Übersicht zu erhalten.
Da es sich um die Programmiersprache C mit objektorientierten Ansätzen handelt, werden UML Diagramme, insbesondere Klassendiagramme, so gut wie möglich auf den Code angepasst um die Erklärungen zu ergänzen und um diese zu veranschaulichen.
Einzelne Begriffe die Protokoll spezifisch sind oder aus dem Source-Code genommen werden in `unterschiedlicher Schriftart` formatiert um die Leserlichkeit zu verbessern.

2 Grundlagen

2.1 Internet Protocol Security (IPsec)

Internet Protocol Security (IPsec) ist eine Protokoll-Suite für den Network Layer, die eine gesicherte Kommunikation über potentiell unsichere IP-Netze wie dem Internet ermöglichen soll. Häufige Anwendungen von IPsec sind Virtual Private Networks (VPN). StrongSwan ermöglicht den Aufbau eines VPN. Nachfolgend gehen wir nur auf die Protokolle ein, welche den Aufbau einer IPsec Verbindung ermöglichen.

2.2 Security Association (SA)

Als Security Association werden alle Parameter bezeichnet, die die Verschlüsselung, Authentisierung und Integritätsprüfung einer IPsec-Verbindung ermöglichen. Eine SA ermöglicht damit eine sichere Übertragung von Daten. Es wird unterschieden zwischen der IKE_SA, die den Austausch von Schlüsseln selbst schützt und den CHILD_SA für den verschlüsselten Austausch der Nutzdaten. Im Wesentlichen werden für das Erstellen einer SA vier kryptografische Algorithmen ausgehandelt:

- Ein Verschlüsselungsalgorithmus
- Ein Algorithmus zur Authentisierung (Integrity Protection)
- Eine Diffie-Hellman Gruppe.
- Eine Pseudo Random Function (PRF, siehe Kapitel 2.4). Diese wird zur Erzeugung von Schlüsselmaterial für alle kryptografischen Algorithmen verwendet.

2.3 Internet Key Exchange Version 2 (IKEv2)

Das Internet Key Exchange Protocol (IKEv2) ist ein in RFC5996 standardisiertes Protokoll, das zur Aushandlung der kryptografischen Parameter für den Aufbau einer IPsec-Verbindung verwendet wird [1]. Es definiert die Aushandlung durch den paarweisen Austausch einer Reihe von Meldungen. Diese lassen sich in Bezug auf den Ablauf des Protokolls und ihrem Zweck in vier grundsätzliche Typen einteilen:

Bezeichnung	Zweck
IKE_SA_INIT	dient der Initialisierung der Verbindung
IKE_AUTH	ist für die gegenseitige Authentisierung der Peers notwendig
CREATE_CHILD_SA	dient zur Änderung der ausgehandelten Schlüssel und der Erstellung von
CHILD_SA	für die Nutzdaten.
INFORMATIONAL	beinhaltet den zusätzlichen Informationsaustausch zwischen den Peers

In jedem Meldungsaustausch wird nach der Richtung unterschieden zwischen dem Initiator, der mit Versenden der Meldungen beginnt und dem Responder, der darauf antwortet. Eine IPsec-Verbindung enthält mindestens zwei Runden (Roundtrips) des Austauschs: Eine Runde im IKE_SA_INIT zum Verbindungsaufbau, der Aushandlung der Krypto-Suiten und der Erstellung der SA für die nächste Runde sowie eine Runde im IKE_AUTH zur gegenseitigen Authentisierung und

der Erzeugung der ersten CHILD_SA für die Nutzdaten. Dabei muss für jedes Subnetz (oder einzelnen Host), für das ein VPN erstellt wird, sowie für jede Richtung separat eine CHILD_SA erstellt werden.
Nachfolgend werden die Meldungstypen näher betrachtet und unterschieden.
Jede Meldung ist aus einem oder mehreren abgrenzbaren Teilen zusammengesetzt, die als Payload bezeichnet werden. In den Grafiken sind diese unterschiedlich eingefärbt. Ist eine solche Payload **rot** umrandet, so erfolgt der Meldungsaustausch in diesem Schritt verschlüsselt.

2.3.1 IKE_SA_INIT

Jeder Verbindungsaufbau beginnt mit der IKE_SA_INIT.
Im IKE_SA_INIT (Abbildung 1) schlägt der Initiator die Krypto-Suiten vor, die er beherrscht (**SA1i**). Zudem sendet er seinen öffentlichen Diffie-Hellman Schlüssel **KEi** sowie eine Zufallszahl (nonce) **Ni**. Der Responder beantwortet in Abbildung 2 den Request mit seinen Krypto-Suiten (**SA1r**), seinem öffentlichen DH-Schlüssel **KEr** und einer eigenen Zufallszahl **Nr**. Optional kann der Responder ein Zertifikat vom Initiator anfordern.

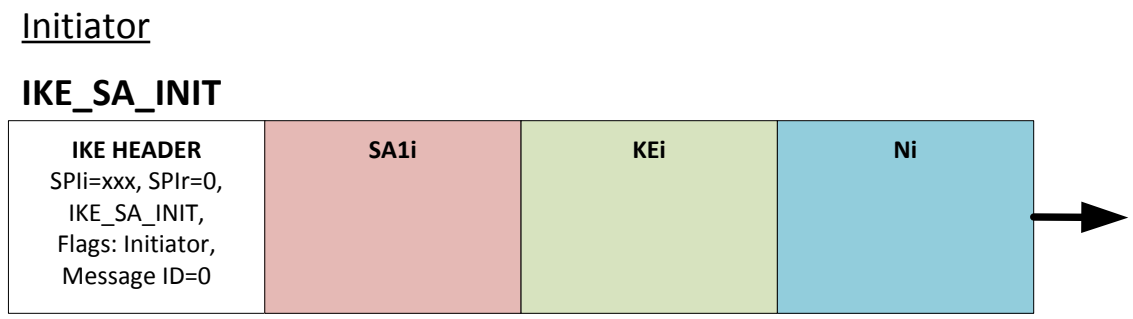

Abbildung 1: IKE_SA_INIT Initiator normaler IKEv2 Verlauf

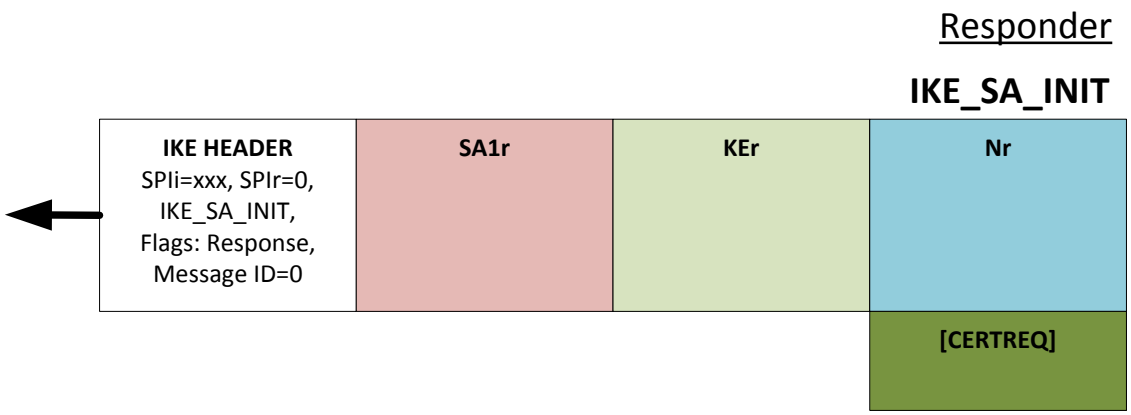

Abbildung 2: IKE_SA_INIT Responder normaler IKEv2 Verlauf

Mit Hilfe des Private Keys und des Public Keys der Gegenseite leiten beide Peers einen gemeinsamen Diffie-Hellman Schlüssel (Shared Secret) aus der in der SA Payload ausgehandelten Gruppe ab. Dieser dient zusammen mit den ausgetauschten Nonces als Grundlage für die Generierung von Schlüsseln für unterschiedliche Zwecke.

Der nachfolgende IKE_AUTH Austausch läuft bereits verschlüsselt (mit **SK_e**) ab und die Integrität der Meldungen ist gesichert (mit **SK_a**). Die Schlüssel sind in Kapitel 2.5 aufgelistet.

2.3.2 IKE_AUTH

Im IKE_AUTH Request des Initiators (Abbildung 3) ist dessen Identität **IDi** und die Authentisierungs-Payload **AUTHi** enthalten. Die Identität kann eine IP- oder eMail-Adresse sein. Das optionale Zertifikat **CERTi** wird nur gesendet werden, falls die Authentisierung über Zertifikate erfolgt. Mit dem ebenfalls optionalen **CERTREQ** Payload kann ein Peer verlangen, dass eine bestimmte **CA** das Zertifikat der Gegenstelle signiert. Mit der **IDr**-Payload kann ein Initiator die Identität des Responders wählen, falls dieser über mehrere Identitäten verfügt.
Die **AUTHi** Payload enthält die Authentisierung. Deren Aufbau ist in 2.6 näher beschrieben.
Die **SAi** Payload enthält wiederum die Krypto-Suiten des Initiators. Mit den **TSi** und **TSr** Payloads werden die Netze angegeben, aus denen Pakete über die Security Association (SA) gesendet und empfangen werden sollen.

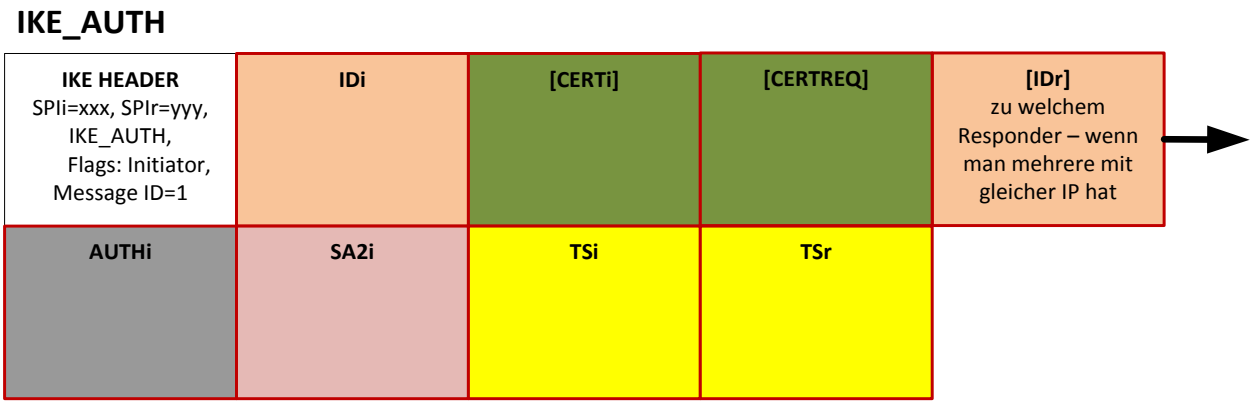

Abbildung 3: IKE_AUTH Initiator normaler IKEv2 Verlauf

Der Message des Responders (Abbildung 4**Fehler! Verweisquelle konnte nicht gefunden werden.**) beinhaltet die gleichen Payloads wie die des Initiators: Die Identität des Responders **IDr**, optional seinem Zertifikat **CERTr** sowie seinen Authentisierungsdaten **AUTHr**.

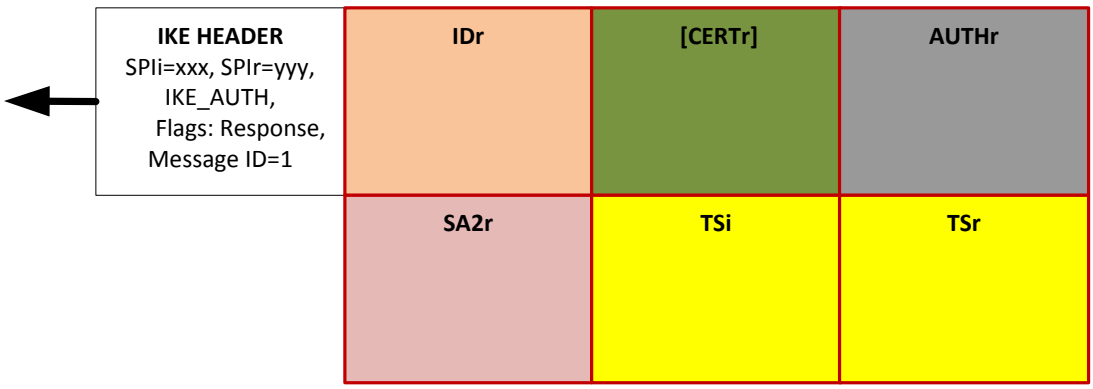

Abbildung 4: IKE_AUTH Responder normaler IKEv2 Verlauf

In **SA2r** wählt der Responder eine Krypto-Suite aus dem vom Initiator vorgeschlagenen Set, womit sich die Peers für die Krypto-Algorithmen für den nachfolgenden Austausch, also die nächste SA, geeinigt haben.
Der Responder kann in den **TSi** und **TSr** Payloads schliesslich die Subnetze einengen, die der Initiator für den Aufbau einer IPsec-Verbindung vorgeschlagen hat.

2.4 Pseudorandom Function

Pseudorandom Functions (prf) werden in IKEv2 und in dieser Dokumentation mehrfach verwendet. Vereinfacht lässt sich sagen, dass eine prf einen Hash-Algorithmus darstellt. Pseudorandom Functions nehmen einen Schlüssel (Key) variabler Länge und eine Zeichenfolge (Seed) als Input und liefern eine Zeichenfolge als Output. Geschrieben wird dies als:

$$Output = prf(< Key >, < Seed >)$$

Formel 1: prf Funktion

Pseudorandom bezieht sich auf den Output, der sich nicht von einer zufällig generierten Zeichenfolge unterscheiden lässt, aber selbst kein zufälliges Material ist.
Wird von einer prf ein längerer Output gefordert als sie liefern kann, so kommt prf+ zum Einsatz. Dazu wird die prf mehrfach nacheinander aufgerufen, wobei an den Seed bei jedem Anruf ein fortlaufender Wert angehängt wird. Die Anzahl der Aufrufe ist durch die Länge des gewünschten Outputs bestimmt:

$$O1\,|O2|\,\ldots = prf + (K, S)$$
$$O1 = prf(K, S \mid 0x01)$$
$$O2 = prf(K, S \mid 0x01)$$
$$\ldots$$

Formel 2: prf+ Funktion

prf+ wird zum Beispiel zur Erzeugung eines Schlüssels eingesetzt, wenn ein Verschlüsselungsalgorithmus einen Schlüssel bestimmter Länge verlangt. Wie in Kapitel 2.3.1 beschrieben, einigen sich zwei IKEv2 Kommunikationspartner auf die Verwendung einer bestimmten prf für eine SA.

2.5 Schlüsselmaterial

Aus dem Shared Secret leiten zwei IKEv2 Kommunikationspartner eine Reihe von Schlüsseln für unterschiedliche Zwecke ab. Dazu wird von beiden Peers die gleiche in IKE_SA_INIT ausgehandelte Pseudorandom Function (prf) verwendet.

Die Schlüssel sind:
SK_e: Encryption, dient zur Verschlüsselung nachfolgenden Meldungen.
SK_a: Authentication, dient zur Authentisierung resp. Integritätssicherung der nachfolgenden Meldungen.

SK_d: Derivation, dient zur Ableitung weiterer Schlüssel für die CHILD_SA
SK_p: Wird für den Aufbau der AUTH Payload gebraucht.
Ausser für den **SK_d** wird pro Richtung wird jeweils ein eigener Schlüssel gebildet, also SKei (Initiator), **SK_er** (Responder) etc.

Für den Rahmen unserer Studienarbeit ist nur der **SK_p** von Belang, für die anderen Schlüssel verweisen wir auf [1].

2.6 Authentisierung: AUTH Payload

Die AUTH Payload enthält die Authentisierung eines Peers gegenüber der Gegenstelle. Je nach eingesetzter Methode für die Authentisierung wird sie unterschiedlich erzeugt. Mit welcher Methode authentisiert wird, ist in der Payload selbst im Feld `Auth Method` enthalten. Der Empfänger richtet sich für die Verifizierung nach dieser `Auth Method`. Sender und Empfänger können also unterschiedliche Methoden für das Erstellen der AUTH Payload verwenden. Es sind die folgenden Varianten möglich:

Variante	AUTH Bildung
Pre-Shared Secret (PSK)	Dabei wird ein Message Authentication Code gebildet.
Signaturen	Erstellen einer Signatur wahlweise mit RSA oder DSS[1].
Extensible Authentication Protocol (EAP)	Die Authentisierung wird über eine externe Stelle, einen Authentication, Authorisation and Accounting (AAA) Server abgewickelt.

Die Daten, über die die Authentisierung gebildet wird, werden als `signed octets` bezeichnet und beinhalten folgende Werte:

- Die gesamte SA_INIT Message des Peers selbst.
- Die empfangene Nonce der Gegenstelle.
- Die eigene ID in der Form `prf(SK_p, ID)`. **SK_p** als Key für die prf wird aus dem erzeugten Schlüsselmaterial des IKE_SA_INIT gewonnen.

[1] Digitale Signaturen von Ravest, Shamir, Adleman (RSA) oder der vom National Institute of Standards and Technology (NIST) empfohlene Digital Signature Standard (DSS)

Authentisiert sich ein Peer mit RSA oder DSS, so besteht die AUTH Payload aus der Signatur über die `signed octets`. Signaturbasierte Verfahren wie RSA oder DSS bilden einen Hash über die `signed octets`, der mit dem gewählten Verfahren signiert wird. Details zum EAP-Verfahren sind in Kapitel 2.16 von RFC5996 zu finden.
Wir stellen hier nur das Verfahren PSK für die Generierung der AUTH Payload näher vor, um in Kapitel 2.8.5 die Unterschiede bei PACE aufzuzeigen.
Wird ein Preshared Secret (PSK) für die Authentisierung verwendet, so wird für die AUTH Payload ein Message Authentication Code (MAC) nach folgendem Vorgehen erzeugt:
Zunächst wird eine prf mit dem Preshared Secret als Key und einem festgelegten String als Seed aufgerufen:

$$prf(Shared\ Secret, \text{Key Pad for IKEv2})$$

Formel 3: AUTH Payload Seed

Der Output dieser prf wird in einem zweiten Schritt als Key für eine neue prf verwendet, mit den `signed octets` als Seed. Der Output der zweiten prf bildet die AUTH Payload:

$$AUTH = prf(prf(Shared\ Secret, "Key\ Pad\ for\ IKEv2"), < InitiatorSignedOctets >)$$

Formel 4: AUTH Payload Data

Verwendet ein Initiator EAP für die Authentisierung, so signalisiert er dies der Gegenstelle, indem die AUTH Payload leer bleibt. Der Responder antwortet mit einer Message, die eine EAP Payload enthält, jedoch keine SA Payload mit der Auswahl der Krypto Suite und keinen Traffic Selector **TS**. Diese werden erst nach erfolgreicher EAP Authentisierung übermittelt.

2.7 Secure Password Framework

2.7.1 GSPM

Neben PACE existieren weitere passwortbasierte Authentisierungsverfahren (PAKE und SPSK_AUTH), die eine symmetrische Authentisierung beider Peers bieten und als Secure Password Methods bezeichnet werden.
Ihnen gemeinsam ist, dass sie symmetrische Verfahren darstellen:
Beide Peers sind also im Besitz des Passworts und die Verbindung kann von jedem der beiden Peers initiiert werden (im Gegensatz zu EAP). Zudem können diese Authentisierungsverfahren nur eingesetzt werden, wenn Sie von beiden Peers verwendet werden.

Da die Einigung auf eines dieser Verfahren vom herkömmlichen IKEv2 Austauschverfahren abweicht, ist für die Aushandlung auf ein solches Verfahren zwischen zwei Peers ein eigenständiges Authentisierungs-Protokoll notwendig, das im Secure Password Framework for IKEv2 definiert ist [2]. Festgelegt wird darin einerseits, wie die Einigung zweier Peers auf ein bestimmtes GSPM-Schlüsselaustausch-Verfahren erfolgt, und andererseits die Art, wie spezifische Daten für ein bestimmtes GSPM-Protokoll zwischen zwei Peers übertragen werden. Es wird dafür ein neuer Payload-Typ Generic Secure Password Method (GSPM) definiert, der als Container für Daten dient, die spezifisch für eine dieser Secure Password Method sind und zwischen den Peers übertragen werden.

2.7.2 Änderung der IKE_SA_INIT

Anders als bei herkömmlichen Verfahren erfolgt die Einigung auf eine Authentisierung mittels Secure Password Method bereits im IKE_SA_INIT Austausch. Möchte sich ein Initiator mittels einem Secure Password Verfahren authentisieren, so hängt er dem herkömmlichen IKE_SA_INIT Message eine weitere NOTIFY[2] Payload hinzu (Abbildung 5), in der er eine oder mehrere Secure Password Verfahren auflistet.

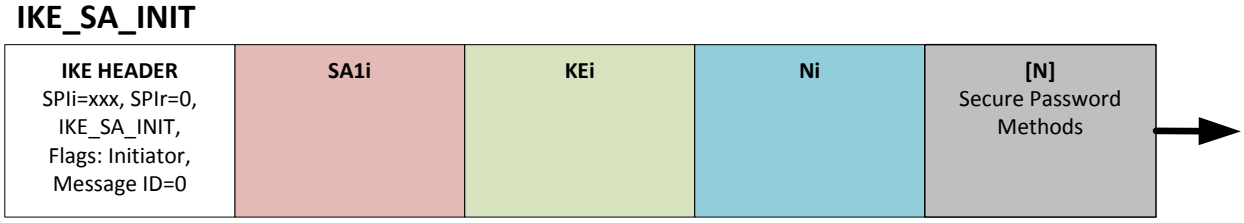

Abbildung 5: IKE_SA_INIT Initiator bei Secure Password Framework

Der Responder beantwortet die Message, indem er seinerseits der IKE_SA_INIT Message eine NOTIFY Payload hinzufügt, die eines der vorgeschlagenen Verfahren enthält (Abbildung 6). Unterstützt der Responder eines dieser Verfahren nicht, schlägt die Authentisierung fehl. Unterstützt er aber mehrere dieser vorgeschlagenen Verfahren, muss er eines selektieren [3]. Welche Verfahren bevorzugt werden, wird in RFC6631 nicht beschrieben und kann somit je nach

[2] NOTIFY Payloads sind gedacht für den Austausch von Informationen und Fehlermeldungen zwischen Peers

Implementierung unterschiedlich ausfallen. Das selektierte Verfahren wird dann von beiden Peers für die Authentisierung benutzt.
Fehlt in der IKE_SA_INIT Message die zusätzliche Payload, so wird vom Initiator ein herkömmliches Verfahren (RSA oder DSS, PSK, EAP) verwendet.

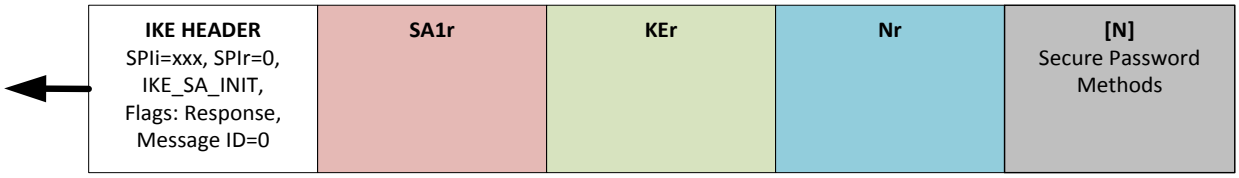

Abbildung 6: IKE_SA_INIT Responder bei Secure Password Framework

Dass die Einigung auf ein GSPM Verfahren bereits im IKE_SA_INIT Austausch erfolgt, hat zwei Gründe. GSPM fügt keine neuen Roundtrips zu IKEv2 hinzu. Deswegen muss die IKE_AUTH Message verändert werden. Zudem wird die IKE_AUTH Message von allen Secure Password Methods in unterschiedlicher Art verändert. Deshalb muss zum Zeitpunkt des Generierens der IKE_AUTH Message bereits ein Verfahren gewählt worden sein.

Die Identität des andern Peers ist erst im IKE_AUTH vorhanden, weswegen die Wahl der Secure Password Method nicht abhängig von der Identität des Initiators erfolgen kann. Einzig dessen IP Adresse ist dem Responder zu diesem Zeitpunkt bekannt.

Für Vorschläge und Auswahl von Secure Password Methods wird kein neuer Payload Typ definiert. Stattdessen wird die Aushandlung innerhalb eines Payloads vom Typ NOTIFY abgewickelt. Grund dafür ist, dass das Typenfeld für eine Payload nur 1 Byte lang ist und von den 255 möglichen Typen eine Mehrzahl bereits besetzt ist.

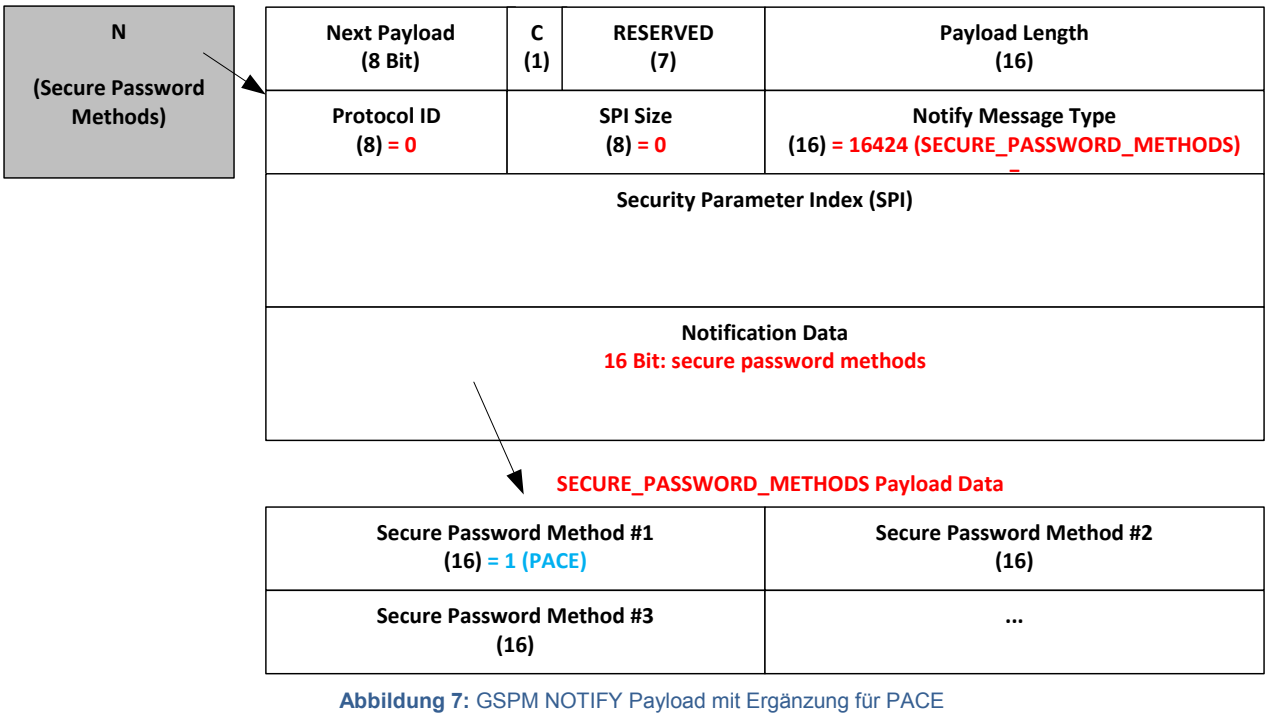

Abbildung 7: GSPM NOTIFY Payload mit Ergänzung für PACE

Die Präferenz für GSPM wird im Feld `Notify Message Type` innerhalb der NOTIFY Payload angezeigt. Der Wert dafür ist **16424**.

Die Liste der vorgeschlagenen GSPM-Methoden (Initiator) resp. die gewählte Methode (Responder) wird im Feld `Notification Data` eingefügt. Die einzelnen Methoden werden mit 16 Bit codiert (siehe Abbildung 7). Die Codierung ist gemäss IANA vorgegeben und für die Methode PACE ist diese **1** (Siehe IANA Assignements [4])

2.7.3 Änderungen in der IKE_AUTH

Da zum Zeitpunkt der Generierung der IKE_AUTH Message bereits eine Einigung über die GSPM Methode erfolgt ist, enthalten Payloads in diesem Schritt nur noch Daten, die für eine bestimmte Methode spezifisch sind. Dafür wird der neue Payload Typ „GSPM Payload“ definiert. Er hat die Nummer **49**. Diese Payload wird je nach gewählter GSPM-Methode ein- oder mehrfach anstelle der AUTH Payload an die Messages im IKE_AUTH hinzugefügt (Abbildung 8).

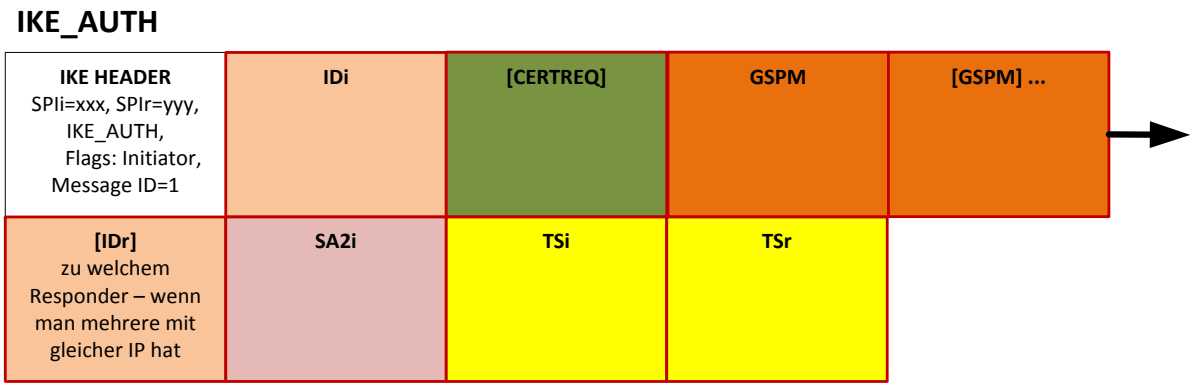

Abbildung 8: IKE_AUTH Initiator bei Secure Password Framework

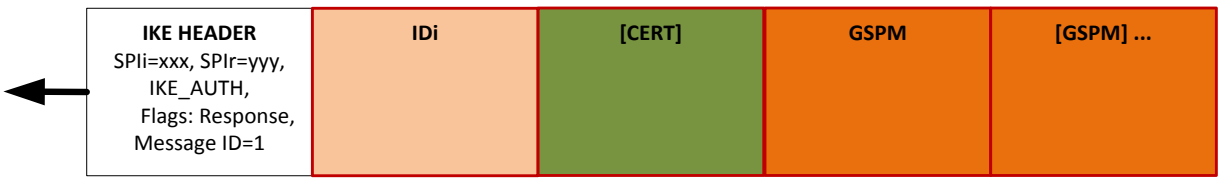

Abbildung 9: IKE_AUTH Responder bei Secure Password Framework

Wiederum je nach gewählter GSPM-Methode sind nach dem ersten IKE_AUTH Exchange mehrere Runden notwendig, in denen die Peers reine GSPM-Payloads austauschen.

Abbildung 10: IKE_AUTH Initiator, weiterer GSPM Austausch

Abbildung 11: IKE_AUTH Responder, weiterer GSPM Austausch

Bei erfolgreicher Authentisierung wird eine AUTH Payload gemäss der gewählten GSPM Methode gebildet und übertragen. Deren Austausch zwischen den beiden Peers beendet die Authentisierung.

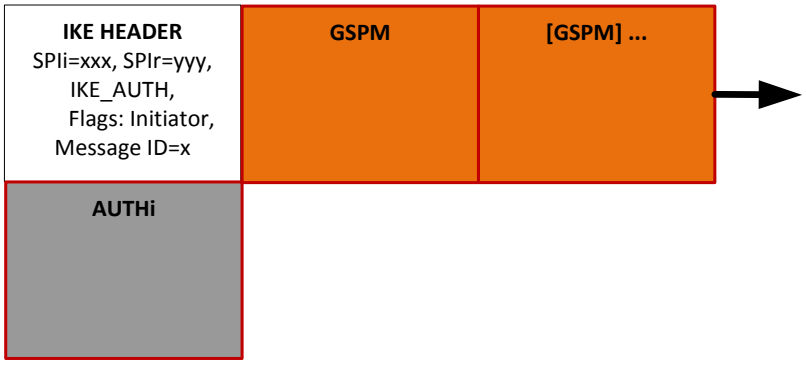

Abbildung 12: IKE_AUTH Initiator Schlussrunde mit AUTH Payload

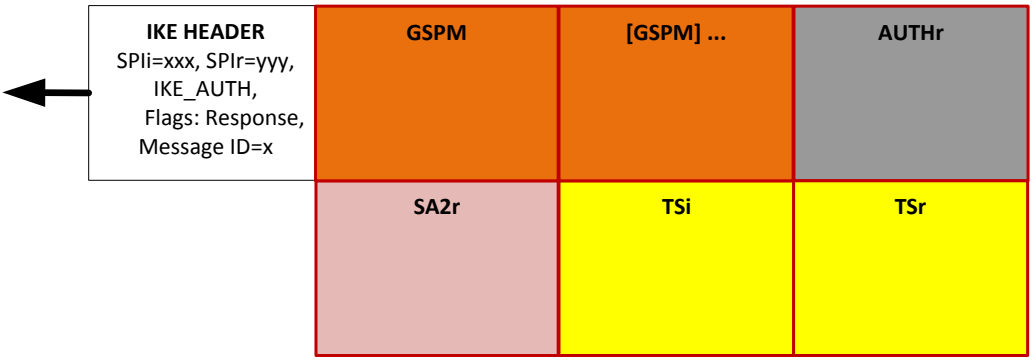

Abbildung 13: IKE_AUTH Responder Schlussrunde mit AUTH Payload

2.7.4 GSPM Payload

Die GSPM Payload selbst hat den folgenden Aufbau:

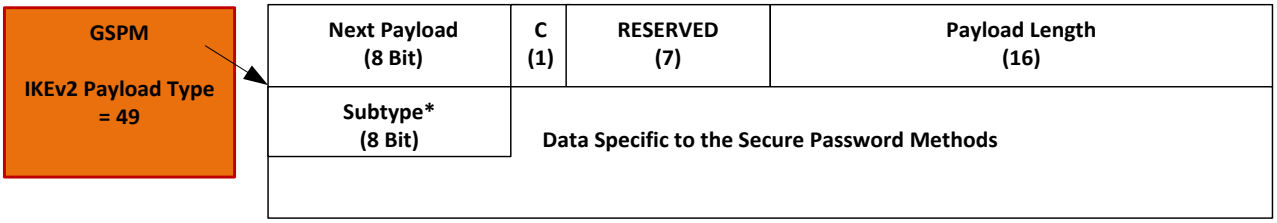

Abbildung 14: GSPM Payload

Der Subtype (in Abbildung 14) ist ein reservierter 8 Bit Wert, welcher aber optional ist.

Die im Payload enthaltenen Daten sind alle in den einzelnen Secure Password Methods spezifiziert. Der Inhalt der GSPM-Payloads sowie die Anzahl und Abfolge der IKE_AUTH Runden werden im Folgenden nur für PACE behandelt.

2.8 Password Authenticated Connection Establishment (PACE)

Password Authenticated Connection Establishment (PACE) ist ein Sicherheitsprokoll, das eine authentisierte und verschlüsselte Verbindung zwischen zwei Peers auf der Basis von kryptografisch schwachen Passwörtern ermöglicht und trotzdem einen starken gemeinsamen Schlüssel liefert. Die kryptografische Sicherheit von PACE wurde in *Security Analysis of the PACE Key-Agreement Protocol* [5] untersucht und bestätigt. Das Verfahren wurde in *Supplemental Access Control forMachine Readable Travel Documents* [6] als einen Standard für das drahtlose Auslesen von digitalen Reisedokumenten aufgenommen. In RFC6631 wird die Implementierung des Verfahrens für IKEv2 spezifiziert, wonach sich unsere Implementierung richtet.
Stark vereinfacht lässt sich der Kern des Verfahrens im Motto zusammenfassen: „Wenn der Schlüssel schwach ist, dann mach den Klartext stark“. So gesehen kehrt er die Eigenschaften von Klartext und Passwort um. Geht man davon aus, dass ein Angreifer den Verkehr zwischen zwei Peers mitschneidet, so ist er in der Lage, ein schwaches Passwort in absehbarer Zeit zu erraten. Dazu wendet er eine Liste der möglichen Passwörter bis zu einer bestimmten Länge zum Entschlüsseln der übertragenen Daten an. Sobald der entschlüsselte Klartext ein Muster erkennen lässt, das in der relativen Häufigkeit und Teilfolgen von Zeichen dem erwarteten Klartext entspricht, ist er auf das richtige Passwort gestossen. Indem PACE mit dem Passwort nur Klartext verschlüsselt und überträgt, der stark zufälligen Charakter hat, wird dem Angreifer das Kriterium zur Entscheidung genommen, wann er auf das richtige Passwort gestossen ist. Der mit dem richtigen Passwort entschlüsselte Klartext unterscheidet sich in seiner Zufälligkeit nicht von denjenigen, die mit falschen Passwörtern entschlüsselt wurden.

Damit ist PACE in der Lage, den grössten Nachteil des PSK-Verfahren zur Authentisierung zu beheben: Seine potentielle Anfälligkeit gegenüber Wörterbuch-Attacken. Der IKEv2-Standard RFC5996 [1] empfiehlt in Kapitel 2.15 ausdrücklich, beim PSK-Verfahren zur Authentisierung keine vom Anwender gewählten Passwörter zu verwenden, sondern eine starke Entropiequelle zur Generierung des PSK einzubeziehen. PACE hingegen lässt ein schwaches Anwender-Passwort zu.

2.8.1 Zero-Knowledge Proof

PACE gehört zu den Zero-Knowledge Proof Verfahren. Vereinfacht zeichnen sich diese dadurch aus, dass zwei Parteien die Kenntnis eines Geheimnisses beweisen können, ohne offenzulegen, worin das Geheimnis besteht. Im Fall von PACE besteht das Geheimnis im Passwort PACESharedSecret.

2.8.2 Funktionsweise: Überblick

Beide Peers verfügen zu Beginn über das potentiell schwache Passwort **Pwd**. Um dieses nicht über längere Zeit in dieser Form zu verwenden, wird es in zwei Schritten zur Form **KPwd**

umgewandelt. Zunächst wird **SPwd** erstellt, indem die prf gefüttert mit dem ASCII-String „IKE with PACE“ als Key und dem **Pwd** als Seed.

$$SPwd = prf(\text{"IKE with PACE"}, Pwd)$$

Formel 5: SPwd Erzeugung

Dieses wird in einem zweiten Schritt als Seed einer weiteren prf zugeführt. Als Key dient die Konkatenation der beiden Nonces aus IKE_SA_INIT. Damit fliessen ins generierte **KPwd** zusätzliche – zufällige – Daten ein, über die beide Peers nach Abschluss des zugehörigen IKE_SA_INIT verfügen. Da im nachträglichen Schritt der Verschlüsselung ein Algorithmus mit fixer Key-Länge zum Einsatz kommen kann, wird dafür prf+ verwendet.

$$KPwd = prf + (Ni \mid Nr, SPwd)$$

Formel 6: KPwd Erzeugung

Der Kern von PACE liegt im nächsten Schritt. Der Initiator wählt eine zufällige Nonce **s** mit starker Entropie und verschlüsselt diese mit dem **KPwd** zur encrypted Nonce ENONCE. Die Sicherheit des Verfahrens beruht auf der Zufälligkeit der Nonce **s**. Es muss zu deren Generierung also eine Quelle mit starker Entropie gewählt werden.

$$ENONCE = E(KPwd, s)$$

Formel 7: Nonce s Verschlüsselung

Die ENONCE wird zum Responder übertragen, der sie mit Hilfe des seinerseits generierten Passworts **KPwd** entschlüsselt:

$$s = D(KPwd, ENONCE)$$

Formel 8: Nonce s Entschlüsselung

Zu beiden Seiten wird nun die Nonce **s** verwendet, um einen Generator GE für eine neue Diffie-Hellman Gruppe zu bilden. Der Generator G bezeichnet dabei den Generator der Diffie-Hellman Gruppe der vorherigen SA, also diejenige des bereits erfolgten IKE_SA_INIT Austausches. SASharedSecret bezeichnet den gemeinsamen Schlüssel der vorherigen SA.

Beide Peers verfügen in diesem Schritt also bereits über diese Daten. Initiator und Responder berechnen beide einen neuen Diffie-Hellman Generator GE. Im Fall von MODP Diffie-Hellman berechnet er sich wie folgt:

$$GE = g^s * SASharedSecret$$

Formel 9: GE Erzeugung, Nonce Mapping

Mit dem neuen Generator GE erzeugen nun beide Parteien ein neues Shared Secret im Diffie-Hellman Verfahren. Im Fall von modularem Diffie-Hellman (MODP) wählt der Initiator zufällig einen Secret Key SKEi und berechnet daraus den Public Key PKEi:

$$SKEi, PKEi = GE^{SKEi}$$

Formel 10: Neuer DH-Key Austausch Initiator

Der Responder berechnet analog:

$$SKEr, PKEr = GE^{SKEr}$$

Formel 11: Neuer DH-Key Austausch Responder

Jede Seite überträgt den Public Key zur Gegenseite, womit das neue Diffie-Hellman Shared Secret berechnet wird. Dieses wird als PACESharedSecret bezeichnet.

$$PACESharedSecret = PKEi^{SKEr} = PKEr^{SKEi}$$

Formel 12: Neuer DH-Key PACESharedSecret

Mit dem PACESharedSecret erfolgt die tatsächliche Authentisierung beider Seiten, das heisst die Generierung der AUTH Payload. Dies erfolgt in zwei Runden wie in Kapitel 2.8.4 beschrieben.

Verwenden die beiden Peers Elliptic-Curve Diffie-Hellman (EC-DH), so wird der neue Generator berechnet mit. Die Operationen * und + bezeichnen dabei die Gruppenoperationen der elliptischen Kurve.

$$GE = s * g + SASharedSecret$$

2.8.3 IKE_SA_INIT mit PACE

Zwei Peers verwenden die in Kapitel 2.7.2 beschriebene Erweiterung des Secure Password Method Frameworks, um sich auf PACE für die Authentisierung zu einigen. In der NOTIFY Payload steht der Wert **1** für die Wahl von PACE. Die in den SA Payloads ausgehandelten Krypto-Suiten dienen als Grundlage für das PACE Protokoll. Das heisst: Die Wahl von EC-DH oder MODP, der Generator G und der zu verwendende Verschlüsselungsalgorithmus steht fest.

2.8.4 IKE_AUTH: Runden und Payloads mit PACE

Die erste Runde im IKE_AUTH dient der Übertragung der vom Initiator generierten Nonce **s** an den Responder und der Aushandlung des neuen Diffie-Hellman Shared Secret. Die verschlüsselte Nonce **s** wird im Payload Type **49** (GSPM) übertragen. Mit Erzeugen der Nonce **s** ist der Initiator bereits in der Lage, die neue Diffie-Hellman Gruppe zu bilden, wählt einen Secret Key **SKEi2** und berechnet den zugehörigen Public Key **KEi**. Die Payload **KEi2** enthält diesen Public Key (Abbildung 15). Alle anderen Payloads werden gleich gebildet wie im üblichen IKE_AUTH Austausch (s. Kapitel 2.3.2).

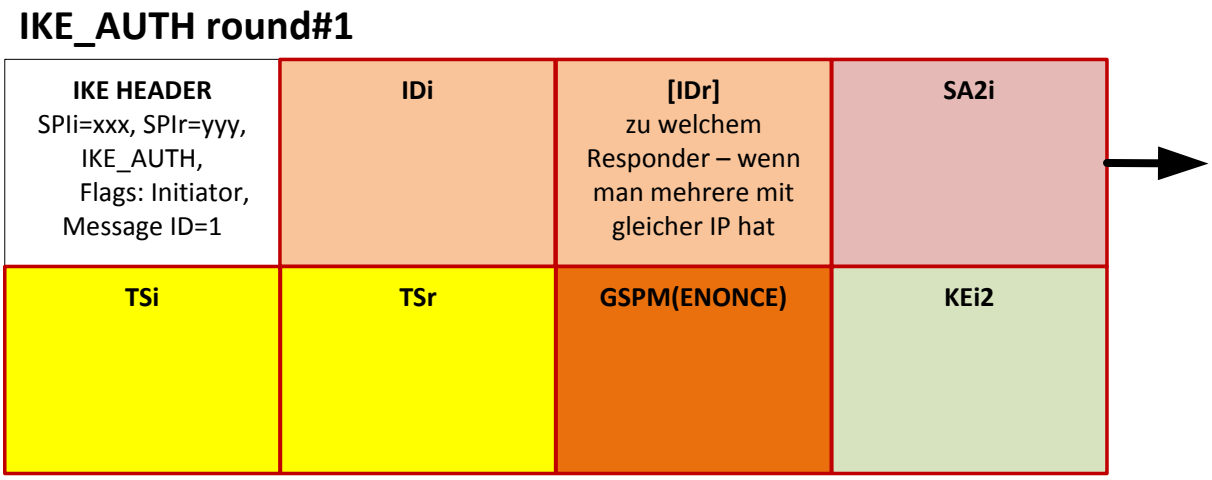

Abbildung 15: IKE_AUTH PACE round#1 Initiator Payloads

Nach Entschlüsselung der Nonce **s** berechnet der Responder den neuen Generator, wählt seinen Secret Key **SKEr2** und überträgt den daraus gebildeten Public Key in der Payload **KEr2** (Abbildung 16) an den Initiator.

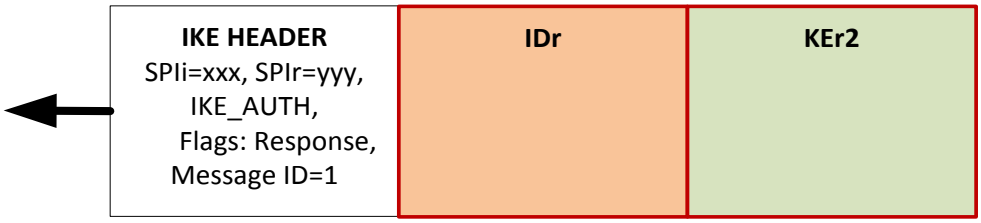

Abbildung 16: IKE_AUTH PACE round#1 Responder Payloads

In der zweiten Runde wird die gegenseitige Authentisierung durchgeführt und die Vorbereitung der CHILD_SA abgeschlossen. Nach Berechnung des PACESharedSecret sind beide Peers in der Lage, die AUTH Payload zu bilden. Jeder Peer überprüft die AUTH Payload der Gegenseite. Optional und in dieser Studienarbeit nicht realisiert, signalisiert der Initiator seine Absicht, ein kryptografisch starkes Passwort (Long-Term-Secret) zu generieren und damit das bestehende schwache initiale Passwort zu ersetzen.

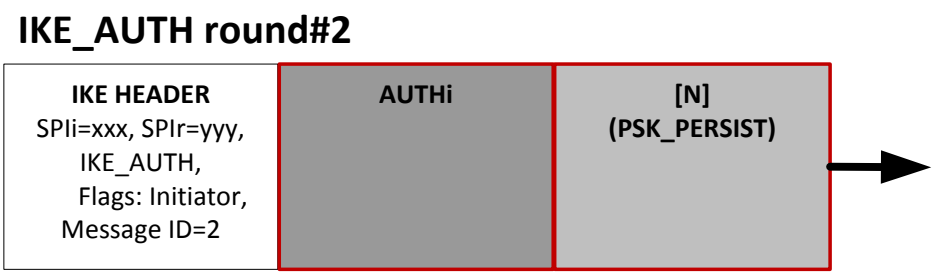

Abbildung 17: IKE_AUTH PACE round#2 Initiator Payloads

Der Responder übermittelt zusätzlich zu seiner **AUTHr** Payload erst nach Überprüfung der **AUTHi** Payload des Initiators – also in der zweiten IKE_AUTH Runde – die Subnetze, über die der Tunnel aufgebaut werden soll (**TSi,TSr**) sowie seine Auswahl aus den Krypto-Suiten, die vom Initiator in IKE_AUTH Runde 1 vorgeschlagen wurden (Abbildung 18).

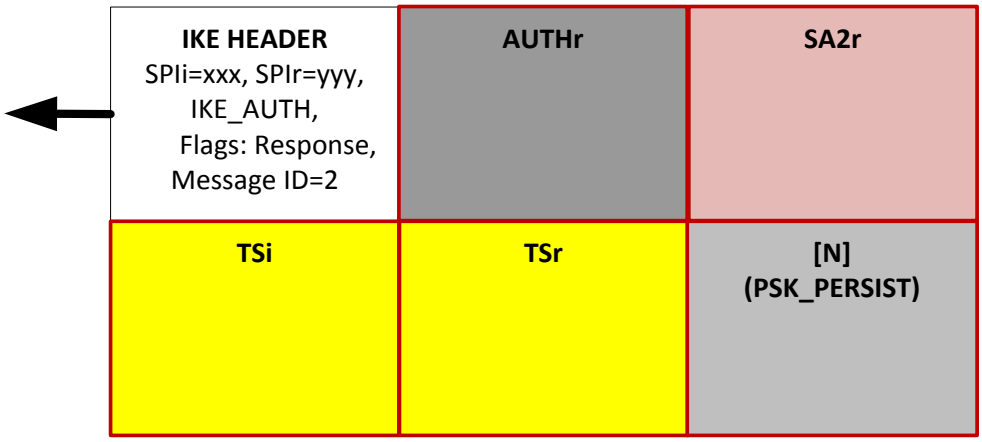

Abbildung 18: IKE_AUTH PACE round#2 Responder Payloads

Mit dem Austausch dieser Payloads und nach Überprüfung der **AUTHr** Payload durch den Initiator sind die Voraussetzungen erfüllt für den Aufbau der ersten CHILD_SA (hier nicht behandelt).
Die optionalen NOTIFY Payloads in Abbildung 17 und Abbildung 18 werden im Kapitel 2.8.7 behandelt.

2.8.5 AUTH Payload

In einem ersten Schritt wird der Key für die Pseudo Random Function (prf) selbst mit einer prf erzeugt. Deren Key bildet die Konkatenation der beiden Nonces aus dem IKE_SA_INIT, der Seed ist das PACESharedSecret:

$$AUTHKEY = (prf(Ni|\ Nr, PACESharedSecret)$$

Formel 13: AUTH Payload Key Erzeugung

Mit diesem Key wir nun eine zweite prf aufgerufen, als Seed dient die Konkatenation von SignedOctets und dem Public Key der Gegenseite. Die AUTH des Initiators ist demnach:

$$AUTHi = prf(AUTHKEY, < InitiatorSignedOctets > | PKEr)$$

Formel 14: AUTHi signed octets Erzeugung

Und die des Responders:

$$AUTHr = prf(AUTHKEY, < ResponderSignedOctets > | PKEi)$$

Formel 15: AUTHr signed octets Erzeugung

Zusammengenommen ist die AUTH Payload ist also wie folgt gebildet:

$$AUTHi = prf(prf + (Ni|\ Nr, PACESharedSecret), < InitiatorSignedOctets > | PKEr)$$

Formel 16: AUTHi Payload Erzeugung

Respektive:

$$AUTHr = prf(prf + (Ni|\ Nr, PACESharedSecret), < InitiatorSignedOctets > | PKEi)$$

Formel 17: AUTHr Payload Erzeugung

2.8.6 Use Case

Für die Authentisierung eines Clients gegenüber einem Server oder Gateway sind asymmetrische Authentisierungsverfahren wie EAP von Vorteil. Der Client-Peer kennt das Passwort selbst nicht, das vom User eingegeben wird. Auch der Gateway hat keine Kenntnis des Passworts und überlässt die Authentisierung einem AAA(Authentication Authorization Accounting) Server.
Der Nutzen für PACE wie andere Zero-Knowledge-Password Verfahren liegt eher in der Authentisierung zweier Peers wie Router untereinander [2]. Das Verfahren ist symmetrisch: Beide Peers müssen im Besitz des gleichen Passworts sein und bei der Authentisierung sind keine weiteren Server beteiligt. Zudem kann die Verbindung von beiden Peers initiiert werden, jeder kann die Rolle des Initiators und Responders einnehmen.
Dass ein schwaches Passwort möglich ist, kann bei dessen Übertragung zwischen den beiden Administratoren des Routers über einen anderen Kanal helfen (z.B. telefonisch oder verschlüsselte Mail). So würde man die Konfiguration eines Site-to-Site Tunnels per eMail übertragen und den Pre-Shared-Key z.B. per Telefon. Lange komplexe Passwörter telefonisch zu übertragen ist mühsam.

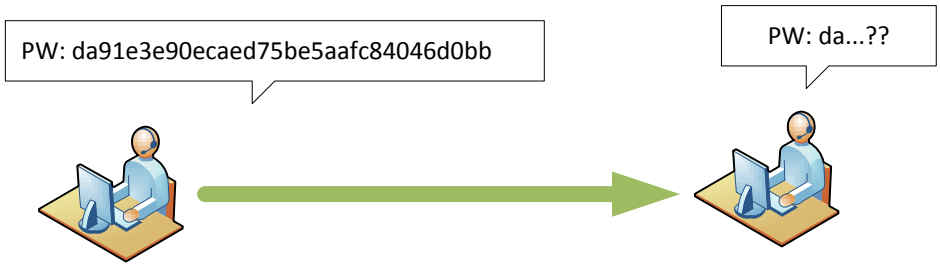

Abbildung 19: PSK Übertragung per Telefon

Kurze verständliche Passwörter für eine Site-to-Site Verbindung dagegen, sind verständlicher

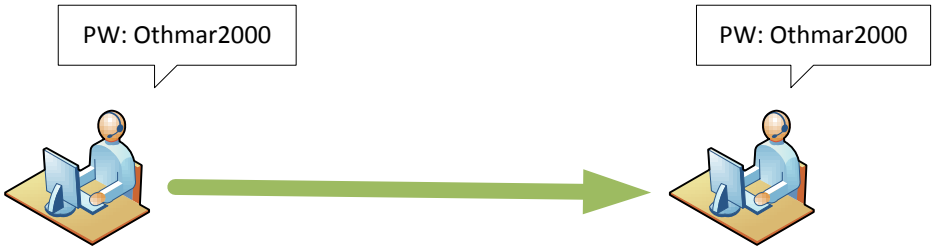

Abbildung 20: PACE Übertragung per Telefon

2.8.7 Long-Term Secret

Da es sich um ein schwaches Passwort handelt, kann es bedenklich sein, dieses als Klartext im Filesystem abzuspeichern und auszulesen. Um dem entgegen zu wirken, wurde in RFC6631 ein optionaler Mechanismus zum Abspeichern eines Shared Secret definiert, ein Long-Term Secret (LTS).

In unserer Implementierung wird diese noch nicht berücksichtigt. Das Passwort wird unter `/etc/ipsec.secrets` im Klartext definiert und verwendet. Das Auslesen verlangt jedoch root-Rechte. Unterstützt ein Peer die Verwendung eines LTS, muss dies über eine zusätzliche NOTIFY Payload in der 2. Runde im IKE_AUTH signalisiert werden. Der Initiator sendet dazu ein

PSK_PERSIST (Notify Type 16425) mit seiner letzten AUTH Message. Falls der Responder auch ein LTS unterstützt, berechnet dieser ein kryptografisch starkes LTS und sendet seinerseits ein PSK_PERSIST. Nach Berechnung das Long-Term Secret in gleicher Weise durch den Initiator schickt dieser zur Bestätigung ein PSK_CONFIRM (Notify Type 16426) an den Responder zurück. Dieser löscht dann sein schwaches Passwort und schickt seinerseits ein PSK_CONFIRM als Bestätigung, worauf der Initiator gleichauf sein schwaches Passwort löscht.

Die PSK_PERSIST und PSK_CONFIRM NOTIFY Payloads enthalten keine Daten.
Das Long-Term Secret wird mit einem erneuten Aufruf einer prf über dem PACESharedSecret mit der Konkatenation der Nonces aus dem IKE_SA_INIT als Key gebildet.

$$LongTermSecret = prf(Ni|Nr, \text{"PACE Generated PSK"}| PACESharedSecret)$$

Formel 18: LongTermSecret Erzeugung

Das **LongTermSecret** ersetzt das schwache initiale Passwort **Pwd**. Für jeden folgenden Verbindungsaufbau wird nur noch das berechnete Long-Term Secret zum Ver- und Entschlüsseln der Nonce **s** verwendet. Die Speicherung des LTS kann dabei in einem persistenten Storage oder wie in unserem Falle in einer Textdatei `/etc/ipsec.secrets` erfolgen.

Die Verwendung von LTS birgt noch ein paar Unklarheiten. So könnte einer der Kommunikationspartner Probleme haben, falls die Verbindung neu konfiguriert werden muss, z.B. wegen eines Disk Crashs oder anderem. Muss diese neu eingerichtet werden, steht kein LTS zur Verfügung. Die Verbindung muss dann auf beiden Seiten erneut eingerichtet werden mit einem schwachen initialen Passwort.
Den Use Case eines LTS sehen wir dabei auch viel eher bei einer Site-to-Site Verbindung, bei der es keine grossen Umstände macht, eine Verbindung auf beiden Seiten erneut aufzusetzen.

Mit LTS ist der gemeinsame Schlüssel sicher besser geschützt weil stärker. Sollte aber jemand auf einem System Zugriff erlangen um dies auszulesen, stimmt sowieso etwas nicht. Dafür spricht eher wieder der Use Case eines Roadwarriors, wobei Benutzer das gleiche Passwort oft auch in anderen Applikationen einsetzen und es deshalb schützenswert ist.
Bei einem Konfigurationsverlust müsste aber auch hier auf Client- und Serverseite interagiert werden, damit wieder eine Verbindung mit einem initialen Passwort möglich ist.

3 Umsetzungskonzept

3.1 Entwicklungsumgebung

Für unsere Arbeit wurde Eclipse als uns gut bekannte integrierte Entwicklungsumgebung gewählt. Damit beide in einer gleichen Umgebung arbeiten, wurden vorläufig virtuelle Maschinen in der Virtualisierungslösung VirtualBox erstellt auf Basis von einem Debian/Ubuntu abstammenden Betriebssystem namens Linux Mint mit Linux Kernel 3.2.0.
Das Arbeiten in virtuellen Maschinen erleichterte uns zudem das Aufbauen einer Testumgebung.

3.1.1 Source-Code Management

Für die Weiterentwicklung an strongSwan haben wir einen Fork des strongSwans Repository unter GitHub erstellt unter einem zusätzlichen Branch für die PACE Implementierung. Dieser kann nach Abschluss der Studienarbeit nach einem Merge mit dem aktualisierten Master-Branch ins strongSwan Repository einfliessen. Während unserer Implementierung wurde im Master-Repository von strongSwan weiter entwickelt. Die dort erfolgten Änderungen hatten aber keinen Einfluss auf unsere Arbeit, weshalb wir auf einen Merge unseres Forks mit dem Master-Respository verzichteten.

3.2 Testumgebung

Um Fortschritte zu kontrollieren und zu testen, wird in VirtualBox ein virtuelles Netzwerk aufgesetzt mit weiteren virtuellen Clients, welche mit unserer Entwicklungsumgebung in einem virtuellen Subnetz stehen.
Der Quellcode wird über ein Script synchronisiert und auf allen Maschinen kompiliert, um den aktuellen Stand der Implementierung zu testen.

3.3 Issue Tracking / Ticketing

Die Erstellung und Aufteilung von Arbeitspaketen wurden im Ticketing-System von Redmine realisiert. Dies ermöglicht uns einen aktuellen und präzisen Überblick über den Stand des Projekt und der Projektplanung, auch die Arbeitszeiten können direkt den Arbeitspaketen zugeordnet werden. Probleme und Fehler werden durch zusätzliche Tickets aufgeführt und zugewiesen.

3.4 Programmierrichtlinien

Die Programmierrichtlinien für strongSwan waren durch die Diplomarbeit von Jan Hutter und Martin Willi [7] vorgegeben. Eigene Programmierrichtlinien haben wir nicht erstellt. Wir verweisen deshalb auf diese Quelle bzw. auf die Developer Documentation auf der strongSwan Website [8].

3.4.1 Programming Style

Es wurden von unserer Seite die Programming Style Guidelines [8] von strongSwan eingehalten, die ähnlich dem BSD/Allman Style definiert sind. Die integrierte Entwicklungsumgebung Eclipse stellt dabei bei der korrekten Quelltextformatierung eine grosse Hilfe dar.

3.4.2 Object oriented C

Bei der IKEv2 Implementierung von strongSwan wurde bewusst ein objektorientierter Ansatz gewählt [7], der Objekte mit Hilfe von structs realisiert. Eine Möglichkeit objektorientierte Ansätze in C zu verwirklichen, beschreibt der „Xine's Hackers Guide" [9]. Wir halten uns an diese Vorgaben und werden Klassen, Interfaces und Makros mit diesem objektorientierten Ansatz in C weiter verwenden.

4 Design / Implementation

4.1 Architektur im Überblick

Durch die Änderungen von GSPM in der IKE_SA sind einige wenige Anpassungen an strongSwan für die Implementation von PACE nötig. So wird ein neuer Payload Typ eingeführt, ein zusätzlicher Authentisierungstyp für GSPM, ein Manager für verschiedene GSPM Strategien und ein zusätzliches Plugin. Die verschiedenen Strategien, welche über den Manager als authenticator gewählt werden, werden ähnlich den bereits implementierten Verfahren von EAP als einzelne Plugins realisiert. Da in EAP bereits mehrere Runden in IKE_AUTH vorkommen, sind bereits ein paar Grundlagen vorhanden, die auch für GSPM relevant sind.
Zusätzlich sind noch Ergänzungen für die Konfiguration nötig, damit GSPM als neues Authentisierungsverfahren ausgewählt werden kann.

In Abbildung 21 versuchen wir Klassen und Komponenten darzustellen, welche für GSPM und PACE in strongSwan neu enthalten sind.
Die einzelnen Komponenten werden dann unter Kapitel 4.2 detaillierter betrachtet.

Im bestehenden Package `sa` werden alle Funktionalitäten zusammengefasst, welche sich um die Verwaltung der Security Association (SA) kümmern [7]. Es enthält u.a. die Klasse `authenticator_t` als Schnittstelle, mit welcher die Authentisierung des Kommunikationspartners durchgeführt werden kann. Die Klasse bietet dabei je eine Funktion zur Erzeugung (`builder`) und Überprüfung (`verifier`) der Signatur / des MAC. Mit GSPM realisieren wir eine weitere Authentisierung mit der Klasse `gspm_authenticator_t`. Alle von GSPM benötigten Klassen zur Authentisierung werden im Package `sa` unter einem weiteren Package `gspm` zusammengefasst (zur Leserlichkeit in der Abbildung 21 nicht dargestellt).

Für GSPM wird im bestehenden Package `payloads` unter `encoding` ein neuer Payload Typ definiert. Die verschiedenen secure password Authentisierungsverfahren sollen als Plugin im Package `plugins` unter `gspm_pace` realisiert werden. In dieser Arbeit wird dies nur PACE sein, weitere Verfahren wie AugPAKE (RFC6628) und Secure PSK Authentication (RFC6617) werden hier als Plugins in Zukunft ihren Platz finden (unter weiteren Packages wie z.B. `gspm_augpake`).

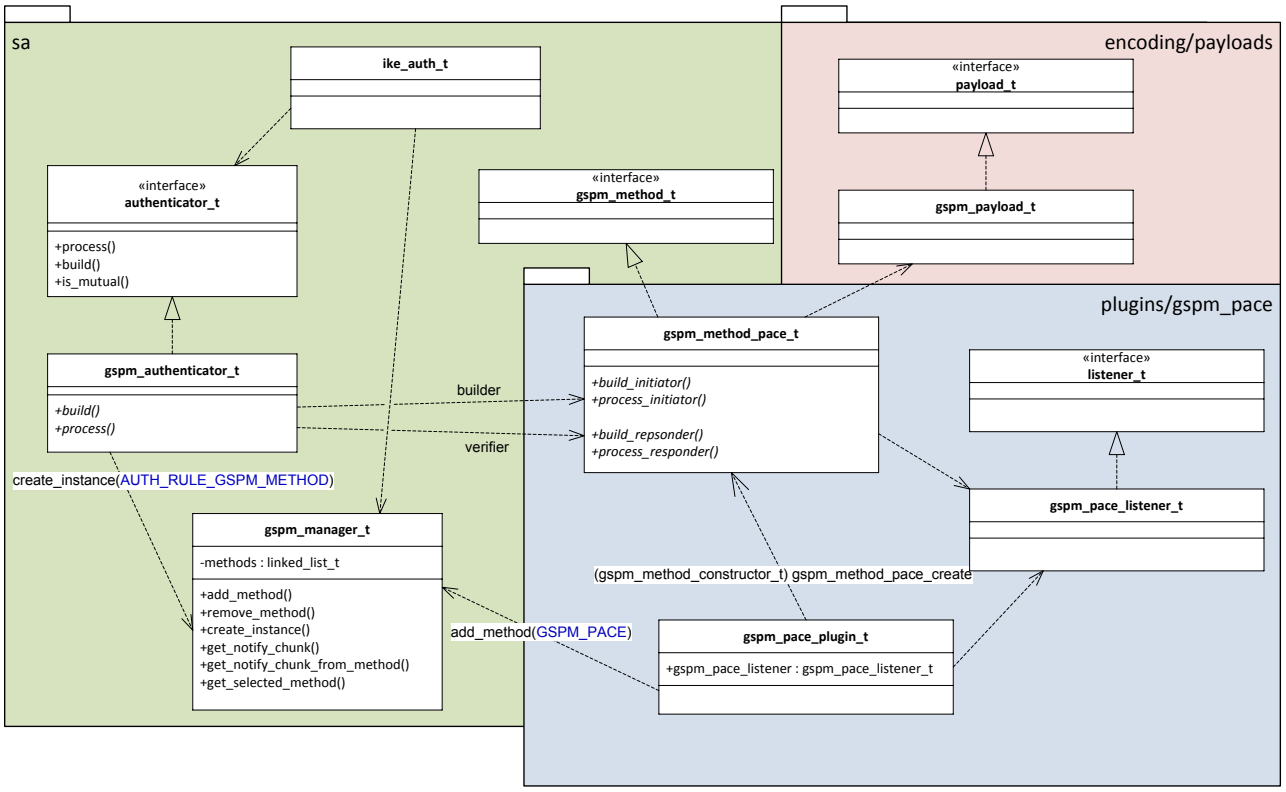

Abbildung 21: Komponenten von GSPM in strongSwan

Da das globale `daemon_t` Objekt `charon` den gesamten Daemon repräsentiert, welcher die Manager, Threads etc. erstellt, wird unter `charon` eine Instanz vom GSPM Manager veröffentlicht. Damit ist ein gemeinsamer Kommunikationsaustausch zwischen den Packages `sa` und `plugins` möglich, denn verschiedene Secure Password Implementierungen müssen sich als Plugin beim GSPM Manager registrieren. Dazu weitere Details unter 4.2.4 GSPM Manager.

Speziell für PACE ist, dass der in IKE_SA_INIT ausgetauschte Diffie-Hellmann Schlüssel weiterhin benötigt wird. Dieser wird normalerweise bei erfolgreicher IKE_SA_INIT abgeräumt. Wir realisieren den Zugriff auf den DH-Schlüssel durch einen Listener, der das Objekt abfängt, sobald dieses ausgetauscht wird.

4.2 Teilsysteme

4.2.1 Config Handler

Da es sich mit GSPM um eine neue Art der Authentisierung in strongSwan handelt, soll diese über die Konfiguration auswählbar sein. Diese wird vom `starter` im `/etc/ipsec.conf` als Connection `conn` geparsed und dem `daemon` zur Verfügung gestellt [8].
Das Argument für die gewählte Authentisierung wird mit `authby` bzw. `leftauth` und `rightauth` angegeben.

Weil PACE eine Methode der Secure Password Methods [4] ist, wird GSPM selbst als Name für die Authentifizierungsart gewählt. Einzelne Secure Password Methods werden über das Aktivieren von Plugins gesteuert, damit der Anwender immer noch eine eindeutige GSPM-Methode auswählen kann.

Im `args.c` wird deshalb ein weiteres Argument definiert namens `gspm`.

```
static const char *LST_authby[] = {
      "psk",
      "secret",
      "pubkey",
      "rsa",
      "rsasig",
      "ecdsa",
      "ecdsasig",
      "xauthpsk",
      "xauthrsasig",
      "never",
      "gspm",
       NULL
};
```

Listing 1: args.c Argumenterweiterung für authby

Während dem Start von strongSwan werden die gefundenen Connections `conn` im `ipsec.conf` aufbereitet durch `starterstroke.c`
Das Argument für `authby` (bzw. auch `leftauth`, `rightauth`) wird geparsed und der Connection hinzugefügt, wie hier ergänzt:

```
else if (streq(conn->authby, "gspm"))
{
      msg.add_conn.me.auth = push_string(&msg, "gspm");
      msg.add_conn.other.auth = push_string(&msg, "gspm");
}
```

Listing 2: starterstroke.c Ergänzung Optionen geparsed

Bei der Aufbereitung der Connection-Configuration durch `stroke` werden dann die passenden Authentication Classes zugeteilt, welche für die Laufzeit der konfigurierten Connection gelten. Diese wird im `stroke_config.c` während der Generierung der `auth_config` für das gefundene `authby` Argument zugewiesen.

```
else if (streq(auth, "gspm"))
{
	cfg->add(cfg, AUTH_RULE_AUTH_CLASS, AUTH_CLASS_GSPM);
}
```

Listing 3: stroker_config.c Zuweisung der Authentisierungsoption

Diese Authentication Class wird im nächsten Abschnitt detaillierter behandelt.

4.2.2 Authentication Class

Während des Authentisierungsprozesses verwendet strongSwan `authentication configs`, welche den einzelnen Peers einer Connection zugeordnet werden. Bei der Aufbereitung der gewählten Authentisierung in `stroke` wird der `config` die gewählte `authentication class` zugeordnet. Diese Klassen werden im `auth_class_t` unter `auth_cfg.h` definiert und auch für GSPM wird eine weitere fortlaufende Klasse hinzugefügt mit der Bezeichnung `AUTH_CLASS_GSPM`

```
enum auth_class_t {
	/** any class acceptable */
	AUTH_CLASS_ANY = 0,
…
	/** authentication using secure password method */
	AUTH_CLASS_GSPM = 5,
};
```

Listing 4: auth_cfg.h authentication class AUTH_CLASS_GSPM

Diese wird mit der Bezeichnung `secure password` als gewählte Authentisierung aufgeführt.

```
ENUM(auth_class_names, AUTH_CLASS_ANY, AUTH_CLASS_GSPM,
	"any",
	"public key",
…
	"secure password",
);
```

Listing 5: auth_cfg.c authentication name für Authentisierung mit GSPM

Während des Verbindungsaufbaus im IKE_SA_INIT muss der zugewiesenen `authentication config` die explizit ausgewählte GSPM Methode beider Parteien hinzugefügt werden, damit für die Connection die richtigen GSPM Methoden im IKE_AUTH verwendet werden.

Der `authentication config` werden in strongSwan deshalb weitere Rules hinzugefügt um über abgestimmte Parameter oder Verhalten zu verfügen welche in dieser Authentisierungsrunde benötigt oder ausgehandelt werden.

Diese sind im `auth_rules_t` definiert und für die ausgewählte GSPM Methode wird eine weitere Rule AUTH_RULE_GSPM_METHOD angelegt

```
enum auth_rule_t {
	/**identity to use for IKEv2 authentication exchange, identification_t* */
	AUTH_RULE_IDENTITY,
	/**GSPM member to propose peer authentication, u_int16_t */
	AUTH_RULE_GSPM_METHOD,
…
}
```

Listing 6: auth_cfg.h Erweiterung authentication rule

Diese Rule beinhaltet einen unsigned integer, der Werte von 0 – 65535 annehmen kann für alle verfügbaren Secure Password Methoden [4]. Zum Zeitpunkt der Studienarbeit waren bisher lediglich 3 Methoden definiert.

4.2.3 Authenticator

Um die IKE_AUTH Messages zu generieren und verifizieren, werden Authenticators verwendet. Die je nach Authentisierung unterschiedlichen Authenticators implementieren das Interface `authenticator_t`, welche dann beim Erstellen oder Verifizieren der IKE_AUTH Messages von Responder und Initator im `ike_auth_t` verwendet werden.

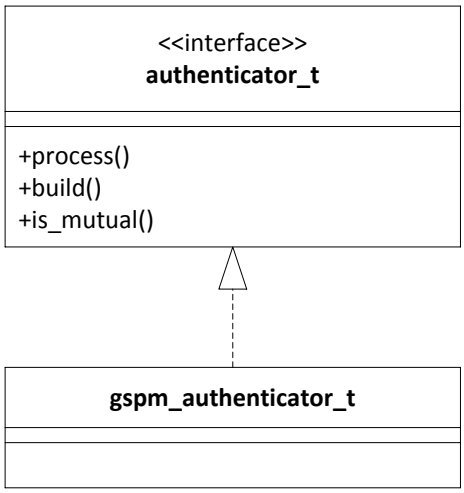

Abbildung 22: authenticator_t Implementierung für GSPM Authenticator

Bei der Erstellung eines `authenticator_builder` und `authenticator_verifier` werden die aus den `auth_cfg` gesetzten Authentisierungs-Klassen abgefragt und dementsprechend die Konstruktoren der zugehörigen `authenticators` aufgerufen. Eine Ergänzung für den `gspm_authenticator_t` wurde implementiert.

Hierbei wurden in der `authenticator.h` auch die IKEv2 Authentication Method Types gemäss RFC6467 ergänzt, welche dann die AUTH payload kennzeichnen. Für die Generic Secure Password Authentication Method ist dies die Nummer 12 [2]

```
enum auth_method_t {
…
	/**
	 * Generic Secure Password Authentication Method as specified in RFC 6467
	 */
	AUTH_GSPM = 12,
…
}
```

Listing 7: authenticator.h Eintrag des Authentication Method Types für GSPM

Wurde AUTH_CLASS_GSPM in der `config` gesetzt, wird in `authenticator.c` der entsprechende Konstruktor von `gspm_authenticator_t` für den `authenticator_builder` aufgerufen.

```
authenticator_t *authenticator_create_builder(ike_sa_t *ike_sa, auth_cfg_t *cfg,
	chunk_t received_nonce, chunk_t sent_nonce,
chunk_t received_init, chunk_t sent_init, char reserved[3])
{
	switch ((uintptr_t)cfg->get(cfg, AUTH_RULE_AUTH_CLASS))
	{
…
		case AUTH_CLASS_GSPM:
			return
(authenticator_t*)gspm_authenticator_create_builder(ike_sa, received_nonce,
sent_nonce, received_init, sent_init, reserved);
…
	}
}
```

Listing 8: authenticator.c Erzeugung des gspm_authenticator für Builder

Da ein `authenticator_verifier` zur Verifizierung von AUTH Payloads auf der Empfängerseite erstellt wird und hierbei verschiedene oder noch unbekannte Authentisierungsmethoden zum Einsatz kommen können, muss anhand des Typs der Authentication Method `Auth Method` der entsprechende Verifier erstellt werden. Beim Verfahren von PACE wird jedoch – wie bei EAP – in der ersten Runde kein AUTH Payload mitgegeben und es kann keine Zuweisung auf Grund des Typs durchgeführt werden.

Bei den bekannten Authentisierungsverfahren in strongSwan wird dies nun folgendermassen überprüft:

- Kommt keine AUTH Payload im IKE_AUTH vor, so kann das Verfahren nur EAP oder GSPM sein.
- Alle bisher implementierten GSPM Verfahren benötigen eine GSPM Payload in der ersten IKE_AUTH Runde.
- Ist also keine AUTH Payload vorhanden, wohl aber eine GSPM Payload in der empfangenen Message, so wird ein `gspm_authenticator` gewählt.
- Wird hierbei keine GSPM Payload in der empfangenen Message gefunden, wird zur Verifizierung ein `eap_authenticator` gewählt.

Die Implementierung im `authenticator.c` prüft diese Bedingungen und erstellt dann durch den entsprechenden Konstruktor den `authenticator_verifier`.

```
authenticator_t *authenticator_create_verifier(…)
{
auth_payload_t *auth_payload;
gspm_payload_t *gspm_payload;
auth_payload = (auth_payload_t*)message->
get_payload(message, AUTHENTICATION);
gspm_payload = (gspm_payload_t*)message->
get_payload(message, GENERIC_SECURE_PASSWORD_METHOD);

if (auth_payload == NULL)
{
      if (gspm_payload)
      {
            return (authenticator_t*)gspm_authenticator_create_verifier(…);
      }
      else
      {
            return (authenticator_t*)eap_authenticator_create_verifier(…);
      }
}
      switch (auth_payload->get_auth_method(auth_payload))
      {
…
      }
}
```

Listing 9: authenticator.c Erzeugung des gspm_authenticator für Verifier

Der `gspm_authenticator_t` weist dann im Zusammenspiel mit dem GSPM Manager (siehe nächster Abschnitt) den `build` und `process` Aktionen des `authenticators` die ausgewählte GSPM Methode zu, indem die `build()` und `process()` Funktionen der Plugins aufgerufen werden. Die Plugins werden im Vorfeld geladen und im GSPM Manager registriert.
Damit wird die eigentliche Strategy[3] des verwendeten Verfahrens aufgerufen.

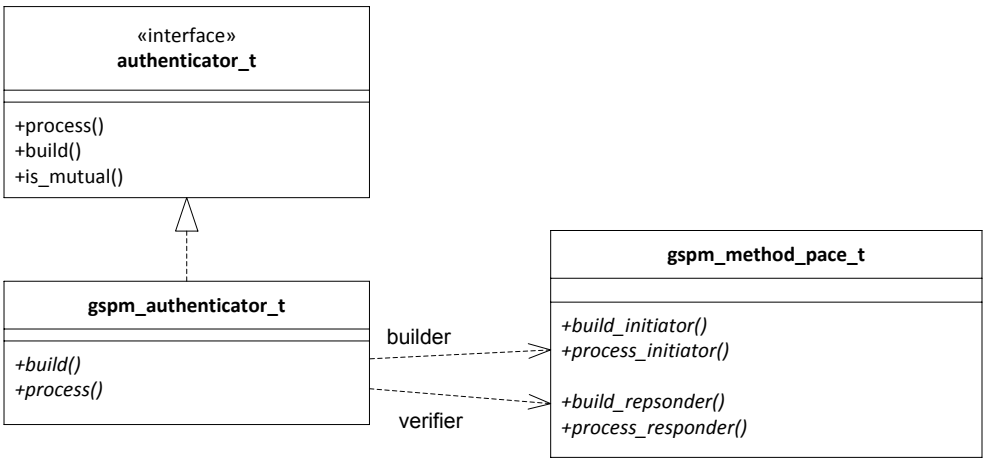

Abbildung 23: gspm_authenticator_t führt Strategy der ausgewählten gspm_method_t aus

[3] Strategy als Realisierung des GoF Strategy-Pattern [11]

Im folgenden Listing wird die entsprechende `build_initiator` Funktion der selektierten GSPM Methode aufgerufen.

```
METHOD(authenticator_t, build_initiator, status_t,
            private_gspm_authenticator_t *this, message_t *message)
{
      auth_cfg_t *auth;
      auth = this->ike_sa->get_auth_cfg(this->ike_sa, TRUE);
      if(!this->initiator_method)
      {
            this->gspm_method_selected = (uintptr_t) auth->
                        get(auth, AUTH_RULE_GSPM_METHOD);
            this->initiator_method = charon->gspm->create_instance(
                        charon->gspm, this->gspm_method_selected, FALSE,
                        this->ike_sa, this->received_nonce, this->sent_nonce,
                        this->received_init, this->sent_init, this->reserved);
      }
      if(!this->initiator_method)
      {
…
            return FAILED;
      }
      return this->initiator_method->build(this->initiator_method, message);
}
```

Listing 10: gspm_authenticator.c Aufruf der Strategy für build() des Initiators

4.2.4 GSPM Manager

Damit verschiedene Secure Password Verfahren für die GSPM Implementierung als Plugins verwendet werden können, müssen die Plugins erst einmal bekannt gemacht werden, damit sie über die verschiedenen Libraries von `libcharon` und `plugins` benutzbar sind. Dazu soll jedes für GSPM entwickelte Verfahren als Plugin ein gemeinsames Interface verwenden, sich bei einem Manager registrieren und schlussendlich als konkrete Strategy im GSPM-Authenticator verwendet werden. Das Interface definiert die für Verifier und Builder relevanten Funktionalitäten `build()` und `process()`. Die in Abbildung 24 aufgeführte Klasse `gspm_method_XY_t` stellt hier einen Platzhalter für weitere zu realisierende GSPM Methoden in strongSwan dar.

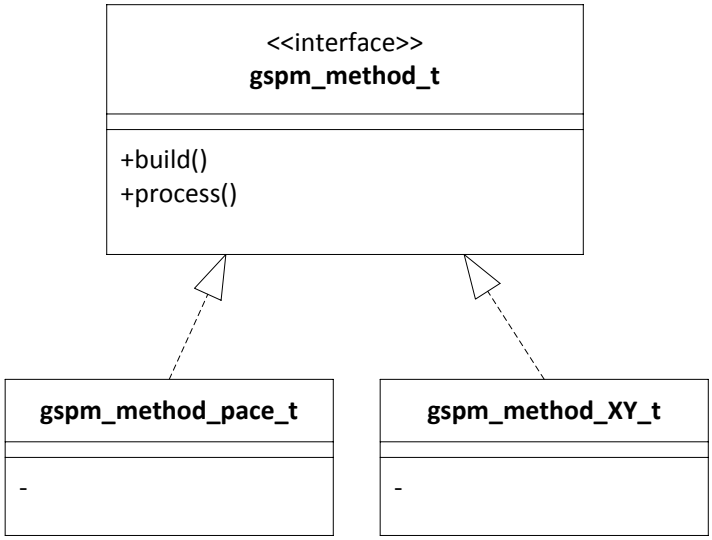

Abbildung 24: gspm_method_t Interface für verschiedene GSPM Methoden

Jedes Secure Password Verfahren realisiert als Plugin das Interface `gspm_method_t` und registriert sich beim Start von strongSwan über den GSPM-Manager `gspm_manager_t`, indem es einen Funktionszeiger zum eigenen Konstruktor im GSPM-Manager einträgt.
Im GSPM-Manager wird eine Liste über die registrierten Funktionszeiger geführt. Über diese Liste werden einerseits die geladenen GSPM Methoden ermittelt und zudem wird für den GSPM-Authenticator ein Funktionszeiger zur registrierten GSPM Methode aufbewahrt, damit die im IKE_SA_INIT ausgewählte Methode schlussendlich erstellt und ausgeführt werden kann.

Abbildung 25: Der GSPM-Manager stellt Funktionen zur Verfügung zum Registrieren der Funktionszeiger der geladenen Plugins über `add_method()`, zum Austragen (z.B. wenn Plugin verworfen wird) über `remove_method()`, und um Abruf der Funktionszeiger zu der verwendeten GSPM-Methode über `create_instance()`.

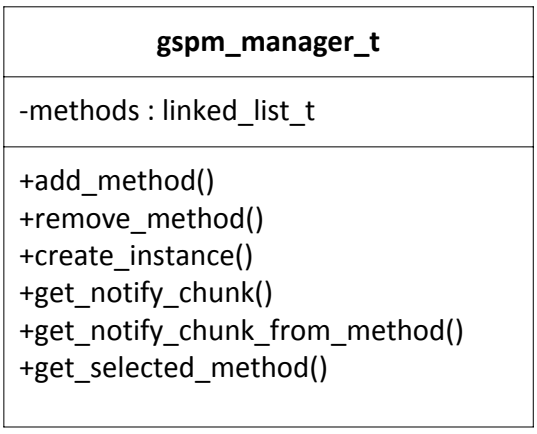

Abbildung 25: gspm_mamanger_t Implementierung

Der GSPM-Manager verfügt dadurch über eine Liste mit allen registrierten GSPM-Methoden und kann dadurch weitere Funktionalitäten als Information Expert ausüben. So wird mit `get_notify_chunk()` die im IKE_SA_INIT benötigte NOTIFY Payload mit allen vom Initiator verfügbaren GSPM-Methoden erstellt und mit `get_selected_method()` die bevorzugte GSPM-Methode beim Responder ausgewählt. Im RFC6467 wird kein Auswahlverfahren umschrieben,

falls mehrere der angebotenen zu Verfügung stehen. Deshalb lesen wir für die Selektierung die erste Secure Password Method im NOTIFY Payload aus und gleichen sie mit allen registrierten GSPM-Methoden ab. Bei einem Treffer wird diese erste Methode selektiert, ansonsten wird mit dem Abgleich bei der nächsten Methode fortgefahren.
Der Kontrollfluss im IKE_SA von strongSwan wird gesteuert durch Statusangaben.
Der Status hierbei wird im `status_s` von strongSwan angegeben, wobei diese folgende Bedeutung haben:

Status	Bedeutung
FAILED	Ein Aufruf ist fehlgeschlagen (z.B. Verifizierung fehlgeschlagen oder ein Problem ist aufgetreten)
NEED_MORE	Ein weiterer Aufruf wird benötigt (z.B. weitere IKE_AUTH Runden)
SUCCESS	Ein Aufruf war erfolgreich (z.B. Verifizierung)

Wenn gar kein Treffer erfolgt, wird per FAILED Status abgebrochen. Im Listing 11 wird gezeigt, wie der Vergleich der 16-Bit Werte in der NOTIFY Payload mit den registrierten Methoden in der `linked_list_t methods` realisiert ist.

```
notify_payload = message->get_notify(message, SECURE_PASSWORD_METHOD);
data = notify_payload->get_notification_data(notify_payload);
for(len = 0; len < data.len; len += 2)
{
	data.ptr += len;
	method = ntohs(*(u_int16_t*) data.ptr);
…
	enumerator = this->methods->create_enumerator(this->methods);
	while (enumerator->enumerate(enumerator, &entry))
	{
		if(method == entry->method_id)
		{
			DBG1(DBG_IKE, "%N method %N selected", auth_class_names,
				AUTH_CLASS_GSPM, gspm_methodlist_names, method);
			return method;
…
```

Listing 11: gspm_manager_t Methodenauswahl im IKE_SA_INIT

Über einen ENUM werden die verschiedenen GSPM-Methoden definiert. Diese werden für die Registrierung der Plugins benötigt, um diese eindeutig zu identifizieren. Die zugewiesenen `u_int16_t` Values werden für die Generierung der NOTIFY Payload verwendet mit den dafür vorgesehen IKEv2 Secure Password Method Values [4].

```
#gspm_manager.h

enum gspm_methodlist_t {
	GSPM_PACE = 1,
	GSPM_AUGPAKE = 2,
	GSPM_SPSKA = 3,
};

#gspm_manager.c

ENUM(gspm_methodlist_names, GSPM_PACE, GSPM_SPSKA,
	"PACE",
```

```
        "AugPAKE",
        "Secure PSK Authentication",
);
```

Listing 12: gspm_manager_t Definierung der verschiedenen GSPM-Methoden

4.2.5 GSPM Payload

Für die Secure Password Authentisierung wird ein neue Payload definiert [2] zur Übermittlung von Content, der spezifisch ist für einzelnen Verfahren.

Im Kapitel 2.7.1 wurden die einzelnen Messages und deren Payloads betrachtet.
Hier die detaillierte Ansicht des GSPM Payloads.

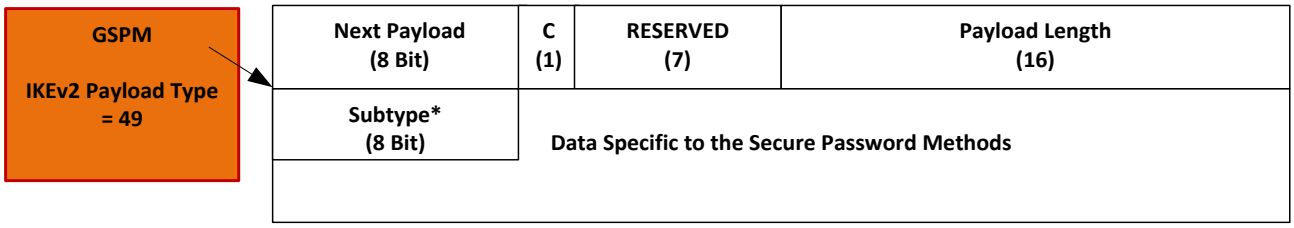

Abbildung 26: GSPM Payload gemäss RFC6467

Der GSPM Payload Type wurde gemäss RFC6467 mit der Nummer 49 den IKEv2 Payload Types definiert und bei Internet Assigned Numbers Authority IANA registriert [4].
Teile von Payloads werden im strongSwan Code als chunk bezeichnet und erstellt
Der 8-Bit chunk[4] Subtype ist optional und kann von den einzelnen Verfahren verwendet werden. Der restliche Data chunk ist genauso verfahrensspezifisch und kann von der Grösse her variieren, angegeben in der Payload Length. In strongSwan sind Payload Typen bereits definiert und lassen sich vom Interface `payload_t` ableiten. Wir implementieren eine neue `payload_t` namens `gspm_payload_t` für den GSPM Payload:

[4] chunk ist hierbei ein Fragment von Informationen eines beliebigen Typs

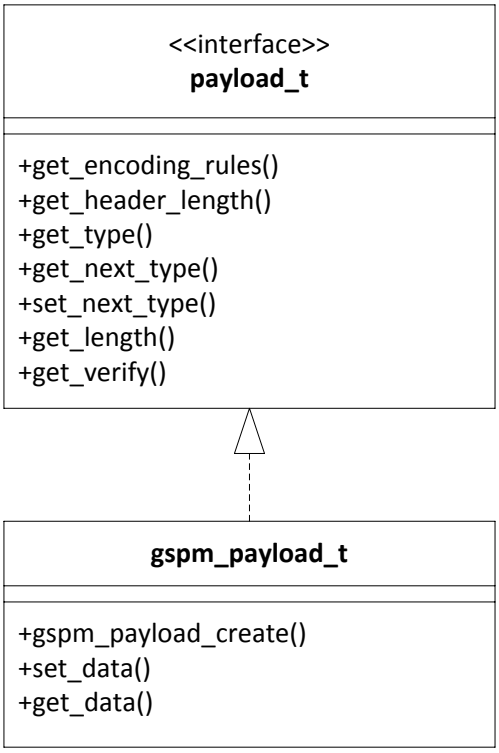

Abbildung 27: Payload Interface und Implementierung

`gspm_payload.c` enthält dann einen Subtype und Data chunk, der von den verschiedenen Verfahren einzeln beschrieben und ausgelesen werden muss, weitere chunks oder Grössen werden im RFC für das Secure Password Framework [2] selbst nicht definiert.

In strongSwan werden für das korrekte Parsen der Payloads `encoding rules` definiert, welche für `gspm_payload_t` wie nachfolgend definiert sind:

```
static encoding_rule_t encodings[] = {
      /* 1 Byte next payload type, stored in the field next_payload */
      { U_INT_8,              offsetof(private_gspm_payload_t, next_payload) },
      /* the critical bit */
      { FLAG,                       offsetof(private_gspm_payload_t, critical)
            },
      /* 7 Bit reserved bits, nowhere stored */
      { RESERVED_BIT,         offsetof(private_gspm_payload_t, reserved[0])  },
      { RESERVED_BIT,         offsetof(private_gspm_payload_t, reserved[1])  },
      { RESERVED_BIT,         offsetof(private_gspm_payload_t, reserved[2])  },
      { RESERVED_BIT,         offsetof(private_gspm_payload_t, reserved[3])  },
      { RESERVED_BIT,         offsetof(private_gspm_payload_t, reserved[4])  },
      { RESERVED_BIT,         offsetof(private_gspm_payload_t, reserved[5])  },
      { RESERVED_BIT,         offsetof(private_gspm_payload_t, reserved[6])  },
      /* Length of the whole payload*/
      { PAYLOAD_LENGTH, offsetof(private_gspm_payload_t, payload_length)},
      /* chunt to data, starting at "code" */
      { CHUNK_DATA,           offsetof(private_gspm_payload_t, gspm_data)
      },
};
```

Listing 13: gspm_payload_t encoding rules

Zusätzlich zu den `encoding rules` sind für den Parser im strongSwan unter `message_t` weitere Regeln beschrieben, welche die Payloads in der aktuell behandelten Message umschreibt, wann diese vorkommen können und dürfen (`payload_order_t`), wie oft ein Payload erscheinen kann (`payload_rules_t`) und in welcher Rund bzw. in welchem Status des IKE_SA.

Hierbei wurde Einträge für GSPM und den neu eingeführten NOTIFY Payloads benötigt.
Eine Definition unsererseits war, dass GSPM Payloads im IKE_AUTH maximal 3-Mal vorkommen dürfen. In RFC6631 wird erwähnt, dass kaum eine Secure Password Method mehr als 3 Runden mit GSPM Paylods benötigen wird, und bei den aktuell verfügbaren Verfahren ist dies auch nicht der Fall. Hierbei ist für `MAX_GSPM_PAYLOADS` der Wert 3 definiert.

```
static payload_rule_t ike_auth_r_rules[] = {
/*    payload type                              min   max
            encr  suff */
      {NOTIFY,                                  0,    MAX_NOTIFY_PAYLOADS,
      TRUE, TRUE},
      {KEY_EXCHANGE,                            0,    1,
            TRUE, FALSE},
      {GENERIC_SECURE_PASSWORD_METHOD,0, MAX_GSPM_PAYLOADS,          TRUE,
      TRUE},
…
```

Listing 14: message_t payload_rule_t für IKE_AUTH Messages

Zudem wird GSPM als neuer Payload Typ in der Klasse `payload_t` eingetragen unter `payload_type_t`

```
enum payload_type_t {
…
      /**
       * Generic Secure Password Method (GSPM).
       */
      GENERIC_SECURE_PASSWORD_METHOD = 49,
...
```

Listing 15: payload_t Payload Type Definition in payload_type_t

4.2.6 Listener

Eine wichtige Erweiterung in strongSwan für die GSPM Methode PACE ist der Plugin-Listener. Da seither das Shared Secret vom Diffie-Hellman Austausch in IKEv2 nach dem IKE_SA_INIT nicht mehr benötigt wird, wurde das Diffie-Hellman Objekt und das darin enthaltene Shared Secret (bzw. **SKEr** und **SKEi**) gelöscht. Für PACE wird das Shared Secret aus dem IKE_SA_INIT aber noch im IKE_AUTH benötigt.

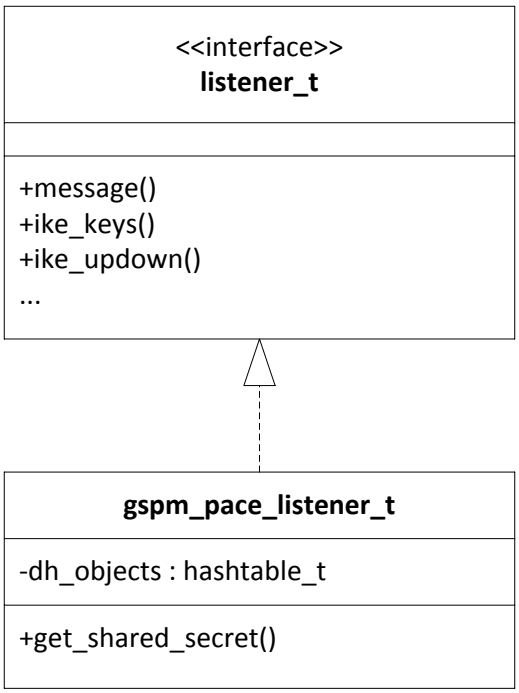

Abbildung 28: gspm_pace_listener_t Listener für Shared Secret aus IKE_SA_INIT

Der Listener wird beim Laden des `gspm_pace_plugin_t` Plugins erstellt und für die Verwendung in `gspm_pace_t` veröffentlicht.
Auf dem Bus des IKEv2 keying daemons `charon` werden Signale von verschiedenen Threads empfangen. Für unseren Listener sind die Objekte `message` und `ike_keys` von Interesse, sowie Änderungen des Verbindungsstatus. Für das spätere Ableiten des neuen DH-Generators brauchen wir Zugriff auf das DH-Objekt des IKE_SA_INIT. Damit das richtige Diffie-Hellman Objekt und das darin enthaltene Shared Secret einer Verbindung aufgenommen werden kann, muss der Listener über eine eindeutige Verbindungsidentifizierung verfügen. Die „IKE SPIs“ connection identifier sind laut RFC5996 eindeutige Identifier für eine IKE_SA [1]. Über einen message hook werden `message_t` Objekte abgefangen. Es gilt nun zu überprüfen, ob diese `message_t` Bestandteil eines mit GSPM verwendeten Verbindungaufbaus angehört. Der `exchange_type_t` des Message Objekt wird auf IKE_SA_INIT überprüft sowie auf dessen NOTIFY Typ, der SECURE_PASSWORD_METHOD sein muss (Value 16424 gemäss RFC6467 [2]).
Da momentan nur die GSPM-Methode PACE ein Shared Secret Objekt im IKE_AUTH benötigt, wird die NOTIFY Payload auf das Vorhandensein von PACE überprüft. Ist dies der Fall, wird ein Hash des nun überprüften IKE SPIs mit einem Entry in die Hashtable des `gspm_pace_listener_t` eingetragen.

Für den Responder muss zum Zeitpunkt, wenn die erste Message im IKE_SA_INIT mit NOTIFY Payload SECURE_PASSWORD_METHOD eintrifft, das Shared Secret Objekt bereits gespeichert werden, da es sonst bei der darauffolgenden Antwort vom Responder wieder entfernt wird. Da aber beim Eintreffen der ersten IKE_SA_INIT Message noch nicht klar ist, ob PACE als GSPM-Methode ausgewählt wird, wird in unserer Implementierung diese IKE SPI bereits hier mit einer Entry in die Hashtable eingetragen. Sollte dann für die gleiche IKE SPI die IKE_SA_INIT Antwort keine GSPM-Methode PACE im NOTIFY beinhalten, wird der Entry wieder gelöscht. Beim Initiator kann die ausgewählte GSPM-Methode einfacher überprüft werden, da die eingehende IKE_SA_INIT Antwort des Responders die eine gewählte Methode beinhaltet.

```
if (incoming && message->get_exchange_type(message) == IKE_SA_INIT)
{
	notify_payload = message->get_notify(message,
		SECURE_PASSWORD_METHOD);
	if(notify_payload)
	{
		dh_entry = malloc_thing(dh_entry_t);
		dh_entry->ike_sa_id = id->clone(id);
		this->dh_objects->put(this->dh_objects, id,
			dh_entry);
	}
}
if (!incoming && message->get_exchange_type(message) == IKE_SA_INIT)
{
	notify_payload = message->get_notify(message,
		SECURE_PASSWORD_METHOD);
	if(notify_payload)
	{
		method = ntohs(*(u_int16_t*) notify_payload->
				get_notification_data(notify_payload).ptr);
		if(method != GSPM_PACE)
		{
			this->dh_objects->remove(this->dh_objects, id);
		}
	}
}
```

Listing 16: gspm_listenet_t Abfangen des Shared Secret Objekt in IKE_SA_INIT

Gleich darauf folgend mit dem Eintreffen der Message im IKE_SA_INIT wird die Diffie-Hellman Berechnung ausgeführt und ein `ike_keys` über den Bus signalisiert. Der Listener wird dann über den `ike_keys` Hook der korrespondierende IKE SPI mit der Hashfunktion auf einen vorhandenen Eintrag in der Hashtable überprüfen. Bei einem Treffer wird der vorhandene Entry mit dem Shared Secret Objekt ergänzt. Sollte die Verbindung getrennt werden, wird über das signalisierte `ike_updown` der Verbindungabbau abgefragt und der korrespondierende Entry wieder aus der Hashtable ausgetragen.

Über die Funktion `get_shared_secret()` kann dann das zum IKE SPI korrespondierende Shared Secret Objekt im `gspm_method_pace_t` benutzt werden für die Erzeugung des Diffie-Hellman Generators GE.

4.2.7 PACE Plugin

Jede GSPM-Methode wird als strongSwan Plugin implementiert und steht damit beim Start von strongSwan bereits zur Verfügung. Für unsere Zwecke wird das Plugin vor allem für die Erzeugung eines Listeners genutzt, der während des Startups die im Absatz 4.2.6 aufgeführten Funktionalitäten realisieren kann. Das Plugin wird unter dem Namen `gspm-pace` veröffentlicht.

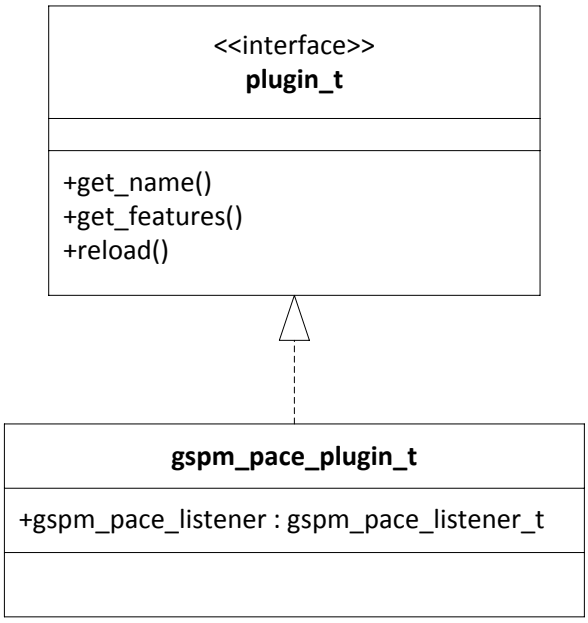

Abbildung 29: gspm_pace_plugin_t Implementierung

Bei der Erzeugung des Plugins wird ein `gspm_pace_listener_t` Objekt erzeugt, auf dem Bus registriert und für `gspm_method_pace_t` zur Verfügung gestellt.
Das Plugin registriert sich dann beim GSPM Manager als Methode PACE und übergibt einen Funktionszeiger zum Konstruktor von `gspm_method_pace_t`

```
gspm_pace_listener = gspm_pace_listener_create(),
charon->bus->add_listener(charon->bus, &gspm_pace_listener->listener);
charon->gspm->add_method(charon->gspm, GSPM_PACE,
            (gspm_method_constructor_t) gspm_method_pace_create);
```

Listing 17: gspm_pace_plugin_t Konstruktor für Listener und GSPM Methode

Das Plugin kann in strongSwan nun mit der Autoconf Options `--enable-gspm-pace` geladen werden.

4.2.8 IKE_SA_INIT und IKE_AUTH Anpassungen

Für die GSPM-Method PACE sind gegenüber den normalen IKEv2 Abläufen ein paar Ergänzungen notwendig:

- Zusätzliche NOTIFY Payload in IKE_SA_INIT
- Neue GSPM Payload in IKE_AUTH
- Zusätzliche Authentifizierungsrunde in IKE_AUTH

In strongSwan sind gewisse EAP-Authentifizierungsverfahren mit ähnlichen Ergänzungen bereits implementiert. Für GSPM waren ein paar zusätzliche Anpassungen in `ike_auth_t` notwendig: In den `build` Funktionen wird eine zusätzliche NOTIFY Payload generiert, falls in der Authentication Config die Authentisierungsklasse GSPM AUTH_CLASS_GSPM erscheint. Hierzu wurde eine Hilfsfunktion `gspm_auth_enabled()` in `ike_auth_t` erstellt, um die Authentication Config nach dieser Klasse abzufragen. Bei einer positiven Abfrage wird in der `build` Funktion des Initiators eine zusätzliche NOTIFY Payload erstellt mit allen verfügbaren GSPM-Methoden. Die Funktion `get_notify_chunk()` des GSPM Managers liefert hierbei den passenden `chunk_t`, welcher alle registrierten Methoden beinhaltet.

```
if (message->get_exchange_type(message) == IKE_SA_INIT)
{
	/** build and add GSPM notify if necessary*/
	if (gspm_auth_enabled(this))
	{
		message->add_notify(message, FALSE, SECURE_PASSWORD_METHOD,
				charon->gspm->get_notify_chunk(charon->gspm));
	}
...
```

Listing 18: ike_auth_t zusätzlicher NOTIFY Payload in build_i()

Im `process` des Responder muss dann die generierte NOTIFY Payload nach GSPM-Methoden durchforstet und eine geeignete Methode selektiert werden. Diese Funktionalität ist im GSPM Manager enthalten mit `get_selected_method()` und beschrieben unter 4.2.4. Falls die selektierte Methode 0 zurückgibt, schlägt die Verbindung fehl, da keine verwendbare Methode gefunden wurde. Die selektierte Methode wird dann in einem weiteren Schritt als Authentication Rule AUTH_RULE_GSPM_METHOD gesetzt.

```
if (message->get_exchange_type(message) == IKE_SA_INIT)
{
	/** process notify and get GSPM methods */
	if (message->get_notify(message, SECURE_PASSWORD_METHOD))
	{
		this->gspm_method_selected = charon->gspm->
			get_selected_method(charon->gspm, message, FALSE);
		if(this->gspm_method_selected == 0)
		{
			return FAILED;
		}
...
```

Listing 19: ike_auth_t NOTIFY Payload Analyse in process_r()

Der Responder wird dann im `build` nur noch mit der selektierten Methode im NOTIFY Payload antworten und der Initiator wird im `process` zur Sicherheit die gleiche Selektierung mit dem NOTIFY Payload durchführen, obwohl eigentlich klar ist, welche GSPM-Methode nun verwendet wird.

Wenn die erste Authentisierungsrunde von PACE durchgeführt wird, muss auf beiden Seiten die Authentication Config ergänzt werden für beide Peers, damit das richtige Verfahren verwendet wird um zu verifizieren. Wenn in IKE_AUTH kein AUTH Payload vorhanden ist, wurde in strongSwan bisher angenommen, dass es sich um ein EAP-basiertes Authentisierungsverfahren handelt, um dann die Peer Configs mit den nötigen Informationen aufzubauen. In `ike_auth_t` wurde deshalb noch eine weitere Fallunterscheid eingebaut, bei der die Authentication Config gleich mit der selektierten GSPM-Methode ergänzt wird:

```
if (message->get_payload(message, AUTHENTICATION) == NULL)
{
	if(this->gspm_method_selected)
	{
		cfg->add(cfg, AUTH_RULE_GSPM_METHOD, this->gspm_method_selected);
	}
	else
	{
```

Listing 20: ike_auth_t Fallunterscheidung wenn kein AUTH Payload vorhanden

4.3 PACE

Die eigentliche Implementierung der PACE spezifischen Funktionalitäten ist in `gspm_method_pace_t` realisiert. Da in den vorherigen Kapiteln alle nötigen Vortreffungen und Änderungen für Secure Password Verfahren in strongSwan erläutert wurden, können nun die Einzelheiten für die Password Authenticated Connection Establishment behandelt werden.
Um die Funktionen der `gspm_method_pace_t` Klasse im Überblick zu behalten, sind sie in Abbildung 30 aufgeführt. Die Funktionen `build()` und `process()` werden durch den Konstruktur, je nachdem ob `authenticator` ein Builder oder Verifier ist, den privaten Funktionen `build_initiator()`, `process_initiator()` oder `build_responder()`, `process_responder()` zugeordnet.

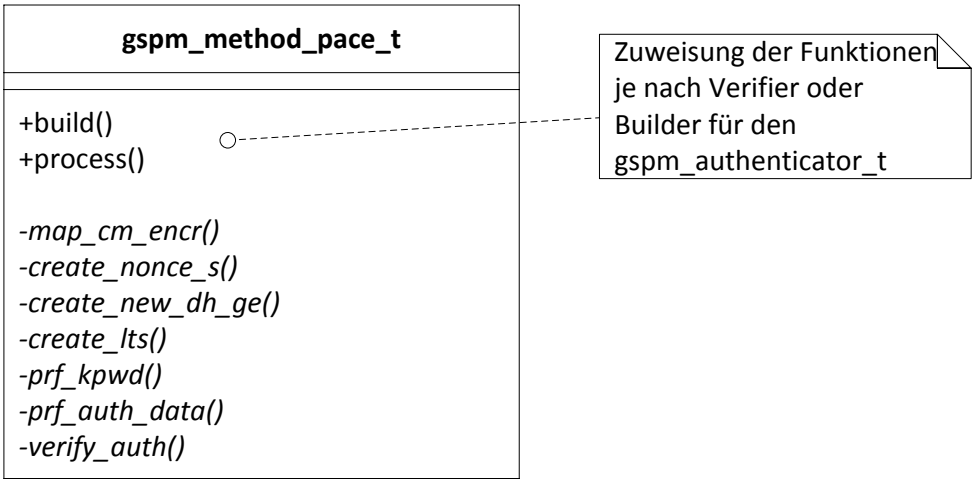

Abbildung 30: gspm_method_pace_t Implementierung

4.3.1 Nonce Generierung

Essentieller Teil des PACE Verfahren ist eine möglichst zufällig gewählte Nonce. Normalerweise wird eine WEAK Randomization für Nonces in der IKE_SA_INIT und für Initialization Vectors (IV's) gewählt. Da eine TRUE Randomization über `/dev/random` den Ablauf je nach Benutzerinteraktion eine Zeit lang blockiert - bei unseren Tests gute 3-6 Sekunden - ist diese für einen schnellen Verbindungsaufbau nur mässig geeignet. Die STRONG Randomization über `/dev/urandom` sollte aber genügend hohe Entropie liefern für die Verwendung der Nonce **s**.
Sollte die Nonce **s** schlecht gewählt worden sein, wird beim Mapping die Möglichkeit erhöht, dass der neu gewählte DH Generator = 1 wird, bzw. die Überprüfung G^s = 1/SASharedSecret fehlschlägt und die Nonce **s** erneut gewählt werden muss [3].
Realisiert wurde diese Überprüfung später beim Mapping durch eine Schleife, wobei zur Sicherheit ein Maximum an Durchläufen definiert wird mit ROUND_LIMIT. Sollte nach 3-maligem Neuerstellen der Nonce **s** die Überprüfung immer noch fehlschlagen, wird abgebrochen.

```
#define ROUND_LIMIT 3
bool create_nonce_s(private_gspm_method_pace_t *this)
{
	rng_t *rng;

	rng = lib->crypto->create_rng(lib->crypto, RNG_STRONG);
…
	if(!rng->allocate_bytes(rng, NONCE_SIZE, &this->s))
	{
		DBG2(DBG_IKE, "could not create random nonce s from RNG");
		rng->destroy(rng);
		return FALSE;
	}
…
```

Listing 21: gspm_method_pace_t Nonce Generierung mit STRONG Randomgenerator

4.3.2 Nonce Mapping

Das Mapping beschreibt das Generieren eines neuen Generators, bzw. einer Primitivwurzel g, für den Diffie-Hellman Schlüsselaustausch in Runde 1 von PACE.
Diese ist für Modulare und Elliptische Diffie-Hellman Verfahren natürlich unterschiedlich.
Da für die Berechnung elliptischer Kurven noch weitere Implementierungen erforderlich sind, damit ein neuer Custom Generator GE gebildet werden kann mithilfe der OpenSSL Elliptic Curve Library, wurde diese Funktion wegen zu grossem Aufwand in unserer Studienarbeit ausgelassen.
Die Berechnung des MODP-DH Generators lässt sich hingegen mit Big Numbers realisieren, wobei die GNU Multiple Precision Arithmetic Library (`gmp`) die nötigen Funktionalitäten liefert für die Berechnung grosser Werte. Diese wird in strongSwan bereits genutzt, und es existiert eine Funktion für die Generierung von Custom DH Objekten.

$$GE = g^s * SASharedSecret$$

Formel 19: Neuer GE bei modularem Diffie-Hellman

Die Formel 19 zum Berechnen des neuen Generators GE kann folgender massen auseinander genommen werden: $SASharedSecret = g^{ir}$
Das SASharedSecret entsteht aus den SKE von Initiator und Responder in IKE_SA_INIT.

Die verwendete Basis **g** ist bei beiden Exponenten gleich und da es sich um den gleichen Generator aus IKE_SA_INIT handelt, ist die gewählte Primzahl **p** dieser Diffie-Hellman Runde identisch, womit der resultierende Generator GE auch reduziert mit **mod** von **p** immer noch in der gleichen DH Gruppe liegt. Damit erhalten wir:

$$GE = (g^s * g^{ir}) mod\ p$$

Formel 20: Vereinfachung der GE Berechnung

Dies lässt sich weiter vereinfachen zu $g^s * g^{ir} = g^{s+ir}$
Da wir aber aus der IKE_SA_INIT nur das SharedSecret über den Listener abfangen und nicht SKEi/SKEr erhalten, wird die Multiplikation mit SASharedSecret durchgeführt. Dies erzielt auch dann noch eine kleine Performancesteigerung bei der Berechnung von GE und lässt sich einwandfrei mit `gmp` realisieren.

```
bool create_new_dh_ge(private_gspm_method_pace_t *this, bool initiator)
{
…
	mpz_import(g, dh_param->generator.len, 1, 1, 1, 0,
	dh_param->generator.ptr);
	mpz_import(p, dh_param->prime.len, 1, 1, 1, 0, dh_param->prime.ptr);
	mpz_import(sa_secret, shared_secret.len, 1, 1, 1, 0, shared_secret.ptr);
…
	if(initiator)
	{
		while(!verified)
		{
			if (nonce_round >= ROUND_LIMIT)
			{
				return FALSE;
			}
			create_nonce_s(this);
			…
			mpz_import(nonce_s, this->s.len, 1, 1, 1, 0, this->s.ptr);

			if(diffie_hellman_group_is_ec(this->dh_group))
			{
				/** TODO implementation of EC with OpenSSL*/
				return FALSE;
			}
			else
			{
				mpz_powm(ge, g, nonce_s, p);
				mpz_mul(ge, ge, sa_secret);
				mpz_mod(ge, ge, p);

				if(!mpz_cmp(ge, one) == 0)
				{
					verified = TRUE;
				}
…
```

Listing 22: gspm_method_pace_t Nonce Mapping, Berechnung von GE

4.3.2.1 Neuer Diffie-Hellman Schlüsselaustausch

Mit der neu erzeugten Basis wird ein neues Diffie-Hellman Objekt erzeugt mit Gruppe MODP_CUSTOM. Daraus lassen sich wiederum die öffentlichen Schlüssel KEi2 und KEr2 bilden für den Austausch des PACESharedSecret.

```
	new_ge =  mpz_to_chunk(ge);

	/**
	 * Creates new DH with MODP_CUSTOM and g, p
	 */
	this->dh_ge = lib->crypto->create_dh(lib->crypto, MODP_CUSTOM,
		new_ge, dh_param->prime);
```

Listing 23: gspm_method_pace_t Diffie-Hellman Generierung

4.3.3 Nonce Verschlüsselung Round#1

Falls Authenticated Encryption with Associated Data (AEAD) zur Verschlüsselung ausgehandelt wurde, muss das korrespondierende nicht-autorisierende Verschlüsselungsverfahren für die Nonce Verschlüsselung verwendet werden [3]. Im Falle von AES_GCM_ICV8 wäre dies z.B. AES_CTR mit derselben Schlüssellänge.
Ein Mapping der AEAD-basierten Verfahren wird in `map_cm_encr()` durchgeführt, indem die vorhandenen Algorithmen in strongSwan abgefragt werden.

```
void map_cm_encr(private_gspm_method_pace_t *this)
{
	switch(this->enc_algorithm)
	{
		case ENCR_AES_CCM_ICV8:
		case ENCR_AES_CCM_ICV12:
		case ENCR_AES_CCM_ICV16:
		case ENCR_AES_GCM_ICV8:
		case ENCR_AES_GCM_ICV12:
		case ENCR_AES_GCM_ICV16:
			this->enc_algorithm = ENCR_AES_CTR;
			break;
		case ENCR_CAMELLIA_CCM_ICV8:
		case ENCR_CAMELLIA_CCM_ICV12:
		case ENCR_CAMELLIA_CCM_ICV16:
			this->enc_algorithm = ENCR_CAMELLIA_CTR;
			break;
…
```

Listing 24: gspm_method_pace_t Encryption Mapping

Der Key **KPwd** für die Nonce Verschlüsselung wird durch eine prf+ errechnet, beschrieben in RFC5996, welche in strongSwan bereits implementiert ist für IKEv2.
Dazu wird das gemeinsame Passwort für PACE in ein „stored password“ **SPwd** konvertiert. Für unsere Implementation haben wir aber noch kein eigenes Secret, sondern nutzen das PSK Secret in `/etc/ipsec.secrets` für den Verbindungsaufbau. Für eine spätere Implementierung in strongSwan müsste ein zusätzliches Feld in `ipsec.secrets` definiert werden oder man führt einen neuen Speicherort ein, welcher dann auch für das Long-Term-Secret genutzt werden kann. Dies ist aber optional und wurde von uns nicht vollständig implementiert.
Das **SPwd** wird durch eine prf erzeugt mit einem Key, bestehend aus den ASCII characters „**IKE with PACE**“. In der Funktion `prf_kpwd()` von `gspm_method_pace_t` wird der `chunk_t pwd`,

welches nun das Shared Secret von PSK aus dem `/etc/ipsec.secrets` ist, mit diesem Key randomisiert und erzeugt dabei das **SPwd**.
Als Key für die Erzeugung des **KPwd** mit einer prf+ werden die aus IKE_SA_INIT verwendeten Nonces von Initiator und Responder zusammen verkettet. Mit prf+ werden mehrere Runden einer prf durchgeführt, bis das Material in gewünschter Länger vorliegt. **KPwd** soll dabei die gleiche Länge haben wie die Schlüssellänge des verwendeten Verschlüsselungsverfahrens.

```
bool prf_kpwd(private_gspm_method_pace_t *this, chunk_t pwd, chunk_t nonce_i,
      chunk_t nonce_r, chunk_t *kpwd)
{
      prf_plus_t *prfp;
      chunk_t key, nonce_key, spwd;

      key = chunk_from_str("IKE with PACE");
      nonce_key = chunk_cat("cc", nonce_i, nonce_r);

      if (!this->prf->set_key(this->prf, key ) ||
            !this->prf->allocate_bytes(this->prf, pwd, &spwd) ||
            !this->prf->set_key(this->prf, nonce_key))
      {
            return FALSE;
      }

      prfp = prf_plus_create(this->prf, TRUE, spwd);
      if(!prfp->allocate_bytes(prfp, this->enc_keysize / 8, kpwd))
      {
            return FALSE;
      }
…
```

Listing 25: gspm_method_pace_t KPwd Generierung

Wurde KPwd erst einmal erzeugt, dient dieses als Schlüssel für die Verschlüsselung des Nonce **s**. Falls ein IV für das verwendete Verschlüsselungsverfahren benötigt wird, muss dieser für die Nonce Verschlüsselung erzeugt und im GSPM Payload mitgegeben werden.

```
METHOD(gspm_method_t, build_initiator, status_t,
            private_gspm_method_pace_t *this, message_t *message)
{

      if(!(kpwd.len == crypter->get_key_size(crypter)))
      {
            return FAILED;
      }
      if(!create_new_dh_ge(this, TRUE))
…
      rng = lib->crypto->create_rng(lib->crypto, RNG_WEAK);
…
      iv.len = crypter->get_iv_size(crypter);
      if(!rng->allocate_bytes(rng, iv.len, &iv))
      {
            return FAILED;
      }
…
      if(   !crypter->set_key(crypter, kpwd) ||
            !crypter->encrypt(crypter, this->s, iv, &enonce))
      {
```

```
            return FAILED;
        }
…
        gspm_data = chunk_cat("mmm", chunk_from_thing(st), iv, enonce);
        gspm_payload = gspm_payload_create();
        gspm_payload->set_data(gspm_payload, gspm_data);
        chunk_free(&gspm_data);
        message->add_payload(message, (payload_t*)gspm_payload);

        ke_payload = ke_payload_create_from_diffie_hellman(KEY_EXCHANGE,
                    this->dh_ge);
        message->add_payload(message, (payload_t*)ke_payload);

        this->round_two = TRUE;

        return NEED_MORE;
}
```

Listing 26: gspm_method_pace_t Nonce Verschlüsselung

`gspm_data` setzt sich hierbei, wie in den Kapiteln Grundlagen und GSPM Payload erwähnt, zusammen aus einem Subtype, hier ein mit `chunk_from_thing(st)` erzeugten 8 Bit Value mit Inhalt 0, sowie aus dem erzeugten IV und der verschlüsselten Nonce **s** ENONCE.
Die **KEi2**/**KEr2** Payload ist gleichgesetzt mit der normalen **KE** Payload in IKE_SA_INIT und beinhaltet den öffentlichen Schlüssel des DH Austausches. Eine Hilfsfunktion in strongSwan ist bereits vorhanden, sie erzeugt gleich den **KE** Payload mit dem im DH-Objekt vorhandenen öffentlichen Schlüssel.

Nachdem die Message in `build_initiator()` zusammengesetzt wurde, wird mit einem boolean `round_two` im Builder signalisiert, dass die erste Runde von PACE für den Initiator abgeschlossen ist. Beim Responder wird dann die eingehende Message überprüft. Falls kein AUTH Payload vorhanden ist, deutet dies auf die erste PACE Runde hin. Die gleichen Funktionen wie beim Initiator werden in `gspm_method_pace_t` wieder verwendet für den Verifier des Responders, um u.a. die Verschlüsselungsverfahren zu mappen und das **KPwd** zu erzeugen.

Im `process_responder()` wird zuerst der Subtype aus der GSPM Payload ausgelesen. Falls dieser nicht = 0 ist, also nicht dem PACE_RESERVED gemäss RFC6631 entspricht, wird abgebrochen. Der IV und dessen Länge gemäss Verschlüsselungsverfahren wird ausgelesen sowie die restlichen Daten der Payload, welche dem ENONCE entsprechen. Mit dem gleich erzeugten **KPwd** wird das ENONCE dann entschlüsselt.

```
…
map_cm_encr(this);

crypter = lib->crypto->create_crypter(lib->crypto, this->enc_algorithm,
      this->enc_keysize / 8);
if(!crypter)
{
      return FAILED;
}
this->enc_keysize = crypter->get_key_size(crypter)*8;
iv.len = crypter->get_iv_size(crypter);

if(!prf_kpwd(this, shared_key->get_key(shared_key), this->received_nonce,
      this->sent_nonce, &kpwd))
{
      return FAILED;
}
gspm_payload = (gspm_payload_t*)message->get_payload(message,
      GENERIC_SECURE_PASSWORD_METHOD);
if(!gspm_payload )
{
      return FAILED;
}
ke_payload = (ke_payload_t*)message->get_payload(message, KEY_EXCHANGE);
if(!ke_payload)
{
      return FAILED;
}
recv_gspm_data = chunk_clone(gspm_payload->get_data(gspm_payload));
reader = bio_reader_create(recv_gspm_data);
if(!reader->read_uint8(reader, &st) ||
      !reader->read_data(reader, iv.len, &iv) ||
      !reader->read_data(reader, reader->remaining(reader), &enonce))
{
      return FAILED;
}
if(st != 0)
{
      return FAILED;
}
if(!crypter->set_key(crypter, kpwd) ||
      !crypter->decrypt(crypter, enonce, iv, &this->s))
{
      return FAILED;
}

if(!create_new_dh_ge(this, FALSE))
{
      return FAILED;
}
…
```

Listing 27: gspm_method_pace_t Nonce Entschlüsselung

Nach der Entschlüsselung der Nonce **s** kann das Diffie-Hellman Mapping wieder durchgeführt werden mit `create_new_dh_ge()` wie in 4.3.2, wobei aber das Boolean `initiator` auf FALSE gesetzt wird, damit keine neue Nonce **s** erzeugt sondern das bestehende genommen wird, und wir Code wieder verwenden können. Gleich wie beim Initiator wird im Builder dann die **KE**-Payload aus dem DH-Objekt erzeugt und der Message hinzugefügt und die `round_two` gesetzt

.

4.3.4 AUTH Generierung und Verifizierung Round#2

Aus den ausgetauschten **SKEi** und **SKEr** von Runde 1 können nun die AUTH Payloads erzeugt werden mit dem gemeinsamen PACESharedSecret. Die bestehenden DH-Objekte werden mit den öffentlichen Schlüsseln ergänzt und das SharedSecret kann berechnet werden.
Die AUTH Payloads unterscheiden sich ein wenig von den Payloads in IKEv2, die signed octets aber bleiben gleich [1]. Dazu existiert im `keymat_v2_t` von strongSwan bereits eine Hilfsfunktion `get_auth_octets()`. Bei Inititator und Responder müssen wir bei der signed octetes Erzeugung in der Funktion `prf_auth_data()` nur noch eine Unterscheidung machen, damit die richtigen Nonces aus IKE_SA_INIT genommen werden.
Um Code wieder zu verwenden, wird im `prf_auth_data()` mit einem Boolean `verify` entweder die Erzeugung oder die Verifizierung eines AUTH Payload signalisiert.

```
bool prf_auth_data(private_gspm_method_pace_t *this, chunk_t *auth_data,
      identification_t *id, chunk_t init, chunk_t nonce, bool verify)
{
…
      if(this->ike_sa->get_id(this->ike_sa)->is_initiator(this->ike_sa->
            get_id(this->ike_sa)))
      {
      nonce_seed = chunk_cat("cc", this->sent_nonce, this->received_nonce);
      }
      else
      {
      nonce_seed = chunk_cat("cc", this->received_nonce, this->sent_nonce);
      }
      if (!this->dh_ge->get_shared_secret(this->dh_ge, &pace_shared_secret) ==
SUCCESS)
      {
            return FAILED;
      }
      keymat = (keymat_v2_t*)this->ike_sa->get_keymat(this->ike_sa);
      keymat->get_auth_octets(keymat, verify, init, nonce,
            id, this->reserved, &auth_octets);
      if(!this->prf->set_key(this->prf, nonce_seed) ||
            !this->prf->allocate_bytes(this->prf, pace_shared_secret, &prf_key))
      {
            return FAILED;
      }
      if(verify)
      {
            prf_seed = chunk_cat("mc", auth_octets, this->my_pke);
      }
      else
      {
            prf_seed = chunk_cat("mc", auth_octets, this->other_pke);
      }

      if(!this->prf->set_key(this->prf, prf_key) ||
            !this->prf->allocate_bytes(this->prf, prf_seed, auth_data))
      {
…
            return FAILED;
      }
…
```

Listing 28: gspm_method_pace_t AUTH Generierung

$$AUTHir = prf(prf + (Ni|\ Nr, PACESharedSecret), < Initiator/SenderSignedOctets > | PKEri)$$

Formel 21: AUTHi/r Berechnung prf+

In RFC6631 (Stand 27.05.2013) wird bei der AUTH Erzeugung in Formel 19 immer noch ein `prf+` erwähnt, was aber einem normalen `prf` entspricht (so besprochen bei Meeting am 22.04.2013).

$$AUTHir = prf(prf(Ni|\ Nr, PACESharedSecret), < Initiator/SenderSignedOctets > | PKEri)$$

Formel 22: Eigentliche AUTHi/r Berechnung mit prf

Die signed octets werden mit den zugehörigen öffentlichen Schlüssel des DH Austauschs in Runde 1 verkettet. Als Key für die `prf` wird eine weiterere `prf` aufgerufen mit den verketteten Nonces aus IKE_SA_INIT und dem PACESharedSecret, gewonnen aus dem DH-Austausch aus Runde 1. Die damit erzeugte AUTH Payload wird in Runde 2 im `build_initiator()` bzw. `build_responder()` der Message hinzugefügt. Beim `process_initiator()` bzw. `process_responder()` wird mit der Funktion `verify_auth()` die gleiche AUTH Payload nachgebaut und mit der empfangenen AUTH Payload verglichen. Sollten die beiden Payloads nicht übereinstimmen, schlägt die Authentifizierung (die Verifizierung der Authentisierungsrunde) fehl. Hierbei wird das PACEShareSecret falsch sein, bzw. das Passwort welches der Initiator oder Responder gesetzt hat, da ansonsten alle anderen verwendeten Werte wie die Nonces aus IKE_SA_INIT identisch sein sollten. Stimmen die AUTH Payloads überein, wird auf beiden Seiten die Authentifizierung erfolgreich abgeschlossen.

```
bool verify_auth(private_gspm_method_pace_t *this, message_t *message)
{
…
	auth_payload = (auth_payload_t*)message->get_payload(message,
AUTHENTICATION);
…
	if(!prf_auth_data(this, &auth_data, this->ike_sa->get_other_id(this-
>ike_sa),
		this->received_init, this->received_nonce, TRUE))
	{
		return FALSE;
	}
	recv_auth_data = chunk_clone(auth_payload->get_data(auth_payload));
	if(!auth_data.len || !chunk_equals(auth_data, recv_auth_data))
	{
		return FALSE;
	}
…
	auth = this->ike_sa->get_auth_cfg(this->ike_sa, FALSE);
	auth->add(auth, AUTH_RULE_AUTH_CLASS, AUTH_CLASS_GSPM);
	return TRUE;
}
```

Listing 29: gspm_method_pace_t AUTH Verifizierung

4.3.5 LTS Generierung

Als Ergänzung zu 4.3.4: Wird LTS verwendet, wird der Message beim Initiator noch ein weiteres NOTIFY hinzugefügt vom Typ PSK_PERSIST ohne jeglichen Inhalt. Sollte der Responder auch LTS unterstützen, und er bekommt von Initiator ein NOTIFY vom Typ PSK_PERSIST, wird das

PACESharedSecret zu einem Long-Term-Secret umgewandelt und so abgespeichert. Der Responder sendet dann seine Message ergänzt mit einem NOTIFY vom Typ PSK_CONFIRM zurück, damit der Initiator die gleichen Schritte in die Wege leitet.
Wie in 4.3.3 erwähnt wurde das optionale Verfahren zum LTS in unserer Lösung nicht vollständig implementiert. Die Funktion `create_lts()` für die Erzeugung des LTS ist aber bereits für zukünftige Verwendung in `gspm_method_pace_t` definiert. Es braucht dazu noch eine geeignete Ablage zum Speichern des LTS und eine zusätzliche Fallunterscheidung in `gspm_method_pace_t`, ob LTS verwendet wird oder noch das gemeinsame Passwort im Klartext.

```
bool create_lts(private_gspm_method_pace_t *this, chunk_t *lts)
{
…
      if(this->ike_sa->get_id(this->ike_sa)->is_initiator(this->ike_sa->
            get_id(this->ike_sa)))
      {
            nonce_seed = chunk_cat("cc", this->sent_nonce, this-
>received_nonce);
      }
      else
      {
            nonce_seed = chunk_cat("cc", this->received_nonce, this-
>sent_nonce);
      }
      if (!this->dh_ge->get_shared_secret(this->dh_ge, &pace_shared_secret) ==
SUCCESS)
      {
            return FALSE;
      }
      pace_seed = chunk_from_str("PACE Generated PSK");
      pace_seed = chunk_cat("mm", pace_seed, pace_shared_secret);

      if(!this->prf->set_key(this->prf, nonce_seed) ||
            !this->prf->allocate_bytes(this->prf, pace_seed, lts))
      {
            return FALSE;
      }
…
```

Listing 30: gspm_method_pace_t LTS Generierung

Eine Möglichkeit für die Ablage der LTS wäre, wenn man das initiale Passwort für PACE im `/etc/ipsec.secrets` oder einem anderen Credential Storage definiert, wobei man diese Ablage nur mit root Rechten auslesen kann, und dieses Klartext Kennwort dann durch das generierte LTS im `/etc/ipsec.secrets` ersetzt.

5 Tests

Tests wurden von unserer Seite aus auf zwei virtuellen Teststationen über ein virtuelles Netzwerk durchgeführt. Dazu verwendeten wir einen Linux Mint 13 Client aus unserer Entwicklungsumgebung und einen Server, wahlweise aufgesetzt auf Debian 6 und Debian 7.

5.1 Manuelle Tests

Die virtuellen Maschinen sind stets mit dem aktuellsten Source Code ausgestattet damit Änderungen schnellstmöglich getestet werden können. Dazu wurden passende Skripts und Configs angefertigt, um die Synchronisierung des Codes, die Kompilierung, Starten und Debuggen der beiden Peers etwas zu beschleunigen.
Die virtuellen Maschinen, ein Client der die Verbindung aufbaut und wahlweise ein Debian 7 System oder ein Debian 6 (um z.B. auch eine ältere Version von libgmp zu testen) als Server, der auf eingehende Verbindungen wartet, sind über ein internes Netzwerk von Virtualbox verbunden: `intnet: 10.10.10.0/24`.
Beide Peers verfügen über ein local loopback Interface, damit die Verbindung zwischen 2 verschiedenen Subnets aufgebaut werden kann.

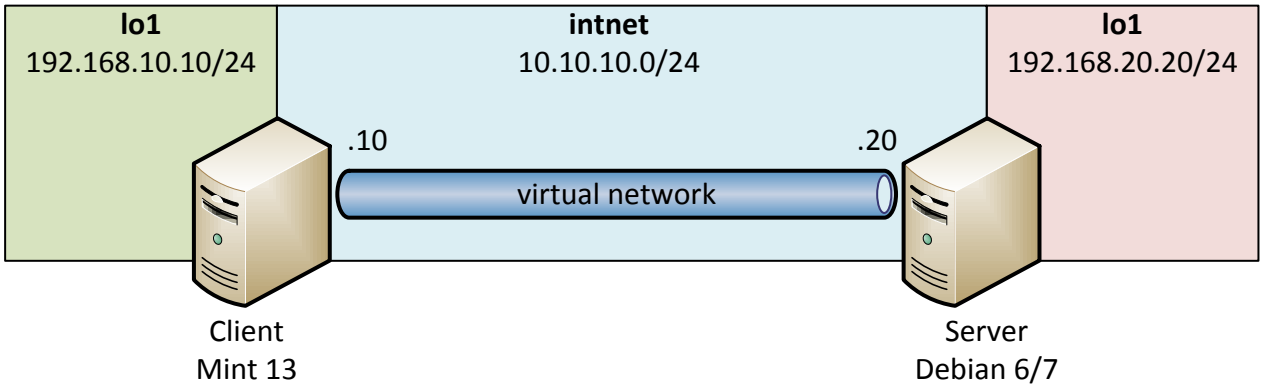

Abbildung 31: Testumgebung Client/Server über virtuelles Netzwerk

Die Client Config wurde für verschiedene Verschlüsselungsalgorithmen angepasst um z.B. das Mapping vom authentisierenden zu dem korrespondierenden nicht-authentisierenden Verschlüsselungsalgorithmus zu testen.

`/etc/ipsec.conf` Einstellungen für den Client waren z.B.

- ike=aes128ccm12-aesxcbc-modp2048!
- ike=aes128ctr-aesxcbc-modp2048!
- ike=3des-sha1-modp1536!
- ike=aes256-md5-modp2048!

5.1.1 CUSTOM MODP Test

Da in Formel 20 der neue Generator mit einem **mod** von **p** berechnet wurde, es aber möglich wäre auf die Modulo Berechnung zu verzichten, haben wir kurzerhand eine Testverbindung aufgesetzt, bei der die Clientseite keine **mod p** und die Serverseite eine **mod p** Berechnung für die Generierung von GE durchführt, wobei beide in der gleichen DH-Gruppe und Subgruppe der Primitivwurzel bleiben sollten. Der Test wurde erfolgreich durchgeführt.
Für die Berechnung mit `gmp` wurde anstatt **mod** von **p** ein **mod** von **1**, da `mpz_pow_ui` nur Werte von `unsigned long int` berechnet.

```
//mpz_powm(ge, g, nonce_s, p);
mpz_powm(ge, g, nonce_s ,one);
mpz_mul(ge, ge, sa_secret);
//mpz_mod(ge, ge, p);
```

Listing 31: gspm_pace_method_t Test mit GE Berechnung ohne mod p

5.1.2 Falsches Passwort

Ein wichtiger und einfacher Test, der nicht vergessen werden darf, ist die Authentifizierung mit unterschiedlichen Passwörtern. Diese muss natürlich fehlschlagen.
Unterscheiden sich die Passwörter der Kommunikationspartner mit denen sie sich authentisieren möchten, so schlägt die Verifizerung der AUTH Payload auf Seite des Responders fehl, weil unterschiedliche Werte vorhanden sind.
Werden unterschiedliche Passwörter verwendet, wird bei der Nonce **s** Verschlüsslung und Entschlüsselung ein unterschiedlicher Key KPwd verwendet. Unterscheiden sich die Nonces, werden beim Nonce Mapping unterschiedliche Diffie-Hellman Generatoren GE erzeugt und somit entsteht PACE Runde 2 ein unterschiedliches Shared Secret und PACESharedSecret. Dieses wird bei der Erzeugung des AUTH Payloads verwendet. Erst beim Austausch und Verifizierung der AUTH Payloads wird die Unstimmigkeit bemerkt und die ganze Authentisierung schlägt fehl.

Für einen Test dieses Verhaltens in unserer Testumgebung wurde bei einem der Peers ein anderes Passwort im `/etc/ipsec.secrets` eingetragen und ein Verbindungsaufbau durchgeführt. Unterschiedliche PACESharedSecret und AUTH Payloads wurden generiert, ein Debug der Payloads und der Abbruchmeldung ist in Listing 32 vom Initiator und Listing 33 vom Responder aufgeführt.

5.1.3 Andere Implementierungen

Uns sind zum Zeitpunkt der Fertigstellung keine anderen Implementierungen von PACE in IKEv2 bekannt. Daher war ein Test gegenüber anderen IKEv2-Systemen nicht möglich.

```
May 29 18:02:33 03[IKE] PACESharedSecret => 256 bytes @ 0x7fc70c001410
May 29 18:02:33 03[IKE]    0: 5C 91 0E 06 22 D3 9F E6 E5 3A C6 D7 12 16 BB E1  \..."....:......
May 29 18:02:33 03[IKE]   16: 68 06 35 25 E1 FE B6 E4 C0 34 EE 89 A4 0C 6E CA  h.5%.....4....n.
May 29 18:02:33 03[IKE]   32: 91 EE 0B ED CC 0E 79 02 C2 BB E9 4F 6A C6 0C 41  ......y....Oj..A
May 29 18:02:33 03[IKE]   48: 69 F4 84 2F 1A 4E A0 B5 2D A5 AE A5 FF 40 49 28  i../.N..-....@I(
May 29 18:02:33 03[IKE]   64: 49 CE E2 24 60 3F 3C AF EE 3A B9 53 57 0A 9F 58  I..$`?<..:.SW..X
May 29 18:02:33 03[IKE]   80: F4 F4 F8 D1 13 78 F7 27 5D 6F 77 F9 7A 59 6B C1  .....x.']ow.zYk.
May 29 18:02:33 03[IKE]   96: B6 FD 4A 76 5F AC C5 DC 38 D8 58 2C FB 0B 33 99  ..Jv_...8.X,..3.
May 29 18:02:33 03[IKE]  112: 84 52 E6 D5 24 8C D4 0C 20 CF 30 1C 0D 43 9A 80  .R..$... .0..C..
May 29 18:02:33 03[IKE]  128: 74 EC 18 24 6A 3B 16 50 20 D0 63 86 D5 8A D8 CB  t..$j;.P .c.....
May 29 18:02:33 03[IKE]  144: 1F 8B 6A FC E9 CF B7 C5 42 DA AB F9 3B 1E AC A4  ..j.....B...;...
May 29 18:02:33 03[IKE]  160: 85 85 FC 15 C0 41 BC 9B 43 2F BE B6 72 9C F5 C4  .....A..C/..r...
May 29 18:02:33 03[IKE]  176: 24 42 0A CC 5E 04 A5 B2 6F 63 50 E2 6A 37 A5 5B  $B..^...ocP.j7.[
May 29 18:02:33 03[IKE]  192: AD C9 45 2F 92 71 36 B9 81 0B 7A 41 B2 98 04 16  ..E/.q6...zA....
May 29 18:02:33 03[IKE]  208: E3 21 3B 36 ED DC C4 AB FB 1B BF 5C CF E2 37 4A  .!;6.......\..7J
May 29 18:02:33 03[IKE]  224: 84 1E CF 78 EC F3 33 CF D0 05 76 E2 20 D8 CB 4E  ...x..3...v. ..N
May 29 18:02:33 03[IKE]  240: D8 F6 CB 3D 84 72 B3 98 B1 60 90 2E 8A 48 20 DB  ...=.r...`...H .
May 29 18:02:33 03[IKE] AUTHi = prf(prf(Ni | Nr, PACESharedSecret),<InitiatorSignedOctets> | PK
Er) => 20 bytes @ 0x7fc70c000a10
May 29 18:02:33 03[IKE]    0: 95 84 B5 50 52 43 86 EC AF 2D E0 48 D3 69 BB 25  ...PRC...-.H.i.%
May 29 18:02:33 03[IKE]   16: 83 FD F7 22                                      ..."
May 29 18:02:33 03[ENC] generating IKE_AUTH request 2 [ AUTH ]
```

Listing 32: Debug Auszug Initiator bei unterschiedlichen Passwörtern

```
May 29 18:02:33 06[IKE] PACESharedSecret => 256 bytes @ 0x97087b0
May 29 18:02:33 06[IKE]    0: 6E 31 29 67 05 AB 26 1B A9 63 FC 47 49 E0 9E 1F  n1)g..&..c.GI...
May 29 18:02:33 06[IKE]   16: F4 51 06 EF 3E 10 89 03 BD 56 62 16 93 74 D9 E9  .Q..>....Vb..t..
May 29 18:02:33 06[IKE]   32: F9 21 70 86 1B 7E 6B 81 43 E1 2D 8A 25 EE 80 EC  .!p..~k.C.-.%...
May 29 18:02:33 06[IKE]   48: 43 ED C7 3E DD D7 44 6B D0 49 27 FE 5F EA DF D9  C..>..Dk.I'._...
May 29 18:02:33 06[IKE]   64: BD D6 FF 92 6A 5E D0 83 37 EB 42 C7 62 D5 98 0E  ....j^..7.B.b...
May 29 18:02:33 06[IKE]   80: 6D 91 E7 3C 28 E0 4F 4C 12 16 B6 35 31 2D 98 0E  m..<(.OL...51-..
May 29 18:02:33 06[IKE]   96: 04 1C AD 72 F4 DE BE DD 72 B6 4E B8 A1 F1 6F EE  ...r....r.N...o.
May 29 18:02:33 06[IKE]  112: F1 41 9F 85 D2 E6 68 84 FE 06 6F 50 67 55 C8 C8  .A....h...oPgU..
May 29 18:02:33 06[IKE]  128: 88 C0 96 88 D6 B7 F1 20 95 3A 3A 0B B1 38 0D FF  ....... .::..8..
May 29 18:02:33 06[IKE]  144: 42 61 56 36 25 A3 AE CF D9 87 F6 30 79 45 1A 72  BaV6%......0yE.r
May 29 18:02:33 06[IKE]  160: 4F D4 F9 28 12 A6 D3 3A 2B CC D7 A7 CB 98 6F 98  O..(...:+.....o.
May 29 18:02:33 06[IKE]  176: 33 E1 9B 06 D2 97 58 AC 4F 5D 26 78 AB 71 72 A1  3.....X.O]&x.qr.
May 29 18:02:33 06[IKE]  192: 5B BA 12 25 EC DF 13 46 6E B6 1B 32 8E 81 6D 6F  [..%...Fn..2..mo
May 29 18:02:33 06[IKE]  208: 26 BA 01 FD 21 29 77 55 BA E1 7C 88 9E C5 DE 92  &...!)wU..|.....
May 29 18:02:33 06[IKE]  224: D5 05 AA 79 D4 00 36 0F C2 73 EE B2 EA FE 23 17  ...y..6..s....#.
May 29 18:02:33 06[IKE]  240: 19 54 A1 CA 4F 18 83 94 25 95 F1 FF 36 F1 DD 3A  .T..O...%...6..:
May 29 18:02:33 06[IKE] AUTHr = prf(prf(Ni | Nr, PACESharedSecret),<ResponderSignedOctets> | PKEi)
=> 20 bytes @ 0x96e80d0
May 29 18:02:33 06[IKE]    0: 54 69 DB 87 34 FA B0 86 E9 B5 86 48 3C DA 48 79  Ti..4......H<.Hy
May 29 18:02:33 06[IKE]   16: F3 E9 AE DC                                      ....
May 29 18:02:33 06[IKE] verification of AUTH payloads failed
```

Listing 33: Debug Responder bei unterschiedlichen Passwörtern

5.1.4 AEAD Verschlüsselung für NONCE

Da die Verwendung von AEAD den Schlüsselaustausch auf Wörterbuchattacken verwundbar macht, wird auf das korrespondierende nicht authentisierende Verschlüsselungsverfahren geschaltet. Um zu testen, ob die richtige Verschlüsselung gewählt wird, wurde mit GCM getestet. Die Konfiguration und die entsprechenden Info- und Debug-Meldungen sind unter Appendix B.7.1 AEAD aufgeführt.

6 Projektstand

Mit dem Abschluss unserer Studienarbeit sind die Voraussetzungen gegeben, dass sich PACE zur Authentisierung in strongSwan mit MODP Diffie-Hellman einsetzen lässt. Vom Auslesen des Passworts, Generieren und verschlüsselte Übertragen der Nonce über die Ableitung der neuen DH-Gruppe, das Generieren des PACESharedSecret bis zum Generieren, Übertragen und Überprüfen der Authentisierung sind alle Bedingungen für eine erfolgreiche Authentisierung zweier strongSwan Kommunikationspartner erfüllt. Darüber hinaus bildet die Implementierung des Secure Password Framework eine Grundlage für die Entwicklung weiterer Plugins, die eine Secure Password Method zur Authentisierung verwenden.
Unsere Arbeit wird als Merge in das von Andreas Steffen und Tobias Brunner betreute strongSwan Repository einfliessen.

6.1 Ausblick

Als Weiterführung unserer Arbeit sehen wir die folgenden Möglichkeiten:

EC-Diffie-Hellman
Für die Generierung des Generators GE im Elliptic Curve Diffie-Hellman-Verfahren muss das Mapping der Nonce mit Hilfe der OpenSSL Elliptic Curve Library erfolgen.
Denkbar wäre eine Auslagerung unseres Mappings in eine generische Methode im `diffie_hellman_t`, die das Mapping für MODP- und EC-Diffie-Hellman erlaubt, obwohl diese momentan nur von PACE beansprucht wird. Dazu könnte man eine weitere Diffie-Hellman Gruppe in `diffie_hellman_group_t` einführen, um diese Mappings zu übernehmen.

Long Term Secret
Das Ableiten und Speichern eines Long-Term-Secret kann entweder durch eine Ersetzung des in der Konfiguration abgelegten initialen Passworts erfolgen oder durch Erzeugen, Aktualisieren und Auslesen eines Files, das die Long-Term-Secrets pro Verbindung auflistet. Möglich wäre damit auch eine interaktive Eingabe des schwachen Passworts durch den Benutzer.

Implementierung weiterer GSPM Methoden
In Zukunft könnten weitere GSPM Methoden als Plugins in strongSwan einfliessen.
Momentan definierte Secure Password Verfahren wären (Stand Juni 2013):

- “Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2” (AugPAKE, RFC6628), http://tools.ietf.org/html/rfc6628
- “Secure PSK Authentication for IKE” (Draft): http://tools.ietf.org/html/draft-harkins-ipsecme-spsk-auth-06

7 Projektmanagement

7.1 Projektplan

Gegenüber der ursprünglichen Planung (Abbildung 32) wurden die optionalen Funktionalitäten 3.5 Creating a LTS,3.6 Using the LTS nicht implementiert. Der Aufwand für die Implementierung der GSPM Authentisierung und das Einarbeiten in die strongSwan Architektur war grösser als ursprünglich angenommen.

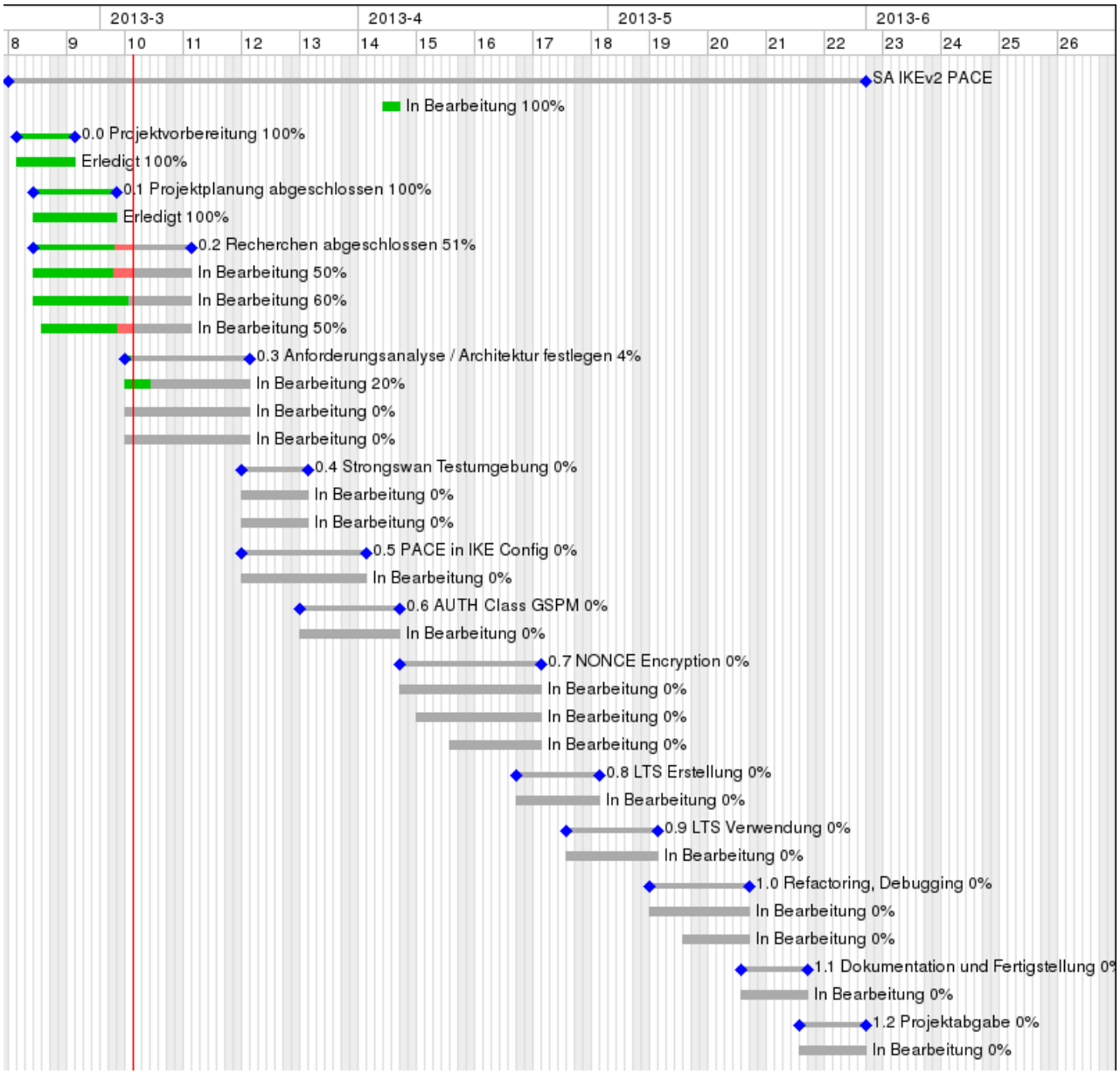

Abbildung 32: Projektplan zu Beginn

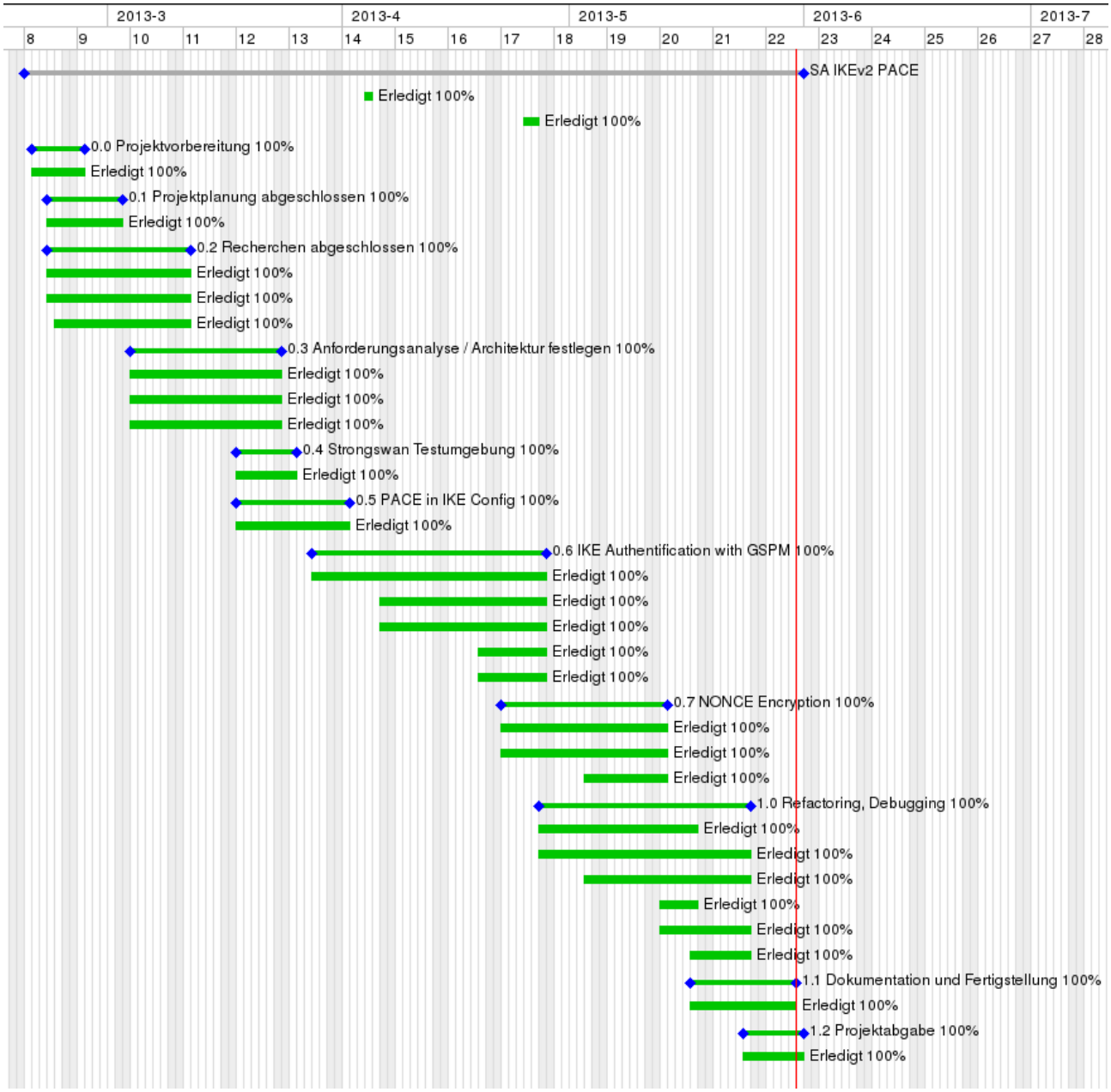

Abbildung 33: Projektplan bei Fertigstellung

7.2 Zeitabrechnung

Das folgende Diagramm (Abbildung 34) zeigt die aufgewendete Zeit pro Person und Kalenderwoche an. Schwankungen kamen zustande durch unterschiedliche Arbeitsaufteilung beim Codieren (Reto Schelbert) und Dokumentieren (Simon Schreiber) gegen Ende des Projekts.

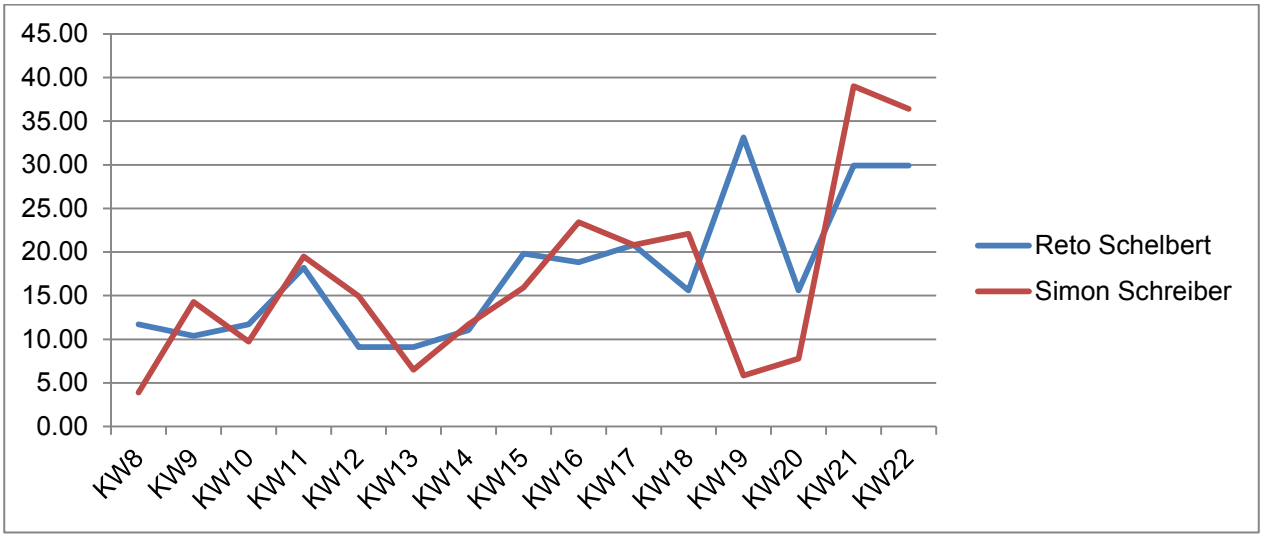

Abbildung 34: Zeitdiagramm aufgewendeter Zeit pro Person und Kalenderwoche

Viel Aufwand, wie zu erwarten, gab die Implementierung selbst (Development) und die Dokumentierung (Documentation), wie in folgendem Diagramm aufgezeigt (Abbildung 35)

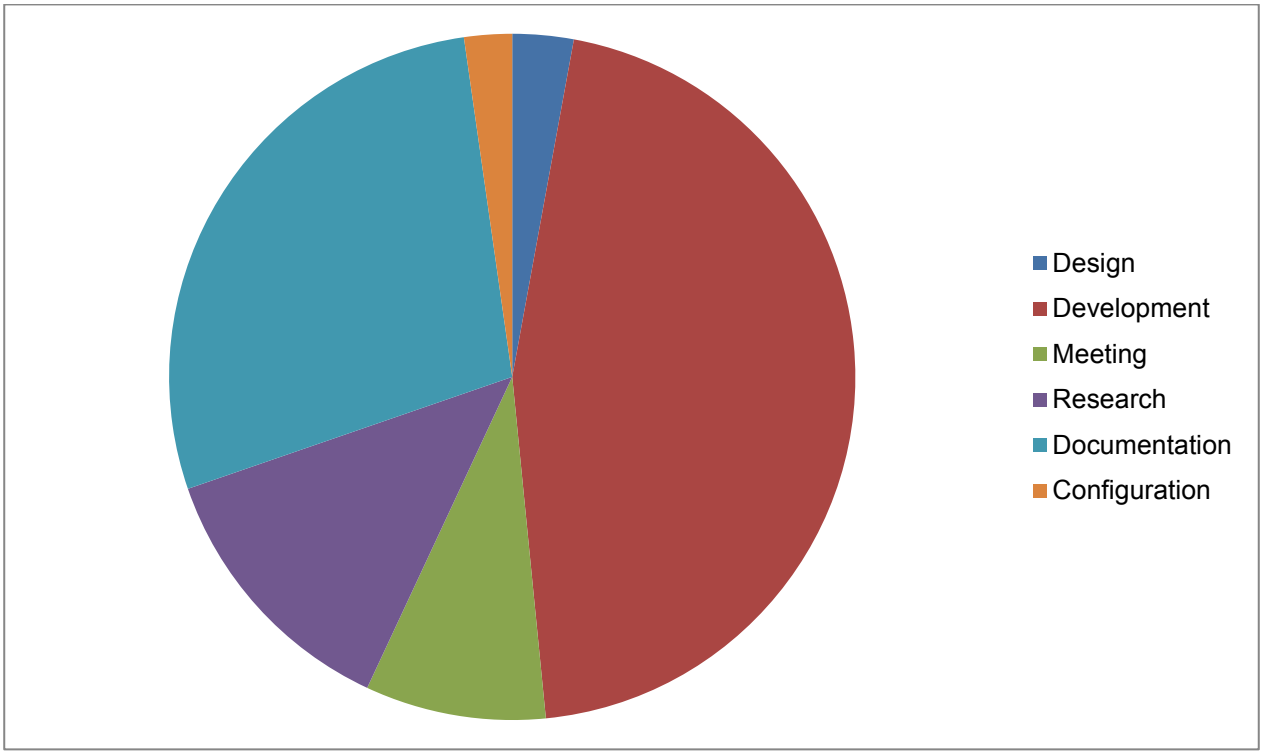

Abbildung 35: Kuchendiagramm aufgewendeter Zeit der Tätigkeiten

Von Kalenderwoche 8 bis 22 sind dies insgesamt 15 Wochen, wobei für die SA 240h pro Person verlangt wird.

Wer	Durchschnitt (h/Woche)	Gesamtaufwand (h)
Reto Schelbert	17.7	264.9
Simon Schreiber	16.8	251.9
SA verlangter Aufwand	16	240

A. Erfahrungsberichte

A.1 Reto Schelbert

Die Arbeit an einem grösseren Open Source Projekt war eindrücklich und lernreich.
Während der Semesterarbeit habe ich meine Programmierkenntnisse mit C fortlaufend ausgebaut und konnte tiefere Einblicke gewinnen in objektorientierte Ansätze, welche in strongSwan sehr gut realisiert wurden. Somit hat sich die Implementierung auch etwas komfortabler gestaltet, hat man diese Konzepte erst einmal begriffen.

Zu Beginn des Projekts war sehr viel Einarbeitung nötig bis wir mit der eigentlichen Implementierung beginnen konnten. Dabei hat man strongSwan auch als Produkt kennen gelernt und hat dessen Vielfältigkeit entdeckt, wobei wir aber gerade auf Grund der vielen implementierten und ineinander zahnende Funktionen strongSwan gerne einmal als „komplexes Biest“ bezeichnet haben. Als Teilzeitarbeitender bin ich nebenbei erwähnt überzeugt, diese Lösung bei einigen unserer Kunden in Zukunft ein zu setzen.
Das Thema war aus meiner Sicht sehr spannend und hat von der hohen Komplexität her einen gewissen Anreiz gegeben, mich mehr mit Authentifizierungsverfahren zu beschäftigen. Eine Implementation gemäss einem Request for Comments zu realisieren, hat mir einige Erleuchtungen gegeben, an wen sich solche RFCs richten und wie detailliert, oder eben auch nicht, diese umschrieben sind.

Rückblickend betrachtet, haben wir sicher viel Zeit benötigt um bestehenden Code durch zu lesen und zu durchforsten, um auch an den Meetings die Voraussetzungen für unsere Implementierung ab zu klären. Die Zusammenarbeit verlief aber stets positiv und ich möchte mich herzlichst bei Andreas Steffen und Tobias Brunner bedanken für viele nützliche Inputs und Ihre Bemühungen, wann immer möglich zusammen mit uns eine Lösung zu finden. Dank Anregungen und Besprechungen während der Studienarbeit, hat man uns gut auf eine Bachelorarbeit vorbereitet.

A.2 Simon Schreiber

Da meine Interessensgebiete in Kryptografie und Netzwerkanwendungen liegen, hab ich mich ausserordentlich gefreut, eine Studienarbeit in diesem Bereich in Angriff nehmen zu dürfen. Dass die Ergebnisse in das weit verbreitete und sehr aktive OpenSource Projekt strongSwan zum produktiven Einsatz einfliessen sollten, stellte einen zusätzlichen Motivationsschub dar.

Die Einarbeitung in die Grundlagen von IKEv2, strongSwan und die benötigten RFCs war zwar umfangreich und mit erheblichem Arbeitsaufwand verbunden. Die klare Struktur der RFCs und die gut verständliche Dokumentation kamen mir dabei aber sehr entgegen.
Steil war für mich die Lernkurve bei C als Programmiersprache in der objekt-orientierten Form, wie sie in strongSwan eingesetzt wird. Im Nachhinein möchte ich dies aber nicht missen und habe Gefallen daran gefunden.

Da unsere Implementierung mit zahlreichen Komponenten der bestehenden Software interagiert, stellte auch die notwendige Kenntnis von Architektur und Code von strongSwan eine Hürde dar, die ich zu Beginn unterschätzt habe. Im Verlauf der ersten beiden Monate stiessen wir oft auf unvermutete Hindernisse, sobald wir hofften, einen gangbaren Weg für die Umsetzung gefunden zu haben. Dann hiess es einen Schritt zurück zu gehen und mit neuem Ansatz fortfahren. Doch nicht zuletzt dank diesen ersten Stolpersteinen denke ich, dass ich mich mittlerweile in einigen Ecken von strongSwan gut auskenne.

Hilfreich waren bei der Orientierung die zahlreichen Meetings, in denen uns mit Herrn Prof. Dr. Andreas Steffen und Tobias Brunner mit ihrer ganzen Kompetenz und viel Einsatz zur Seite standen. Das angenehme und fokussierte Arbeitsklima in den Meetings hat viel zur Entwicklung der Arbeit beigetragen.

Dass ich mit Reto für diese Arbeit einen Partner habe, mit dem sich gut zusammen arbeiten lässt, wusste ich schon aus anderen Projekten. Einmal mehr hat er mich aber oft mit seiner Einsatzfreude und Effizienz überrascht. Von entscheidendem Wert waren diese Qualitäten gerade in Engpässen im Zeitplan aufgrund unserer Teilzeit-Arbeitstätigkeit. Rückblickend war die Studienarbeit eine gute und intensive Zeit und ich denke, dass ich mit dieser Erfahrung gut gerüstet bin, die Bachelor-Arbeit anzugehen.

B. Appendix

B.1 Abbildungen

B.2 Listings

B.3 Formeln

B.4 Literaturverzeichnis

[1] Hoffman und Kaufman, RFC5996 Internet Key Exchange Protocol Version 2 (IKEv2), IETF, 2010.

[2] Kivinen, RFC6467 Secure Password Framework for Internet Key Exchange Version 2 (IKEv2), IETF, 2011.

[3] Sheffer und Kuegler, RFC6631 IKEv2 with PACE, IETF, 2012.

[4] IANA, „IANA Assignements, IKEv2 Parameters," 2013. [Online]. Available: http://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xml. [Zugriff am April 2013].

[5] Bender, Fischlin und Kügler, „Security Analysis of the PACE Key-Agreement Protocol," BSI, Deutschland, 2009.

[6] I. J. S. WG3/TF5, „ICAO: TECHNICAL REPORT: Supplemental Access Control forMachine Readable Travel Documents," [Online]. Available: http://www.icao.int/Security/mrtd/Downloads/Technical%20Reports/Technical%20Report.pdf.

[7] J. Hutter und M. Willi , „ strongSwan II, Eine IKEv2-Implementierung für Linux.," Hochschule für Technik Rapperswil, 2005.

[8] strongSwan, „strongSwan Project," 2005. [Online]. Available: http://wiki.strongswan.org/projects/strongswan/wiki/DeveloperDocumentation. [Zugriff am März 2013].

[9] Xine Project, „Xine Project," 2001. [Online]. Available: http://www.xine-project.org/hackersguide. [Zugriff am März 2013].

[10] „Wikipedia," [Online]. Available: http://www.en.wikipedia.org.

[11] Larmann, Applying UML and Patterns, mitp, 2005.

B.6 Abkürzungsverzeichnis

Abkürzung	Bedeutung
IKEv2	Internet Key Exchange Version 2
PACE	Password Authenticated Connection Establishment
ZKPP	Zero Knowledge Password Proof
IPsec	Internet Protocol Security
VPN	Virtual Private Network
PSK	Pre Shared Secret
EAP	Extensible Authentication Protocol
AAA	Authentication, Authorisation and Accounting
DSS	Digital Signature Standard
RSA	Rimest, Shamir, Adleman
NIST	National Institute of Standards and Technology
IANA	Internet Assigned Numbers Authority
RFC	Request for Comments
LTS	Long Term Secret
MAC	Message Authentication Code
GMP	GNU Multiple Precision Arithmetic Library
AEAD	Authenticated Encryption with Associated Data
EC-DH	Elliptic Curve Diffie-Hellman
MODP-DH	Modular Diffie-Hellmn
CERT	Certificate
CERTREQ	Certificate Request
EAP	Extensible Authentication
HDR	IKE Header (not a payload), contains SPI, version, flags
SPIi, SPIr	SPI, Security Parameter Index, identify under which SA a packet is processed
IDi/IDr	Identification – initiator/responder
KEr/KEi	Key Exchange initiator/responder (Diffie-Hellmann public value)
Ni, Nr	Nonce Initiator, Responder
N	Notify
SK	Encrypted and Authenticated
TSi / TSr	Traffic Selector – initiator/responder
V	Vendor ID
SA	Security Association
SAi / SAr	states the cryptographic algorithms the initiator/responder supports
AUTHi / AUTHr	authentication payload initiator/responder, authenticates the previous message and the responder identity
GSPM	Generic Secure Password Method
PRF	PseudoRandom Function
PAKE	Password-Authenticated Key Exchange

E()	Symmetric encryption
D()	Symmetric decryption
KA()	Key agreement
Map()	Mapping function
Pwd	Shared password
SPwd	Stored password
KPwd	Sysmmetric key derived from a password Pwd
G	DH Basis G uses for static group generator
GE	Ephemeral (flüchtig/eintägig) group generator
IV	Initialization Vector
ENONCE	Encrypted nonce
PKEi / PKEr	Ephemeral public key of initiator/responder (2. Diffie-Hellman Austausch in PACE)
SKEi / SKEr	Ephemeral secret key of the initiator/responder (2. Diffie-Hellman Austausch in PACE)
CBC-MAC	Cipher Block Chaining Message Authentication Code
CCM	Counter with CBC-MAC
GCM	Galois/Counter Mode

B.7 TestLogs

B.7.1 AEAD

Client config

```
config setup

conn %default
…
        authby=gspm
        keyexchange=ikev2
        ike=aes256gcm16-aesxcbc-modp2048!
        esp=aes256gcm16-modp2048!

conn net-net
        left=10.10.10.10
        leftsubnet=192.168.10.0/24
        leftid=@moon.strongswan.org
        right=10.10.10.20
        rightsubnet=192.168.20.0/24
        rightid=@sun.strongswan.org
```

Server Config

```
config setup

conn %default
…
    authby=gspm
    keyexchange=ikev2
    ike=aes256gcm16-aesxcbc-modp2048!
    esp=aes256gcm16-modp2048!

conn net-net
    left=10.10.10.20
    leftsubnet=192.168.20.0/24
    leftid=@sun.strongswan.org
    right=10.10.10.10
    rightsubnet=192.168.10.0/24
    rightid=@moon.strongswan.org
```

Log Client normal

```
initiating IKE_SA net-net[1] to 10.10.10.20
generating IKE_SA_INIT request 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(SEC_PASSWD) ]
sending packet: from 10.10.10.10[500] to 10.10.10.20[500] (434 bytes)
received packet: from 10.10.10.20[500] to 10.10.10.10[500] (442 bytes)
parsed IKE_SA_INIT response 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(MULT_AUTH) N(SEC_PASSWD) ]
secure password method PACE has been selected by responder
authentication of 'moon.strongswan.org' (myself) with secure password PACE
changing encryption algorithm for nonce encryption from AES_GCM_16 to AES_CTR
establishing CHILD_SA net-net
generating IKE_AUTH request 1 [ IDi N(INIT_CONTACT) IDr SA TSi TSr GSPM KE N(MULT_AUTH) N(EAP_ONLY) ]
sending packet: from 10.10.10.10[500] to 10.10.10.20[500] (527 bytes)
received packet: from 10.10.10.20[500] to 10.10.10.10[500] (347 bytes)
parsed IKE_AUTH response 1 [ IDr KE ]
allow mutual EAP/GSPM-only authentication
generating IKE_AUTH request 2 [ AUTH ]
sending packet: from 10.10.10.10[500] to 10.10.10.20[500] (81 bytes)
received packet: from 10.10.10.20[500] to 10.10.10.10[500] (177 bytes)
parsed IKE_AUTH response 2 [ AUTH SA TSi TSr N(AUTH_LFT) ]
authentication of 'sun.strongswan.org' with secure password method PACE successful
IKE_SA net-net[1] established between 10.10.10.10[moon.strongswan.org]...10.10.10.20[sun.strongswan.org]
scheduling reauthentication in 3266s
maximum IKE_SA lifetime 3446s
```

Log Server IKE DBG4

```
May 27 11:29:59 00[CFG] loading aa certificates from '/etc/ipsec.d/aacerts'
May 27 11:29:59 00[CFG] loading ocsp signer certificates from '/etc/ipsec.d/ocspcerts'
May 27 11:29:59 00[CFG] loading attribute certificates from '/etc/ipsec.d/acerts'
May 27 11:29:59 00[CFG] loading crls from '/etc/ipsec.d/crls'
May 27 11:29:59 00[CFG] loading secrets from '/etc/ipsec.secrets'
May 27 11:29:59 00[CFG]   loaded IKE secret for @moon.strongswan.org @sun.strongswan.org
May 27 11:29:59 00[DMN] loaded plugins: charon aes des sha1 sha2 md5 random nonce x509 revocation constraints pubkey pkcs1
pkcs8 pgp dnskey pem fips-prf gmp xcbc cmac hmac ctr ccm gcm attr kernel-netlink resolve socket-default stroke updown gspm-pace
xauth-generic
May 27 11:29:59 00[JOB] spawning 16 worker threads
May 27 11:29:59 14[CFG] received stroke: add connection 'net-net'
May 27 11:29:59 14[CFG] added configuration 'net-net'
May 27 11:30:36 05[NET] received packet: from 10.10.10.10[500] to 10.10.10.20[500] (434 bytes)
May 27 11:30:36 05[ENC] parsed IKE_SA_INIT request 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(SEC_PASSWD) ]
May 27 11:30:36 05[IKE] 10.10.10.10 is initiating an IKE_SA
May 27 11:30:36 05[IKE] IKE_SA (unnamed)[1] state change: CREATED => CONNECTING
May 27 11:30:36 05[IKE] natd_chunk => 22 bytes @ 0x81850c8
May 27 11:30:36 05[IKE]    0: AB A9 AB C8 6E 45 34 84 00 00 00 00 00 00 00 00  ....nE4.........
May 27 11:30:36 05[IKE]   16: 0A 0A 0A 14 01 F4                              ......
May 27 11:30:36 05[IKE] natd_hash => 20 bytes @ 0x8184d48
May 27 11:30:36 05[IKE]    0: F0 2B E3 09 EA E9 1D 4C ED 82 87 E9 6C 68 DD 92  .+.....L....lh..
May 27 11:30:36 05[IKE]   16: 91 62 05 E5                                    .b..
May 27 11:30:36 05[IKE] natd_chunk => 22 bytes @ 0x81850c8
May 27 11:30:36 05[IKE]    0: AB A9 AB C8 6E 45 34 84 00 00 00 00 00 00 00 00  ....nE4.........
May 27 11:30:36 05[IKE]   16: 0A 0A 0A 0A 01 F4                              ......
May 27 11:30:36 05[IKE] natd_hash => 20 bytes @ 0x81831c0
May 27 11:30:36 05[IKE]    0: C6 88 08 2D 4E 34 D4 BB 74 08 F8 D7 2A 99 8B 26  ...-N4..t...*..&
May 27 11:30:36 05[IKE]   16: 8F CF 7F DE                                    ....
May 27 11:30:36 05[IKE] precalculated src_hash => 20 bytes @ 0x81831c0
May 27 11:30:36 05[IKE]    0: C6 88 08 2D 4E 34 D4 BB 74 08 F8 D7 2A 99 8B 26  ...-N4..t...*..&
May 27 11:30:36 05[IKE]   16: 8F CF 7F DE                                    ....
May 27 11:30:36 05[IKE] precalculated dst_hash => 20 bytes @ 0x8184d48
May 27 11:30:36 05[IKE]    0: F0 2B E3 09 EA E9 1D 4C ED 82 87 E9 6C 68 DD 92  .+.....L....lh..
May 27 11:30:36 05[IKE]   16: 91 62 05 E5                                    .b..
```

```
May 27 11:30:36 05[IKE] received src_hash => 20 bytes @ 0x8181e88
May 27 11:30:36 05[IKE]    0: C6 88 08 2D 4E 34 D4 BB 74 08 F8 D7 2A 99 8B 26  ...-N4..t...*..&
May 27 11:30:36 05[IKE]   16: 8F CF 7F DE                                      ....
May 27 11:30:36 05[IKE] received dst_hash => 20 bytes @ 0x81830f0
May 27 11:30:36 05[IKE]    0: F0 2B E3 09 EA E9 1D 4C ED 82 87 E9 6C 68 DD 92  .+.....L....lh..
May 27 11:30:36 05[IKE]   16: 91 62 05 E5                                      .b..
May 27 11:30:36 05[IKE] secure password method PACE selected
May 27 11:30:36 05[IKE] shared Diffie Hellman secret => 256 bytes @ 0x8186090
May 27 11:30:36 05[IKE]    0: B4 CF FB F0 55 C4 7C 65 C7 96 94 C9 62 7F 3B F1  ....U.|e....b.;.
May 27 11:30:36 05[IKE]   16: D2 B8 7D 9A BB 09 82 AA 28 2B 0D B0 C2 E9 F4 9A  ..}.....(+......
May 27 11:30:36 05[IKE]   32: FA 4D 99 49 CE 75 39 8D A0 4A 78 8B 3E 89 BD 58  .M.I.u9..Jx.>..X
May 27 11:30:36 05[IKE]   48: 7D A1 18 B6 3F 4A 01 ED 8F 8A A3 50 B8 DB 42 FF  }...?J.....P..B.
May 27 11:30:36 05[IKE]   64: 1C CD 12 2E 32 33 4D 73 37 5E 7A A5 CA C5 B9 25  ....23Ms7^z....%
May 27 11:30:36 05[IKE]   80: 11 5A 6C 6A BA 99 10 70 8F 8C CA 96 1B 2A 47 F4  .Zlj...p.....*G.
May 27 11:30:36 05[IKE]   96: AC 2D 31 C2 66 D2 8B 33 03 47 C3 5F FC 0D A8 62  .-1.f..3.G._...b
May 27 11:30:36 05[IKE]  112: EF 03 37 95 E9 99 93 71 3A A0 5D 5F 02 8A B1 E4  ..7....q:.]_....
May 27 11:30:36 05[IKE]  128: 07 9E 43 7D 5D 94 6A 46 AF BF 9B 32 09 01 BF 53  ..C}].jF...2...S
May 27 11:30:36 05[IKE]  144: F2 F5 C7 00 AD 05 B4 B4 2C 58 ED 55 35 B6 46 5B  ........,X.U5.F[
May 27 11:30:36 05[IKE]  160: 3F CB 86 48 95 5C 3F C6 60 FE 28 97 F5 AB B9 26  ?..H.\?.`.(....&
May 27 11:30:36 05[IKE]  176: 5C 2A 03 75 4E CE A0 13 6E 5C 1E 38 72 7B 8D 4F  \*.uN...n\.8r{.O
May 27 11:30:36 05[IKE]  192: 89 01 74 52 5E CC B0 BF 00 91 98 52 13 A8 17 B5  ..tR^......R....
May 27 11:30:36 05[IKE]  208: 77 48 28 F4 99 58 33 20 9C 0E 38 77 19 34 A5 72  wH(..X3 ..8w.4.r
May 27 11:30:36 05[IKE]  224: 41 16 ED A4 E8 85 F9 4F D4 9B FC A9 BE BC EF E4  A......O........
May 27 11:30:36 05[IKE]  240: EC 09 E3 C1 E5 64 59 90 4F 70 A9 C8 29 FD 48 6F  .....dY.Op..).Ho
May 27 11:30:36 05[IKE] SKEYSEED => 16 bytes @ 0x8186660
May 27 11:30:36 05[IKE]    0: 1D EB 8C 2F 81 13 06 93 7A 8D 73 68 69 E6 CE 5D  .../....z.shi..]
May 27 11:30:36 05[IKE] Sk_d secret => 16 bytes @ 0x81831c0
May 27 11:30:36 05[IKE]    0: 0B 94 31 71 95 74 B5 6F 66 C4 0B 20 47 46 F0 B6  ..1q.t.of.. GF..
May 27 11:30:36 05[IKE] Sk_ei secret => 36 bytes @ 0x8186638
May 27 11:30:36 05[IKE]    0: EA BB AE 83 36 95 4D BC 38 91 EB 7A 3B 17 30 BD  ....6.M.8..z;.0.
May 27 11:30:36 05[IKE]   16: 34 69 92 9A FC A4 C5 72 B3 21 27 65 F5 3D EF BC  4i.....r.!'e.=..
May 27 11:30:36 05[IKE]   32: A9 C5 36 FD                                      ..6.
May 27 11:30:36 05[IKE] Sk_er secret => 36 bytes @ 0x8186638
May 27 11:30:36 05[IKE]    0: D2 75 FD 9C 72 FE 11 17 C3 68 CC 36 7C 7E 15 F4  .u..r....h.6|~..
May 27 11:30:36 05[IKE]   16: 4A 51 F3 7F B3 3E 56 69 AB 89 C9 45 55 7D 10 0E  JQ...>Vi...EU}..
May 27 11:30:36 05[IKE]   32: 1E AB 4A AB                                      ..J.
May 27 11:30:36 05[IKE] Sk_pi secret => 16 bytes @ 0x8186660
May 27 11:30:36 05[IKE]    0: 8A 8F 07 9C 2D 61 2F F3 AB 70 86 D7 6E A7 9B D8  ....-a/..p..n...
May 27 11:30:36 05[IKE] Sk_pr secret => 16 bytes @ 0x8186620
May 27 11:30:36 05[IKE]    0: 1A CD C2 CD B2 C7 E6 85 23 85 DA 5A BA 21 CE F4  ........#..Z.!..
May 27 11:30:36 05[IKE] natd_chunk => 22 bytes @ 0x81865c0
May 27 11:30:36 05[IKE]    0: AB A9 AB C8 6E 45 34 84 F3 8D B0 E1 2E 00 99 7C  ....nE4........|
May 27 11:30:36 05[IKE]   16: 0A 0A 0A 14 01 F4                                ......
May 27 11:30:36 05[IKE] natd_hash => 20 bytes @ 0x8186700
May 27 11:30:36 05[IKE]    0: 58 6D 9B 83 DE 08 A0 15 E4 F0 00 C5 D2 24 D8 4C  Xm...........$.L
May 27 11:30:36 05[IKE]   16: B0 6F 65 B2                                      .oe.
May 27 11:30:36 05[IKE] natd_chunk => 22 bytes @ 0x81865c0
May 27 11:30:36 05[IKE]    0: AB A9 AB C8 6E 45 34 84 F3 8D B0 E1 2E 00 99 7C  ....nE4........|
May 27 11:30:36 05[IKE]   16: 0A 0A 0A 0A 01 F4                                ......
May 27 11:30:36 05[IKE] natd_hash => 20 bytes @ 0x8186700
May 27 11:30:36 05[IKE]    0: 74 50 F5 F3 86 F3 36 A0 65 2E 47 41 1F 25 16 40  tP....6.e.GA.%.@
May 27 11:30:36 05[IKE]   16: 66 3E 98 AE                                      f>..
May 27 11:30:36 05[ENC] generating IKE_SA_INIT response 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(MULT_AUTH)
N(SEC_PASSWD) ]
May 27 11:30:36 05[NET] sending packet: from 10.10.10.20[500] to 10.10.10.10[500] (442 bytes)
May 27 11:30:36 04[NET] received packet: from 10.10.10.10[500] to 10.10.10.20[500] (527 bytes)
May 27 11:30:36 04[ENC] parsed IKE_AUTH request 1 [ IDi N(INIT_CONTACT) IDr SA TSi TSr GSPM KE N(MULT_AUTH)
N(EAP_ONLY) ]
May 27 11:30:36 04[CFG] looking for peer configs matching 10.10.10.20[sun.strongswan.org]...10.10.10.10[moon.strongswan.org]
May 27 11:30:36 04[CFG] selected peer config 'net-net'
May 27 11:30:36 04[IKE] authentication of 'sun.strongswan.org' (myself) with secure password PACE
```

```
May 27 11:30:36 04[IKE] changing encryption algorithm for nonce encryption from AES_GCM_16 to AES_CTR
May 27 11:30:36 04[IKE] encryptet nonce s => 32 bytes @ 0x8185bc9
May 27 11:30:36 04[IKE]    0: 2F CA 07 60 F9 D9 F6 2E 92 5D E4 26 93 2C 13 C1  /..`.....].&.,..
May 27 11:30:36 04[IKE]   16: CE 7A 85 8E 07 D1 F9 B0 1F CB 54 88 68 46 DD 42  .z........T.hF.B
May 27 11:30:36 04[IKE] nonce s => 32 bytes @ 0x8183180
May 27 11:30:36 04[IKE]    0: 65 BB 1A BE 15 C9 B8 12 DB 54 6F 1D 8A EA F2 F7  e........To.....
May 27 11:30:36 04[IKE]   16: 65 82 DD 8D B9 62 EE A0 40 C2 1E 56 82 F1 D0 39  e....b..@..V...9
May 27 11:30:36 04[ENC] generating IKE_AUTH response 1 [ IDr KE ]
May 27 11:30:36 04[NET] sending packet: from 10.10.10.20[500] to 10.10.10.10[500] (347 bytes)
May 27 11:30:36 03[NET] received packet: from 10.10.10.10[500] to 10.10.10.20[500] (81 bytes)
May 27 11:30:36 03[ENC] parsed IKE_AUTH request 2 [ AUTH ]
May 27 11:30:36 03[IKE] IDx' => 23 bytes @ 0xb6411e80
May 27 11:30:36 03[IKE]    0: 02 00 00 00 6D 6F 6F 6E 2E 73 74 72 6F 6E 67 73  ....moon.strongs
May 27 11:30:36 03[IKE]   16: 77 61 6E 2E 6F 72 67                           wan.org
May 27 11:30:36 03[IKE] SK_p => 16 bytes @ 0x8186660
May 27 11:30:36 03[IKE]    0: 8A 8F 07 9C 2D 61 2F F3 AB 70 86 D7 6E A7 9B D8  ....-a/..p..n...
May 27 11:30:36 03[IKE] octets = message + nonce + prf(Sk_px, IDx') => 482 bytes @ 0x81875a8
May 27 11:30:36 03[IKE]    0: AB A9 AB C8 6E 45 34 84 00 00 00 00 00 00 00 00  ....nE4.........
May 27 11:30:36 03[IKE]   16: 21 20 22 08 00 00 00 00 00 00 01 B2 22 00 00 28  ! "........."..(
May 27 11:30:36 03[IKE]   32: 00 00 00 24 01 01 00 03 03 00 00 0C 01 00 00 14  ...$............
May 27 11:30:36 03[IKE]   48: 80 0E 01 00 03 00 00 08 02 00 00 04 00 00 00 08  ................
May 27 11:30:36 03[IKE]   64: 04 00 00 0E 28 00 01 08 00 0E 00 00 CA 03 8E E2  ....(...........
May 27 11:30:36 03[IKE]   80: 3A CA C5 32 7E 4D 42 E3 86 D6 37 ED 66 31 33 5B  :..2~MB...7.f13[
May 27 11:30:36 03[IKE]   96: 20 76 BD 78 88 F3 03 30 D0 76 BE 2B F3 24 C4 21   v.x...0.v.+.$.!
May 27 11:30:36 03[IKE]  112: 7B 68 99 08 88 8C BA 27 01 BB 35 AF 1D 02 56 F2  {h.....'..5...V.
May 27 11:30:36 03[IKE]  128: CC A2 24 6A 2B C9 EC F3 82 9F 59 E9 26 C6 A4 BA  ..$j+.....Y.&...
May 27 11:30:36 03[IKE]  144: 9A 5B D4 73 15 E7 AA 64 AC 73 7A BF 82 16 80 D8  .[.s...d.sz.....
May 27 11:30:36 03[IKE]  160: 06 1C 40 63 4A C4 6E 82 C8 01 14 39 70 CC 1E 6B  ..@cJ.n....9p..k
May 27 11:30:36 03[IKE]  176: 79 4C 3B E7 7C 10 B6 8E DC F4 B0 B3 31 F1 47 E8  yL;.|.......1.G.
May 27 11:30:36 03[IKE]  192: 04 3E 9E 99 B4 63 96 45 F0 44 15 2F 78 54 42 BE  .>...c.E.D./xTB.
May 27 11:30:36 03[IKE]  208: AD 7B A0 7F EE B2 CD E7 3F FE 33 80 4F 83 3E C1  .{......?.3.O.>.
May 27 11:30:36 03[IKE]  224: 6E 08 72 35 EC CA E4 90 F5 EE 8E B6 72 8B E6 FF  n.r5........r...
May 27 11:30:36 03[IKE]  240: 1C 13 C9 1F CE 63 B3 69 A6 CC 22 B0 F3 9B 0E 57  .....c.i.."....W
May 27 11:30:36 03[IKE]  256: 01 94 94 6C E6 82 87 1B 6B C6 69 B8 86 1F 6C F8  ...l....k.i...l.
May 27 11:30:36 03[IKE]  272: CB 74 1B 3C 95 43 B4 F4 E6 AF 0A 35 10 B2 20 BE  .t.<.C.....5.. .
May 27 11:30:36 03[IKE]  288: 5C D4 45 45 E0 91 C4 56 19 6E 09 32 F7 DA 2C 28  \.EE...V.n.2..,(
May 27 11:30:36 03[IKE]  304: 31 53 A2 42 52 F9 40 82 8D F4 23 1C 8F 09 DF 60  1S.BR.@...#....`
May 27 11:30:36 03[IKE]  320: 53 66 DD F9 12 E1 66 F1 7A 0F E1 0C 29 00 00 24  Sf....f.z...)..$
May 27 11:30:36 03[IKE]  336: A8 72 33 11 11 02 92 83 F6 72 BF 56 41 80 C0 18  .r3......r.VA...
May 27 11:30:36 03[IKE]  352: 67 A0 A6 CB 01 EB 40 DC 1D 47 41 3C 3B D5 09 FF  g.....@..GA<;...
May 27 11:30:36 03[IKE]  368: 29 00 00 1C 00 00 40 04 C6 88 08 2D 4E 34 D4 BB  ).....@....-N4..
May 27 11:30:36 03[IKE]  384: 74 08 F8 D7 2A 99 8B 26 8F CF 7F DE 29 00 00 1C  t...*..&....)...
May 27 11:30:36 03[IKE]  400: 00 00 40 05 F0 2B E3 09 EA E9 1D 4C ED 82 87 E9  ..@..+.....L....
May 27 11:30:36 03[IKE]  416: 6C 68 DD 92 91 62 05 E5 00 00 00 0A 00 00 40 28  lh...b........@(
May 27 11:30:36 03[IKE]  432: 00 01 A8 72 33 11 11 02 92 83 F6 72 BF 56 41 80  ...r3......r.VA.
May 27 11:30:36 03[IKE]  448: C0 18 67 A0 A6 CB 01 EB 40 DC 1D 47 41 3C 3B D5  ..g.....@..GA<;.
May 27 11:30:36 03[IKE]  464: 09 FF 98 0B 89 C8 38 47 CB 91 D3 5A 80 48 A1 3F  ......8G...Z.H.?
May 27 11:30:36 03[IKE]  480: E0 01                                           ..
May 27 11:30:36 03[IKE] PACESharedSecret => 256 bytes @ 0x8183248
May 27 11:30:36 03[IKE]    0: D7 02 2A 19 CA 84 89 0E 4B 4C 78 F4 51 23 0C 72  ..*.....KLx.Q#.r
May 27 11:30:36 03[IKE]   16: AC 92 B3 DF AC 3F 38 00 D5 C8 AA 62 63 E9 64 97  .....?8....bc.d.
May 27 11:30:36 03[IKE]   32: 08 58 26 39 0B 69 32 5A CA 1D E2 98 CF 34 B0 DA  .X&9.i2Z.....4..
May 27 11:30:36 03[IKE]   48: 1E A3 E7 CA 78 F6 A0 70 B7 8B 30 F6 01 6D A0 6D  ....x..p..0..m.m
May 27 11:30:36 03[IKE]   64: 3E 1F 32 4D 84 58 60 77 EA 01 8E 65 61 37 2A 2E  >.2M.X`w...ea7*.
May 27 11:30:36 03[IKE]   80: 99 5E 7B 79 48 F5 6C 0A 0F 42 B9 FA AA B5 C9 A4  .^{yH.l..B......
May 27 11:30:36 03[IKE]   96: FC A0 40 0D 26 7B 6B 9B 35 97 DB 11 29 37 CB 71  ..@.&{k.5...)7.q
May 27 11:30:36 03[IKE]  112: D8 65 42 78 32 68 DB 68 84 9B A3 A2 EB 7B 58 2C  .eBx2h.h.....{X,
May 27 11:30:36 03[IKE]  128: 8B 6D D1 30 40 95 BC F1 FF 24 AF 07 5C B1 31 EF  .m.0@....$..\.1.
May 27 11:30:36 03[IKE]  144: 86 3E FB 04 4F 0D 74 7E C1 57 AE B6 AA 00 75 4D  .>..O.t~.W....uM
May 27 11:30:36 03[IKE]  160: FE 5C 99 FF 78 AD C6 B0 A7 5B 23 72 39 02 03 F3  .\..x....[#r9...
May 27 11:30:36 03[IKE]  176: 75 59 81 8A 2A EF 51 19 56 3B DF EA 6E 52 B8 EA  uY..*.Q.V;..nR..
```

```
May 27 11:30:36 03[IKE]  192: 9D 8D 98 13 61 32 D8 C8 F8 94 13 BA 6B 8B D2 62  ....a2......k..b
May 27 11:30:36 03[IKE]  208: D7 F2 DE A7 39 59 7F 0B AA 1F 16 AA 26 9F F7 B8  ....9Y......&...
May 27 11:30:36 03[IKE]  224: F6 61 8A 89 7F B8 D9 22 2D AF D4 B8 BA 47 7E 0C  .a.....".-....G~.
May 27 11:30:36 03[IKE]  240: BF 83 1B 34 2C 9A 19 40 64 23 FD 56 58 CE 96 53  ...4,..@d#.VX..S
May 27 11:30:36 03[IKE] AUTHr = prf(prf(Ni | Nr, PACESharedSecret),<ResponderSignedOctets> | PKEi) => 16 bytes @ 0x8185b70
May 27 11:30:36 03[IKE]    0: BA 3A 06 21 8B 2E 4F C7 CB E4 65 9C C3 FD F3 D7  .:.!..O...e.....
May 27 11:30:36 03[IKE] verified AUTH payloads successfully
May 27 11:30:36 03[IKE] authentication of 'moon.strongswan.org' with secure password method PACE successful
May 27 11:30:36 03[IKE] IDx' => 22 bytes @ 0xb6411f80
May 27 11:30:36 03[IKE]    0: 02 00 00 00 73 75 6E 2E 73 74 72 6F 6E 67 73 77  ....sun.strongsw
May 27 11:30:36 03[IKE]   16: 61 6E 2E 6F 72 67                              an.org
May 27 11:30:36 03[IKE] SK_p => 16 bytes @ 0x8186620
May 27 11:30:36 03[IKE]    0: 1A CD C2 CD B2 C7 E6 85 23 85 DA 5A BA 21 CE F4  ........#..Z.!..
May 27 11:30:36 03[IKE] octets = message + nonce + prf(Sk_px, IDx') => 490 bytes @ 0x8187580
May 27 11:30:36 03[IKE]    0: AB A9 AB C8 6E 45 34 84 F3 8D B0 E1 2E 00 99 7C  ....nE4........|
May 27 11:30:36 03[IKE]   16: 21 20 22 20 00 00 00 00 00 00 01 BA 22 00 00 28  ! " ........"..(
May 27 11:30:36 03[IKE]   32: 00 00 00 24 01 01 00 03 03 00 00 0C 01 00 00 14  ...$............
May 27 11:30:36 03[IKE]   48: 80 0E 01 00 03 00 00 08 02 00 00 04 00 00 00 08  ................
May 27 11:30:36 03[IKE]   64: 04 00 00 0E 28 00 01 08 00 0E 00 00 72 55 58 FD  ....(.......rUX.
May 27 11:30:36 03[IKE]   80: AB 76 1E ED 9A 52 3A 0A D2 4D C1 AA EC 8E 79 DD  .v...R:..M....y.
May 27 11:30:36 03[IKE]   96: B5 2D 2E D7 3C D3 71 18 79 8B C4 62 7E B2 56 89  .-..<.q.y..b~.V.
May 27 11:30:36 03[IKE]  112: A4 32 E1 01 56 4A AF 01 C9 D5 5A 43 21 0D BD C0  .2..VJ....ZC!...
May 27 11:30:36 03[IKE]  128: FF 6E 7A 0E 01 8F 88 55 5C F2 1D E9 0C 43 39 C5  .nz....U\....C9.
May 27 11:30:36 03[IKE]  144: 28 06 72 37 1A 0C FF 53 AB 4B 7D 29 42 93 33 61  (.r7...S.K})B.3a
May 27 11:30:36 03[IKE]  160: 1E F7 BD 67 4D ED 25 D4 DB 5D 77 3E 62 6B 62 4F  ...gM.%..]w>bkbO
May 27 11:30:36 03[IKE]  176: 8D 13 A8 87 62 A2 BB FA 11 F7 3F 39 7D C8 F2 80  ....b.....?9}...
May 27 11:30:36 03[IKE]  192: CF F4 46 78 DD 33 14 AD B6 D1 C1 76 9B B3 36 CA  ..Fx.3.....v..6.
May 27 11:30:36 03[IKE]  208: 5A 3F 79 09 27 9A FF E7 8C F5 D2 6E C4 65 7D CD  Z?y.'......n.e}.
May 27 11:30:36 03[IKE]  224: EC 29 64 CA 78 D9 DF 34 A6 BC 45 66 C1 B0 6B 08  .)d.x..4..Ef..k.
May 27 11:30:36 03[IKE]  240: 4D 86 90 A6 2C 36 5A 8D 33 84 0F 1D 41 C1 37 04  M...,6Z.3...A.7.
May 27 11:30:36 03[IKE]  256: 63 48 06 F4 E5 1B F8 07 54 E0 9B 50 C5 9B CB A9  cH......T..P....
May 27 11:30:36 03[IKE]  272: 07 23 22 06 50 D3 48 3E F9 79 E4 51 C3 25 48 D5  .#".P.H>.y.Q.%H.
May 27 11:30:36 03[IKE]  288: D5 1C 3A 82 31 0B 63 03 42 AE 28 C7 E1 20 2E 31  ..:.1.c.B.(.. .1
May 27 11:30:36 03[IKE]  304: 07 60 92 1C 3A EC 66 40 D6 83 7B B3 64 8C BF DD  .`..:.f@..{.d...
May 27 11:30:36 03[IKE]  320: 10 A3 CA 6A C0 4B 4D 0D 83 8A 48 63 29 00 00 24  ...j.KM...Hc)..$
May 27 11:30:36 03[IKE]  336: 4B 6D FC 5F 82 A0 0F 86 A6 8B D2 3F 6B 70 F4 DA  Km._.......?kp..
May 27 11:30:36 03[IKE]  352: 5C FA 3B 18 1B 15 C8 34 C4 9D 91 D6 84 D0 53 9C  \.;....4......S.
May 27 11:30:36 03[IKE]  368: 29 00 00 1C 00 00 40 04 58 6D 9B 83 DE 08 A0 15  ).....@.Xm......
May 27 11:30:36 03[IKE]  384: E4 F0 00 C5 D2 24 D8 4C B0 6F 65 B2 29 00 00 1C  .....$.L.oe.)...
May 27 11:30:36 03[IKE]  400: 00 00 40 05 74 50 F5 F3 86 F3 36 A0 65 2E 47 41  ..@.tP....6.e.GA
May 27 11:30:36 03[IKE]  416: 1F 25 16 40 66 3E 98 AE 29 00 00 08 00 00 40 14  .%.@f>..).....@.
May 27 11:30:36 03[IKE]  432: 00 00 00 0A 00 00 40 28 00 01 4B 6D FC 5F 82 A0  ......@(..Km._..
May 27 11:30:36 03[IKE]  448: 0F 86 A6 8B D2 3F 6B 70 F4 DA 5C FA 3B 18 1B 15  .....?kp..\.;...
May 27 11:30:36 03[IKE]  464: C8 34 C4 9D 91 D6 84 D0 53 9C 33 DE CC D3 36 11  .4......S.3...6.
May 27 11:30:36 03[IKE]  480: 6A AF 29 42 39 BE 4E 41 A0 CC                    j.)B9.NA..
May 27 11:30:36 03[IKE] PACESharedSecret => 256 bytes @ 0x8188570
May 27 11:30:36 03[IKE]    0: D7 02 2A 19 CA 84 89 0E 4B 4C 78 F4 51 23 0C 72  ..*.....KLx.Q#.r
May 27 11:30:36 03[IKE]   16: AC 92 B3 DF AC 3F 38 00 D5 C8 AA 62 63 E9 64 97  .....?8....bc.d.
May 27 11:30:36 03[IKE]   32: 08 58 26 39 0B 69 32 5A CA 1D E2 98 CF 34 B0 DA  .X&9.i2Z.....4..
May 27 11:30:36 03[IKE]   48: 1E A3 E7 CA 78 F6 A0 70 B7 8B 30 F6 01 6D A0 6D  ....x..p..0..m.m
May 27 11:30:36 03[IKE]   64: 3E 1F 32 4D 84 58 60 77 EA 01 8E 65 61 37 2A 2E  >.2M.X`w...ea7*.
May 27 11:30:36 03[IKE]   80: 99 5E 7B 79 48 F5 6C 0A 0F 42 B9 FA AA B5 C9 A4  .^{yH.l..B......
May 27 11:30:36 03[IKE]   96: FC A0 40 0D 26 7B 6B 9B 35 97 DB 11 29 37 CB 71  ..@.&{k.5...)7.q
May 27 11:30:36 03[IKE]  112: D8 65 42 78 32 68 DB 68 84 9B A3 A2 EB 7B 58 2C  .eBx2h.h.....{X,
May 27 11:30:36 03[IKE]  128: 8B 6D D1 30 40 95 BC F1 FF 24 AF 07 5C B1 31 EF  .m.0@....$..\.1.
May 27 11:30:36 03[IKE]  144: 86 3E FB 04 4F 0D 74 7E C1 57 AE B6 AA 00 75 4D  .>..O.t~.W....uM
May 27 11:30:36 03[IKE]  160: FE 5C 99 FF 78 AD C6 B0 A7 5B 23 72 39 02 03 F3  .\..x....[#r9...
May 27 11:30:36 03[IKE]  176: 75 59 81 8A 2A EF 51 19 56 3B DF EA 6E 52 B8 EA  uY..*.Q.V;..nR..
May 27 11:30:36 03[IKE]  192: 9D 8D 98 13 61 32 D8 C8 F8 94 13 BA 6B 8B D2 62  ....a2......k..b
May 27 11:30:36 03[IKE]  208: D7 F2 DE A7 39 59 7F 0B AA 1F 16 AA 26 9F F7 B8  ....9Y......&...
May 27 11:30:36 03[IKE]  224: F6 61 8A 89 7F B8 D9 22 2D AF D4 B8 BA 47 7E 0C  .a.....".-....G~.
```

```
May 27 11:30:36 03[IKE]  240: BF 83 1B 34 2C 9A 19 40 64 23 FD 56 58 CE 96 53  ...4,..@d#.VX..S
May 27 11:30:36 03[IKE] AUTHr = prf(prf(Ni | Nr, PACESharedSecret),<ResponderSignedOctets> | PKEi) => 16 bytes @ 0x81830f0
May 27 11:30:36 03[IKE]    0: 94 2E CD 6A F9 FB 89 88 BA 76 7A 15 52 E7 83 DB  ...j.....vz.R...
May 27 11:30:36 03[IKE] IKE_SA net-net[1] established between 10.10.10.20[sun.strongswan.org]...10.10.10.10[moon.strongswan.org]
May 27 11:30:36 03[IKE] IKE_SA net-net[1] state change: CONNECTING => ESTABLISHED
May 27 11:30:36 03[IKE] scheduling reauthentication in 3369s
May 27 11:30:36 03[IKE] maximum IKE_SA lifetime 3549s
May 27 11:30:36 03[IKE] CHILD_SA net-net{1} established with SPIs cc8d9aee_i c13a070b_o and TS 192.168.20.0/24 ===
192.168.10.0/24
May 27 11:30:36 03[ENC] generating IKE_AUTH response 2 [ AUTH SA TSi TSr N(AUTH_LFT) ]
May 27 11:30:36 03[NET] sending packet: from 10.10.10.20[500] to 10.10.10.10[500] (177 bytes)
```