SYT-RDP AUSARBEITUNG

IndInf/SysInt

Christoph Hackenberger

1 Inhaltsverzeichnis

2 Cloud Computing und IoT

2.1 Cloud Ansätze

2.1.1 Infrastructure as a Service (IaaS)

Infrastructure as a Service stellt die unterste Schicht des Cloud-Computings dar.
Hierbei handelt es sich um die Infrastruktur, die vom Cloud-Computing-Anbieter zur Verfügung gestellt wird. Den Benutzer des Dienstes steht es nun frei Software auf diese Infrastruktur zu spielen. Im Normalfall stellt der Anbieter hier ein virtualisiertes Betriebssystem zur Verfügung. Häufig kommen als Betriebssysteme Linux und Windows Server zum Einsatz. Der Anbieter kümmert sich auch um die Lastenverteilung und darum, dass die gekauften Leistungen ständig verfügbar sind. Auch das Backup der Systeme wird in den meisten Fällen vom IaaS-Anbieter übernommen.

Welche Aufgaben eine Infrastruktur hier erledigt, kann sehr vielfältig sein. Da der Benutzer Zugriff auf das Betriebssystem hat, kann er jegliche Software installieren. Mögliche Anwendungsfälle sind Domänencontroller, SQL Server, Internetserver, Mailserver und vieles mehr. Will man Webanwendungen auf der Infrastruktur erstellen, muss man sich um die Installation der notwendigen Laufzeitumgebung wie etwas PHP, Java oder .NET kümmern. Der Administrationsaufwand ist gegenüber SaaS und PaaS deutlich höher, dafür bietet IaaS die höchste Flexibilität. Anwendungen auf dieser Ebene können Sowohl an Kunden weiterverkauft, als auch für unternehmensinterne Zwecke genutzt werden.
Für die Wartung der virtualisierten Umgebung ist der Kunde jedoch selbst verantwortlich.

2.1.2 Platform as a Service (PaaS)

Im Gegensatz zu IaaS-Lösungen bieten Paas-Lösungen keinen Zugriff auf das darunterliegende Betriebssysteme, aber es bietet eine Sandbox mit High-Level APIs zur Anwendungsentwicklung.
Einer der Vorteile ist, dass sich hier nicht mehr um Softwarelizenzen, Installationen und Wartungen gekümmert werden muss, dadurch lassen sich einfach eigene Anwendungen entwickeln, ohne Rücksicht auf Hardware- und Softwareabhängigkeiten nehmen zu müssen.

In den meisten Fällen, bieten PaaS-Plattformen verschiedene APIs für Management der Plattformen, Diagnostik und Datenspeicherung. Platform-as-a-Service-Plattformen haben oft die Aufgabe die Erstellung und Bereitstellung von Webanwendungen zu ermöglichen. Es werden jedoch auch häufig andere Anwendungstypen wie Worker Roles oder Tasks bereitgestellt. Tasks dienen der Abarbeitung von Aufgaben, die in einem separaten Thread oder sogar auf einer eigenen VM ausgeführt werden. Das Ziel von Tasks ist es Webanwendungen von komplexen Aufgaben oder aufwändigeren Berechnungen zu befreien. Dies wiederum erhöht die Leistungsfähigkeit der Webanwendung. Eine Worker Role erfüllt einen ähnlichen Bereich, läuft aber im Normalfall ständig, während ein Task nur gestartet wir, wenn er benötigt wird.

Mit Hilfe der APIs lassen sich aber auch zum Beispiel Datenspeicher-Operationen ausführen. Dafür bieten viele Plattformen Queues, Tabellen und Blobs an. Meistens kommen hier keine relationalen Datenbanken zum Einsatz.

Die Zielgruppe von PaaS-Plattformen sind primär Softwareentwickler.

2.1.3 Software as a Service (SaaS)

Hierbei wird eine Software für den Endanwender im World Wide Web angeboten. Dabei kann es sich um verschiedene Typen von Software handeln. Als Beispiel können hier Office-Lösungen und E-Mail Seiten genannt werden.

Der Vorteil davon ist, dass im Gegensatz zu lokal installierten Anwendungen, SaaS-Anwendungen lediglich am Webserver aktualisiert werden müssen. Außerdem hat der Anwender immer sofort die neuste Version zur Verfügung, was wiederum den Vorteil mit sich bringt, dass Sicherheitslücken schneller geschlossen werden können. Und Dokumente auf einem System zu vergessen gehört somit der Vergangenheit an, was leider wiederrum auch die Abhängigkeit vom Internet verstärkt.
Die Anwendungen bieten jedoch meist keine Anpassungsmöglichkeiten und werden vom Hersteller als fertiges Produktverkauft.
Die Zielgruppe von SaaS-Lösungen sind also somit Endanwender.

2.1.4 Vergleich

	IaaS	**PaaS**	**SaaS**
Abstraktionsgrad	Sehr niedrig	mittel	Sehr hoch
Verwaltungsaufwand	Hoch	Mittel	Niedrig
Anpassbarkeit	Sehr hoch	Hoch	Sehr gering
Zielgruppe	Systemhäuser IT-Dienstleister IT-Abteilung Softwareentwickler	Softwareentwickler	Endanwender
Bezahlung	Pay per Use	Pay per Use	Pay per Use

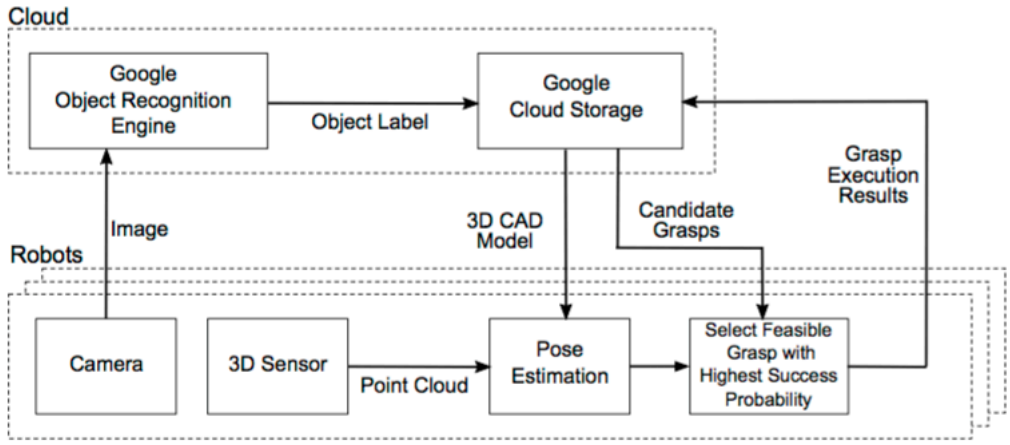
2.2 Cloud Robotics

2.2.1 Big Data

Die Cloud kann es autonomen Robtern ermöglichen auf eine große Menge von Daten zuzugreifen, welche nicht am Onboard-Memory des Roboters gespeichert werden könnten. Solche Daten können z.B. Bilder, Videos, Karten etc. sein. Neben den reinen Zugriff auf Daten können auch rechenintensive Berechnungen in die Cloud ausgelagert werden.
Ein Beispiel hierfür ist das Projekt Mobile Millennium von der Berkley University. In diesem Projekt werden Daten zur Verkehslage (Quellen: Taxis, Smartphones und andere Sensoren) gesammelt und in der Cloud verarbeitet. Dieses Projekt kann genutzt werden um z.B. zukünftige Selbstfahrende-Autos zu ermöglichen Staus auszuweichen.

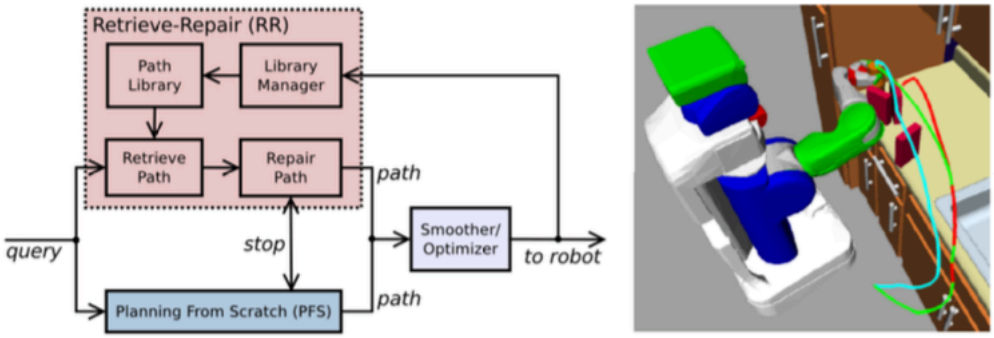
Auch im Bereich der Bildverarbeitung kann die Cloud einen großen Vorteil bringen. In der Robotik ist das Thema Greifen eine sehr komplizierte Aufgabe und hängt stark vom Objekt

ab, welches aufgehoben werden soll. Nutzt man nun eine Cloud welche, zu einem von Roboter aufgenommenen Bild des Objekts, das am besten passende CAD-Modell aus einer großen Datenbank auswählt, so kann der Roboter dann mithilfe dieses CAD-Modells entscheiden welcher Greifer am besten für die Aufgabe geeignet ist.



2.2.2 Kollektives Lernen

Die Cloud ermöglichtes es außerdem eine Art „soziales Netzwerk“ für Roboter aufzubauen. In der Cloud werden verschieden Ergebnisse (Start/End Bedingungen, Bewegungsbahnen, Performance Ergebnisse) gespeichert. Diese sind dann für den späteren Zugriff abrufbar. So kann durch Wiederholung und Verbesserung der Ergebnisse ein besserer Lösungsweg einer Aufgabe gefunden werden.



2.3 Projektumfeld

- Loxone

Mit Hilfe von Cloud Robotik können z.B. Pflegeroboter Gegenstände im Haushalt finden und zu den Patienten bringen. Als Ansatz für die Cloud würde sich PaaS eignen. Ein gutes Beispiel ist die Google Cloud Platform welche eine Cloud Vision API zur Verfügung stellt, welche man in seiner Andwendung benutzen kann.

3 Automatisierung, Regelung und Steuerung

3.1 Simultaneous Localization and Mapping (SLAM)

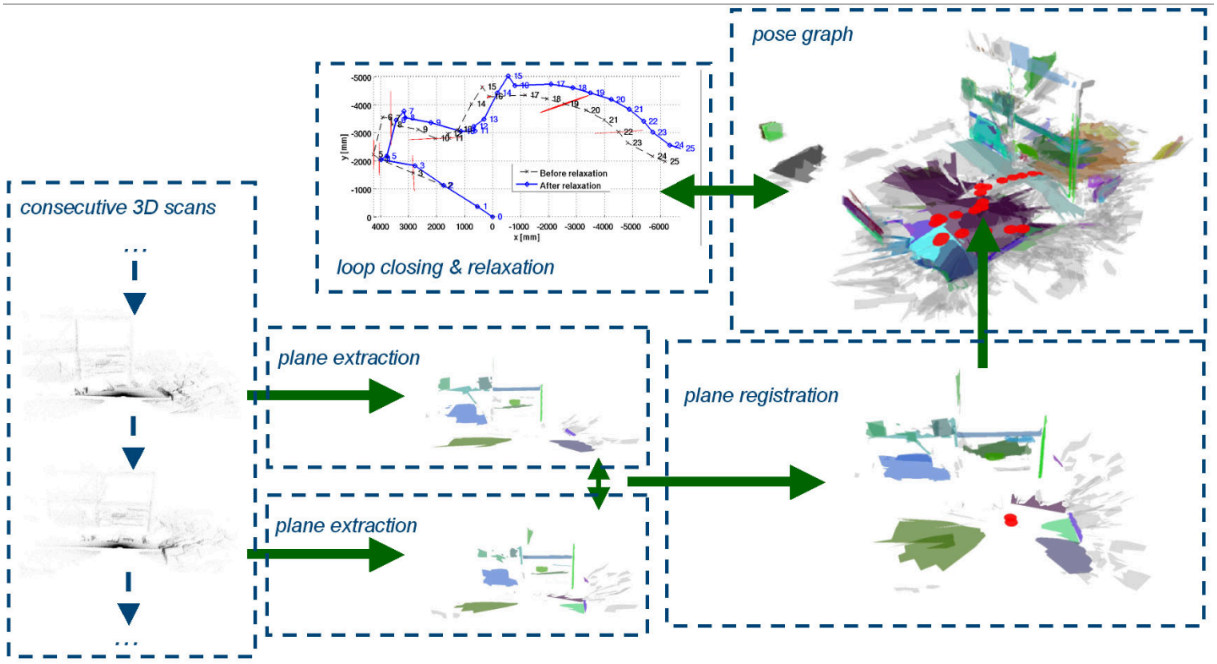
Als SLAM-Problem wird ein Problem der Robotik bezeichnet, bei dem ein mobiler Roboter gleichzeitig eine Karte seiner Umgebung erstellen und seine Posituon innerhalb dieser Karte abschätzen muss.

Eine der grundlegenden Fähigkeiten eines mobilen Roboters besteht darin, sich orientieren zu können, also zu wissen, wie seine Umgebung aussieht und wo er sich befindet. Ist eine Karte der Umgebung vorhanden, kann sich ein Roboter mit Hilfe seiner Sensoren, wie Ultraschall oder Lidar, darin positionieren. Ist die absolute Position des Roboters bekannt, kann eine Karte aufgebaut werden. Dabei misst der Roboter die relative Position möglicher Hindernisse zu ihm und kann mit seiner bekannten Position dann die absolute Position der Hindernisse bestimmen, die anschließend in die Karte eingetragen wird.

SLAM ist somit ein Henne-Ei-Problem, da weder die Karte noch die Position bekannt ist, sondern diese gleichzeitig geschätzt werden sollen.

Es gibt viele verschiedene Ansätze, wobei es grundlegende Ähnlichkeiten gibt. Da ein Roboter normalerweise nur einen Teil der Umgebung sehen kann, wird die Karte inkrementell aufgebaut: Zunächst ist keine Karte vorhanden und die Position des Roboters definiert den Ursprung seines Koordinatensystems. Damit ist trivialerweise die absolute Position des Roboters bekannt und die erste Messung der Umgebung kann direkt in die Karte eingetragen werden. Danach bewegt sich der Roboter und misst erneut seine Umgebung. Wenn sich der Roboter nicht zu weit bewegt hat, wird er einen Teil der schon bekannten Umgebung wiederum, aber auch einen bisher unbekannten Bereich zum ersten Mal vermessen. Aus der Überlappung der neuen Messung mit der bisherigen Karte kann die Bewegung des Roboters berechnet werden, so dass wieder die absolute Position bekannt ist und damit auch die neue Messung in die Karte integriert werden kann. In dieser Vorgehensweise wird die Karte inkrementell erweitert, bis das gesamte Gebiet vermessen ist.

Da die Bestimmung der Bewegung des Roboters zwischen zwei Messungen aber nie exakt ist, wird die berechnete Position des Roboters von der wahren immer weiter abweichen, womit auch die Qualität der Karte abnimmt. Damit die Karte trotzdem konsistent bleibt, muss der Algorithmus in der Lage sein zu erkennen, wenn ein schon bekannter Teil der Umgebung erneut vermessen wird



3.2 Robot Operating System (ROS)

3.2.1 Was ist ROS?

ROS ist eine Sammlung von Software Frameworks für die Entwicklung von Robotern. Es bietet Funktionalitäten für z.B. Hardware Abstraktion, Low-Level Device Control, Message-passing etc.
Das Ziel von ROS ist es wiederkehrende Alrgorithmen im Bereich der Robotik zur Verfügung zu stellen damit diese nicht für jedes Projekt neu implementiert werden müssen.

3.2.2 Aufbau

Die Basis von ROS bilden sogenannte Nodes. Diese Nodes führen die die Berechnung durch. Ein System besteht aus mehreren Nodes. Eine Node kann als Software Modul betrachtet werden. Diese Nodes kommunizieren untereinander in einem Peer-To-Peer Netzwerk. Für die Kommunikation werden sogennante Messages verwendet. Eine Message ist eine stark typisierte Datenstruktur (z.B. Integer, Boolean, Arrays, etc.). Eine Message kann auch aus einer anderen Message bestehen. Die Messages werden nun in sogenannten Topics veröffentlicht. Diese Topics werden mit einem einfachen String z.B. „Odometrie“ versehen. Jede Node die in die Daten einer anderen Node interessiert ist kann dann zu dem jeweiligen Topic subscriben. Jede Node kann mehrere Topics veröffentlichen und auch mehreren anderen Topics folgen. Es können auch Topics mit dem selben Namen veröffentlicht werden. Desweiteren gibt es noch die sogenannten Services, welche ebenfalls wie Topics durch einen String definiert werden. Zusätzlich hat ein Service allerdings noch eine Input und Output Message also Request und Response. Anders als bei Topics muss der String eines Service innerhalb des Systems einmalig sein (ähnlich einer URI). Ein Service kann also wie eine Art Web Service gesehen werden.

3.2.3 Vorteile von ROS

ROS bietet eine große Anzahl von Vorteile, hier sind nun einige aufgelistet.

- Debugging: Durch das System der Nodes, können diese einzeln debuggt werden. Außerdem bietet ROS die Möglichkeit die Nodes während der Laufzeit unterschiedlich miteinander zur verbinden.
- Logging: Durch die Message und Topic Architektur können die Message Streams einfach auf die Festplatte gedumpt werden und später wiedergegeben werden.
- Vordefinierte Subsystems: ROS bietet eine große Anzahl an vordefinierten Paketen von Subsystem welche oftmals „Out oft the Box“ funktionieren (z.B. Pfadplanung etc.)
- Kollaboration: ROS bietet die Möglichkeit eigene Packete mit Subsystems zu erstellen, diese können dann anderen Entwicklern zur Verfügung gestellt werden.

3.3 Projektumfeld

- Loxone

Mitfilfe von SLAM können Roboter sich selbst im Haus zurechtfinden (Es ist keine vorherige Konfiguration nötig).

4 Security, Safety und Availability

4.1 Verschlüsselung

Um nicht die eigentliche Nachricht zu übertragen, wendet man einen Schlüssel an, der aus der Nachricht einen sogenannten Chiffretext generiert. Der Empfänger dieses codierten Textes besitzt einen Dechiffrierschlüssel, um die Nachricht in ihren Ursprung zurück zu verwandeln. Dabei kann das Verhältnis zwischen den beiden benutzten Schlüsseln eine von zwei Ausprägungen annehmen, wir sprechen auch von sogenannten Verschlüsselungsarten. Der Standard wäre die symmetrische Verschlüsselung, wo das gleiche Geheimzeichen für das Ver- und Entschlüsseln benutzt wird. Ein Problem beim Austausch des symmetrischen Schlüssels ist, dass Mitlauschen anderer Teilnehmer.
Für dieses Problem gibt es eine Lösung, die sogenannte asymmetrische Verschlüsselung. Hier gibt es für das En- und Decoding eine eigene Chiffre. Wobei der Verschlüsselungs- key für jeden zugänglich ist aber nur der, der den Entschlüsselungkey besitzt, ist in der Lage, die Nachricht zu entschlüsseln.

4.1.1 Symetrische Verschlüsselung

Die Geschichte der symmetrischen Verschlüsselung reicht bis in die Antike. Damals wussten nur die Empfänger, nach welchem Verfahren die Botschaft verschlüsselt wurde. Cäsar zum Beispiel verschob jeden Buchstaben um 4 Stellen. Aus diesem Verschlüsselungsalgorithmus entstanden zum einen Blockchiffren und die Stromchiffren.

AES-Verschlüsselung

Der öffentliche Verschlüsselungsalgorithmus Advanced Encryption Standard (AES) ist eines der meistgenutzten und sichersten Verschlüsselungsverfahren. Es wird beispielsweise in den USA für Regierungsdokumente der höchsten Geheimhaltungsstufe verwendet. Seine Erfolgsgeschichte begann 1997, als ein Nachfolger für den in die Jahre gekommenen Verschlüsselungsstandard DES gesucht wurde. Vier Jahre dauerte die offiziell vom Standardinstitut NIST ausgerufene Suche. Am Ende konnte sich der Rijndael-Algorithmus – entwickelt von Daemen und Rijmen – gegen die Vielzahl von Bewerbern durchsetzen. Er überzeugte sowohl in Bezug auf Sicherheit als auch in Performance und Flexibilität. 2001 wurde er schließlich offiziell als der neue Standard AES bekanntgegeben.

Seine Funktionsweise beruht auf einer Reihe von Byteersetzungen (Substitutionen), Verwürfelung (Permutationen) und linearen Transformationen, die auf Datenblöcken von 16 Byte ausgeführt werden – daher die Bezeichnung Blockverschlüsselung. Diese Operationen werden mehrmals wiederholt, wobei in jeder dieser Runden ein individueller, aus dem Schlüssel berechneter Rundenschlüssel in die Berechnungen einfließt. Wird nur ein einziges Bit im Schlüssel oder im Datenblock verändert, entsteht ein komplett anderer Chiffreblock – ein Vorteil gegenüber klassischen Stromverschlüsselungen. Die Bezeichnungen AES-128, AES-192 und AES-256 spezifizieren dabei die Länge des Schlüssels: 128, 192 bzw. 256 Bit – eine drastische Verbesserung zur DES-Schlüssellänge von 56 Bit. Zum Vergleich: Einen Schlüssel der Länge 128 Bit mit einem modernen Supercomputer zu knacken, würde länger dauern als das angenommene Alter des Universums – und Boxcryptor nutzt eine Schlüssellänge von 256 Bit. Bis heute ist für keine der AES-Varianten ein praktisch durchführbarer Angriff bekannt. AES ist daher der bevorzugte Verschlüsselungsstandard für Regierungen, Banken und High-Security Systeme weltweit.

4.1.2 Asymetrische Verschlüsselung

RSA-Verschlüsselung

RSA ist eines der aktuell am meisten verbreiteten, asymmetrischen Verschlüsselungssysteme. Ursprünglich wurde es 1973 vom englischen Geheimdienst GCHQ entwickelt, aber dann unter die höchste Geheimhaltungsstufe gestellt. Seine zivile Wiederentdeckung verdankt das Verschlüsselungsverfahren den Kryptologen Rivest, Shamir und Adleman (daher auch die Abkürzung RSA), die 1977 während der Analyse eines anderen kryptographischen Problems auf es stießen.

Im Gegensatz zu klassischen, symmetrischen Verschlüsselungsverfahren arbeitet RSA mit zwei Schlüsseln: einem öffentlichen und einem privaten Schlüssel. Zum Entschlüsseln einer Nachricht benötigt man dann jeweils das Gegenstück des zum Verschlüsseln genutzten Schlüssels. Der öffentliche Schlüssel ist in aller Regel allgemein zugänglich, da eine Berechnung des privaten Schlüssels aus ihm nicht möglich ist.

Um eine Nachricht verschlüsselt zu übertragen, verschlüsselt der Sender die Nachricht mit den Öffentlichen Schlüssel des Empfängers. Nach dem der Empfänger die verschlüsselte Nachricht erhalten hat, kann er diese mit seinem eigenen privaten Schlüssel wieder entschlüsseln.

Die Sicherheit von RSA basiert dabei auf dem mathematischen Problem, große Zahlen in ihre Primfaktoren zu zerlegen. Eine zu verschlüsselnde Nachricht wird vom Algorithmus als eine einzige, große Zahl angesehen. Beim Verschlüsseln wird die Nachricht mit dem Schlüssel potenziert, und mit Rest durch ein festgelegtes Produkt zweier Primzahlen geteilt. Wiederholt man diesen Vorgang mit dem Gegenschlüssel, erhält man den Klartext zurück. Die beste bekannte Methode, die Verschlüsselung zu brechen, ist die Berechnung der Primfaktoren des Teilers. Es ist aktuell jedoch nicht möglich, diese Faktoren für Zahlen größer als 768 Bit zu berechnen. Moderne Sicherheitssysteme setzten aufgrund dessen auf Schlüssel mit einer Mindestlänge von 3072 Bit.

Oftmals wird zur Verschlüsselung von Nachrichten eine symmetrische Verschlüsselung gewählt da diese deutlich schneller sind als asymmetrische. Der Schlüssel der symmetrischen Verschlüsselung wird dann aber asymmetrisch Verschlüsselt.

RSA-Signatur

Bei der RSA-Signatur wird der Hash der Nachricht mit dem eigenen privaten Schlüssel des Senders verschlüsselt. Der Empfänger bildet nun selbst den Hash der Nachricht und entschlüsselt den empfangenen Hash mit dem öffentlichen Schlüssel des Senders. Stimmen nun die beiden Hash-Werte überein so kommt die Nachricht vom „richtigen“ Sender.

4.2 Netzwerksicherheit

4.2.1 Honeypot

Ein Honigtopf oder englisch honeypot ist eine Einrichtung, einen einen Angreifer vom eigentlichen Ziel ablenken soll und ihn in einen Bereich hineinziehen soll, der ihn sonst nicht interessiert hätte. Der Name stammt aus der Natur wo man Bären oft mit einem Honigtopf oder ablenken oder sogar in die Falle locken kann. Ein Honeypot ist also konkret ein Server der bestimmte Netzwerkdienste eines Rechnernetzes oder einfach das Verhalten eines Users simuliert. Honeypots werden vorrangig dazu eingesetzt, um Informationen über das Angriffsmuster und das Angreiferverhalten zu erhalten. Erfolgt durch den Angreifer ein Zugriff auf so einen Honey Pot, werden alle damit verbundenen Aktionen protokolliert und ggf. ein Alarm ausgelöst. Das wirklich wichtige reale Netzwerk bleibt vom Angriff möglichst verschont, da es besser gesichert ist als der Honeypot.

Die Idee hinter dem Einsatz von Honeypot-Systemen ist in einem Netzwerk einen oder am besten mehrere Honeypots zu installieren, die keine vom Anwender oder anderen Kommunikationspartnern benötigten Dienste bieten und so im Normalfall niemals angesprochen werden. Ein Angreifer der jetzt aber nicht zwischen einem realen Server und einem Honeypot unterscheiden kann und routinemäßig das Netz auf Schwachstellen untersucht wird natürlich den schlechter gesicherten Honeypot als Angriffsziel bevorzugen. Der Zugriff wird protokolliert und da ein Honeypot ein ungenutztes System ist, wird jeder Zugriff darauf als potenzieller Angriff gewertet.

Zudem gibt es Honeypots die Anwender simulieren (Honeyclients). Diese nutzen normale Webbrowser etc. und Angriffe auf den Browser oder Browser-Plug-Ins zu erkennen. Mehrere

Honeypots bilden ein zusammengeschlossenes Netz (Honeynet). Ein physischer Honeypot stellt einen realen Rechner im Netzwerk mit eigener Netzwerkadresse dar. Ein virtueller Honeypot, jedoch, ist ein logisch eigenständiges System, welches durch einen anderen Rechner simuliert wird.

4.2.2 Application Firewall

Eine Application Firewall ist eine spezielle Art einer Firewall die input, output und/oder Zugriff zu oder von einer Applikation oder eines Dienstes kontrolliert. Eine Application Firewall arbeitet indem sie den input, output oder Zugriffe auf Systemdienste protokolliert und diese gegebenenfalls blockiert falls ein Verstoß gegen die Firewall-Policy vorliegt. Die Firewall ist dafür ausgerichtet den ganzen Netzwerkverkehr (bis zum application layer) zu kontrollieren. (Im Gegensatz zu einer stateful network firewall die ohne zusätzliche Software keinen Netzwerkverkehr einer bestimmten application kontrollieren kann) Es werden zwei primäre Kategorien von Application Firewalls unterschieden.

Host-based Application Firewall

Eine host-based Application Firewall ist in der Lage input, output und/oder System-dienstzugriffe an oder von einer Application zu protokollieren. Das passiert durch die Analyse der Informationen eines System-calls(und/oder zusätzlich zu einem network stack). Ein wichtiges Merkmal der host-based Application Firewall ist dass sie nur eine Sichehreit für Applications die auf demselben Host laufen bieten kann.

Die HBAFs bestimmen ob ein Prozess eine Verbindung akzeptieren sollte oder nicht. Sie benutzen dabei socket calls um die Verbindungen zwischen dem Application layer und den unteren layers des OSI-Modells herauszufiltern. Diese Firewalls arbeiten ähnlich wie Paketfilter, jedoch geben Application Firewalls bestimmte Filterregeln (erlauben/blockieren) auf Prozessbasis vor, wobei bei Paketfiltern dies auf Portbasis passiert. Allgemein werden prompts für die Definition dieser Regeln für Prozesse benutzt. Application Firewalls arbeiten außerdem sehr oft mit Paketfiltern zusammen.

Application Firewalls filtern zusätzlich Verbindungen durch das Überprüfen der Prozess-IDs von Datenpaketen und den Regeln für die lokalen Prozesse die in der Datenübertragung involviert sind. Der Umfang der Filterung wird wie schon beschrieben vom definierten ruleset bestimmt. Mit einer großen Vielzahl an Zusatzsoftware können application firewalls inzwischen schon sehr komplexe rulesets haben. Ein Nachteil dieser Prozessrulesets ist dass ihre Effektivität im Filtern von jeder möglichen Verbindung mit anderen Prozessen sehr begrenzt ist. Zudem kann so ein ruleset nicht die Modifikation eines Prozesses (z.B. durch memory corruption exploits) verhindern.

Network-bases Application Firewalls

Eine Network-based Application Firewall ist eine Firewall die auf dem Application layer eines Protokoll-Stacks arbeiten und wird auch Proxy-based oder Reverse-Proxy Firewall genannt. NBAFs konzentrieren sich auf eine bestimmte Art von Netzwerkverkehr abhängig vom Dienst.(wie zum Beispiel eine Web Application Firewall)

Die Implementierung einer solchen network-based Application Firewall kann durch eine Software die einfach am Host läuft, oder durch ein eigenständiges Netzwerkgerät realisiert werden. Oft ist es einfach ein Host der verschiedene Arten von Proxy-Servern benutzt um den Netzwerkverkehr abzufangen bevor dieser an den Client oder Server weitergeht.
Weil eine solche Firewall auf dem Application layer arbeitet ist eine Analyse des Inhalts des Datenverkehrs durchaus möglich wodurch bestimmte Inhalte wie zum Beispiel Webseiten oder Viren geblockt werden können.

5 Authentication, Authorization, Accounting

5.1 Authentisierung

Unter Authentisierung versteht man den Nachweis der eigenen Identität. Dabei kann es sich um die Identität einer Person (eines Anwenders) oder auch um die eines Computerprogramms handeln. Wird dagegen eine angegebene Identität überprüft, so spricht man von Authentifizierung. Auch hier kann es um die Identität eines Menschen oder eines Programms gehen.
Ein Beispiel: Ein Anwender, der sich an einem Computer anmeldet, gibt dazu seinen Nutzernamen und zusätzliche Informationen (typischerweise ein Passwort) an. Damit authentisiert er sich gegenüber dem Anmeldeprogramm und authentifiziert den Anwender.

Die eigene Identität zu belegen oder die Identität eines anderen zu überprüfen, stellt eine einfache Aufgabe dar, wenn es sich bei den Beteiligten um Menschen Handelt. Denn diese können sich anhand von Merkmalen wie ihrem Aussehen oder dem Klang ihrer Stimme erkennen. Darüber hinaus besitzen Menschen weitere eindeutige Merkmale wie den Fingerabdruck oder die Unterschrift. Solche Merkmale können ebenfalls für die Identitätsprüfung herangezogen werden, wenn Aussehen oder Stimme unbekannt sind. Auch wenn alle diese Merkmale unbekannt sein sollten, so können sich zwei Menschen immer noch anhand eines gemeinsamen Geheimnisses (Shares Secret) gegenseitig ihre Identität nachweisen. Solch ein gemeinsames Geheimnis kann bei zwei Personen beispielsweise ein Losungs- bzw. Passwort sein.

Computer besitzen zunächst keine vergleichbaren Merkmale, die sie derartig einmalig machen. Außerdem haben sie nur eingeschränkte Fähigkeiten, die oben beschriebenen Merkmale wie Aussehen oder Stimme auszuwerten. Diese Unfähigkeit gleichen sie wiederum dadurch aus, dass sie sich wesentlich längere und damit auch sicherere Geheimnisse merken können als Menschen. Anstelle von Codewörtern benutzen Computer für den Zweck sehr lange kryptographische Schlüssel (Keys). Dabei handelt es sich um möglichst zufällig gewählte Folgen von Nullen und Einsen einer bestimmten Länge. Eine Schlüssellänge von 256 Bit ist heutzutage für symmetrische Kryptografie üblich.

Unsere menschlichen Gehirne sind kaum in der Lage, sich solche kryptografischen Schlüssel zu merken. Wir müssen und daher mit Passwörtern behelfen. Computer können aus diesen Passwörtern wieder um kryptografische Schlüssel erzeugen, mit denen sie besser umgehen können. Natürlich haben solche passwortbasierten Schlüssel lediglich die gleiche Sicherheit wie die zugrundeliegenden Passwörter.

5.1.1 Authentisierung durch Wissen

Hier authentisiert sich der Benutzer durch vorgetragenes Wissen. Beispiele sind ein Passwort oder eine persönliche Identifikationsnummer (PIN). Diese Merkmale haben mehrere Nachteile: So eignen sie sich beispielsweise nicht für sehr vergessliche Menschen und können außerdem ausspioniert werden, auch ohne dass das Opfer das merkt.

5.1.2 Authentisierung durch Besitz

Diese Art der Authentifikation basiert auf dem Besitz einer fälschungssicheren Marke, die Informationen in mechanischer, magnetischer oder elektrischer Form enthält. Dazu zählen Dinge wie der Personalausweis, die EC-Karte oder auch die Transaktionsnummern beim Internet-Banking (TAN-Listen). Im Computerumfeld zählen hierzu Token-Karten, SSL-Zertifikate, Smartcards oder auch Listen von Einmalpasswörtern. Der Nachteil hierbei ist, dass man sie entwenden kann, wobei man hier im Computerumfeld diesem Manko durch eine sehr kurze Gültigkeit dagegen steuern kann.

5.1.3 Authentisierung durch biometrische Merkmale

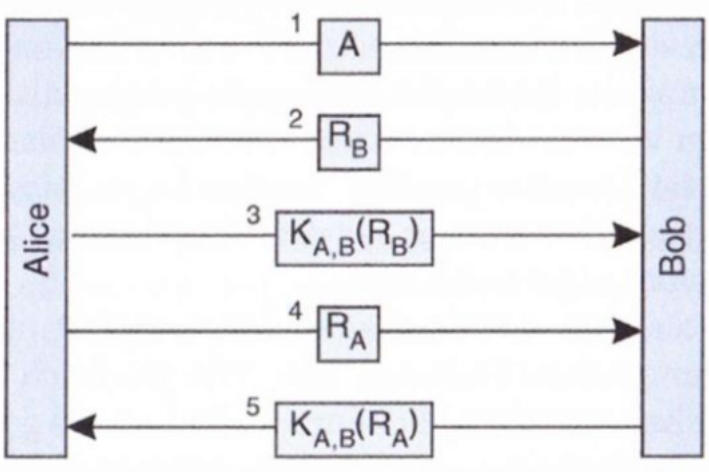
Die biometrischen Merkmale gehören eigentlich auch zu den Dingen, die man besitzt. Sie haben aber den Vorteil, dass sie nicht so leicht zu entwenden sind, Zu diesen Merkmalen zählt man z.B. Aussehen, Stimme, Fingerabdruck, Augenhintergrund oder Unterschrift. Der Nachteil der biometrischen Merkmale liegt darin, dass Computer diese nicht besonders gut erkennen und verarbeiten können und dass die Authentisierung mittels dieser Merkmale daher fehleranfällig ist.

5.2 Authentifizierung

Nehmen wir an, Alice ergreift die Initiative und richtet gemeinsam mit Bob einen Kanal ein. Ist der Kanal einmal eingerichtet worden, kann sich Alice sicher sein, dass sie mit Bob redet und vice versa.
Um in der Folge die Integrität der Datennachrichten sicherzustellen, die nach der Authentifizierung ausgetauscht werden, ist es gängige Praxis, Kryptografie mit geheimen Schlüsseln zu verwenden, indem zufällige Schlüssel (Sitzungsschlüssel) generiert werden. Ein Sitzungsschlüssel ist ein gemeinsamer (geheimer) Schlüssel, der aus Gründen der Integrität und möglicherweise auch der Vertraulichkeit zur Verschlüsselung von Nachrichten verwendet wird. Ein derartiger Schlüssel wird im Allgemeinen nur benutzt solange es den Kanal gibt. Bei der Schließung des Kanals wird sein zugehöriger Schlüssel verworfen (bzw. auf sichere Art zerstört). Wir kommen später auf Sitzungsschlüssel zurück.

5.2.1 Gemeinsamer geheimer Schlüssel

Betrachten wir das Authentifizierungsprotokoll auf der Grundlage eines geheimen Schlüssels, den Alice und Bob bereits gemeinsam nutzen. Das Protokoll verfolgt einen gemeinsamen Ansatz, bei dem eine Seite die andere zu einer Antwort auffordert, die nur korrekt sein kann, wenn die andere Seite den gemeinsamen Schlüssel kennt. Solche Lösungen sind auch als Challenge-Response-Verfahren bekannt.

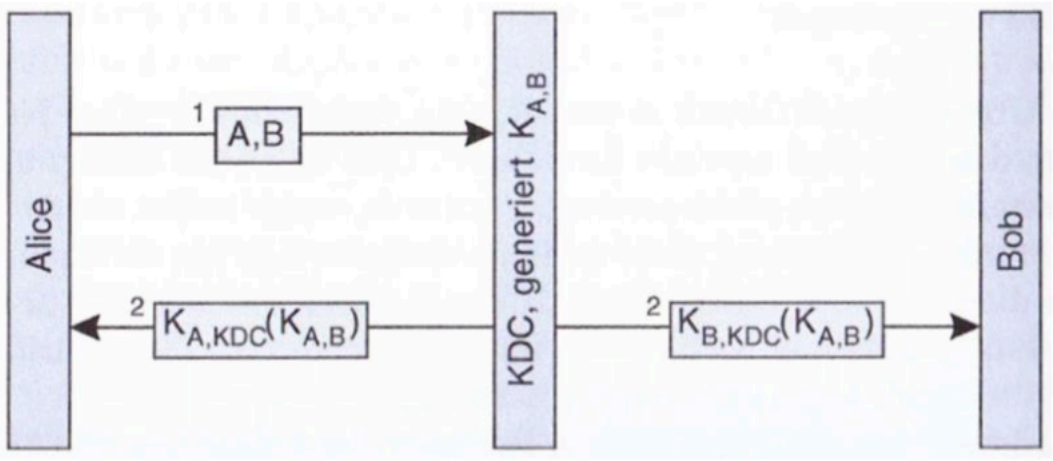


Im Falle der Authentifizierung auf der Grundlage eines gemeinsamen geheimen Schlüssels geht das Protokoll wie in der obigen Abbildung vor. Zunächst sendet Alice Bob ihre Identität (Nachricht 1), womit sie anzeigt, dass sie einen Kommunikationskanal zwischen beiden einrichten möchte. Bob sendet Alice nachfolgend eine Aufgabe RB, dargestellt als Nachricht 2. Eine solche Aufgabe kann in Form einer Zufallszahl erfolgen. Alice wird aufgefordert, diese Aufgabe mit dem geheimen Schlüssel KA,B zu verschlüsseln, den sie mit Bob gemeinsam nutzt, und Bob die verschlüsselte Aufgabe zurückzusenden. Wenn Bob die Antwort KA,B (RB) auf seine Aufgabe RB erhält, kann er die Nachricht erneut unter Verwendung des gemeinsamen Schlüssels entschlüsseln, um zu sehen, ob die RB enthält. Falls ja, dann weiß er, dass sich Alice auf der anderen Seite befindet, denn wer sonst hätte RB mit KA,B verschlüsseln können? Bob hat also nun überprüft, dass er wirklich mit Alice redet. Beachten Sie aber, dass Alice noch nicht überprüft hat, dass wirklich Bob am anderen Ende des Kanals ist. Daher sendet sie eine Aufgabe RA (Nachricht 4), auf die Bob mit KA,B (RA) antwortet, was als Nachricht 5 dargestellt wird. Wenn Alice diese mit KA,B entschlüsselt und ihre RA sieht, weiß sie, dass sie mit Bob redet.

5.2.2 Key Distribution Center (KDC)

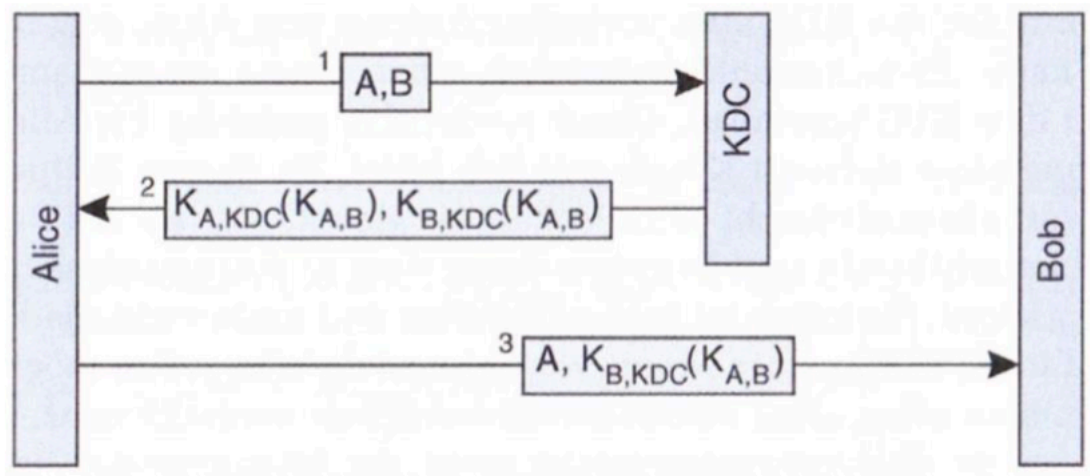
Eines der Probleme bei der Verwendung eines gemeinsamen geheimen Schlüssels zur Authentifizierung ist die Skalierbarkeit. Wenn ein verteiltes System N Hosts enthält und jeder Host gemeinsam mit jedem der anderen N-1 Hosts einen geheimen Schlüssel nutzen muss, erfordert dies, dass das System als als Ganzes N (N-1)/2 und jeder Host N-1 Schlüssel verwalten muss. Bei großen N führt das zu Problemen. Eine Alternative ist ein zentralisierter Ansatz wie ein Key Distribution Center (KDC). Dieses KDC nutzt gemeinsam mit jedem der Hosts einen geheimen Schlüssel, aber die Hosts untereinander benötigen nicht mehr paarweise geheime Schlüssel. Die Verwendung eines KDC macht es also erforderlich, dass wir statt N (N-1)/2 nur N Schlüssel verwalten müssen, was offensichtlich eine Verbesserung darstellt.

Wenn Alice einen sicheren Kanal mit Bob einrichten will, kann sie das mithilfe eines (zuverlässigen) KDC tun. Die Idee dabei ist, dass das KDC einen Schlüssel sowohl an Alice als auch an Bob ausgibt, den sie dann zur Kommunikation nutzen können.



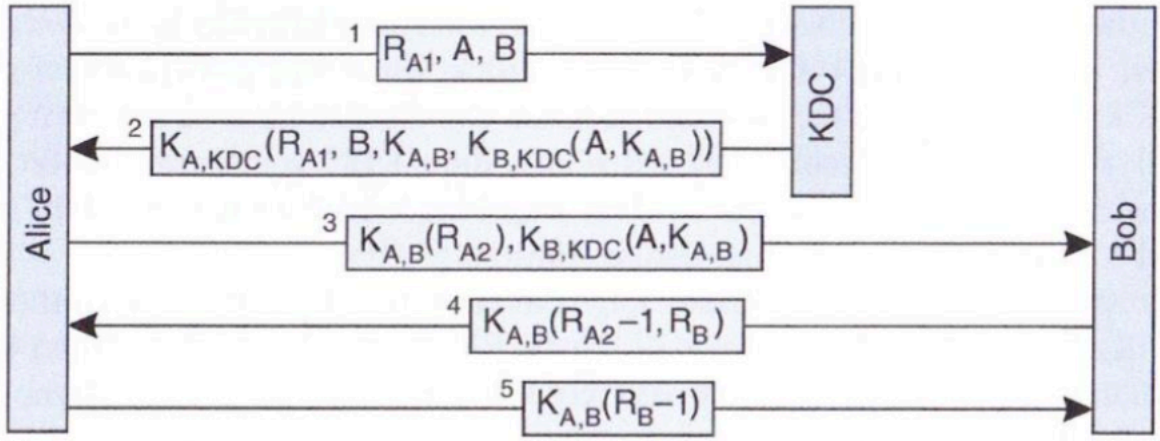
Alice sendet zunächst eine Nachricht an das KDC mit der Mitteilung, dass sie mit Bob reden will. Das KDC sendet eine Nachricht zurück, die einen gemeinsamen geheimen Schlüssel KA,B enthält, dem sie verwenden kann. Die Nachricht wird mit dem geheimen Schlüssel KA,KDC verschlüsselt, den Alice mit dem KDC gemeinsam nutzt. Darüber hinaus sendet das KDC KA,B auch an Bob, nun aber verschlüsselt mit dem geheimen Schlüssel KB,KDC den es mit Bob gemeinsam nutzt.

Der Hauptnachteil dieses Ansatzes liegt darin, dass Alice möglicherweise einen Kanal mit Bob einrichten will, bevor der überhaupt den gemeinsamen geheimen Schlüssel vom KDC erhalten hat. Zudem muss das KDC Bob zum Handeln auffordern, indem es ihm den Schlüssel weitergibt. Diese Probleme lassen sich umgehen indem man Bob's Nachricht ebenfalls an Alice sendet und ihr somit überlässt mit Bob in Kontakt zu treten. Bob's Nachricht nennt man in diesem Fall auch Ticket. Es ist die Aufgabe von Alice dieses Ticket an Bob weiterzuleiten. Bob ist auch der einzige der das Ticket (also die für ihn bestimmte Nachricht) entschlüsseln kann, da nur er neben dem KDC weiß wie das Ticket zu entschlüsseln ist.

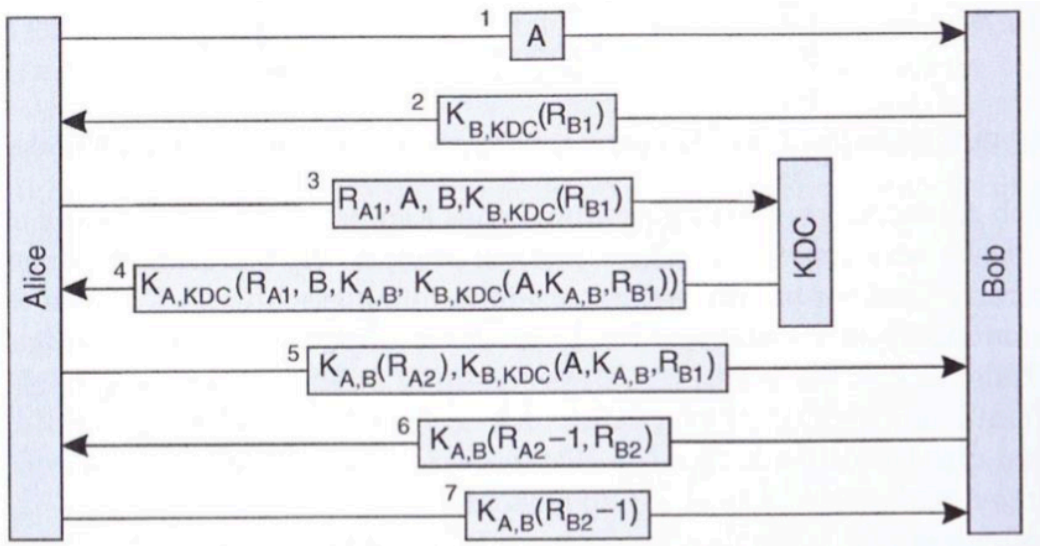


Das in der Abbildung dargestellte Protokoll ist eine Variante des Needham-Schroeder-Authentifizierungsprotokolls unter Verwendung eines KDC.

Das Needham-Schroeder-Protokoll ist ein Mehrweg-Challenge-Response-Verfahren. Es funktioniert folgendermaßen: Wenn Alice einen sicheren Kanal mit Bob einrichten will, sendet sie eine Anfrage an das KDC mit einer Aufgabe RA und ihrer Identität A und natürlich der von Bob. Das KDC antwortet ihr mit der Übersendung des Tickets KB,KDC (KA,B) zusammen mit dem geheimen Schlüssel KA,B, den sie nachfolgend mit Bob gemeinsam nutzen kann. Die Aufgabe RA1, die Alice zusammen mit ihrer Anfrage, einen Kanal zu Bob einzurichten, an das KDC sendet, wird auch als Nonce bezeichnet. Eine Nonce ist eine Zufallszahl, die nur einmal verwendet wird, zum Beispiel eine aus einer sehr großen Menge gewählte. Der Hauptzweck einer Nonce ist, zwei Nachrichten eindeutig miteinander in Beziehung zu setzen, in diesem Fall Nachricht 1 und Nachricht 2. Insbesondere dadurch, dass RA1 wieder in Nachricht 2 enthalten ist, ist sich Alice sicher, dass Nachricht 2 in Antwort auf Nachricht 1 gesendet worden ist und dass es sich zum Beispiel nicht um die Antwort auf eine ältere Nachricht handelt. Somit werden derartige Angriffe ausgeschlossen, da das erneute Einspielen einer alten Nachricht sofort entdeckt wird.



Das Needham-Schroeder-Protokoll, wie wir es hier dargestellt haben, hat jedoch die Schwäche, dass Chuck Nachricht 3 wieder einspielen und Bob dazu bewegen könnte, einen Kanal einzurichten, wenn er je Zugriff auf einen alten Schlüssel KA,B erlangen würde. Bob würde dann glauben, dass er mit Alice spricht. Dementsprechend muss man Nachricht 3 zu Nachricht 1 in Beziehung setzen, d.h. den Schlüssel von der ursprünglichen Anfrage von Alice, einen Kanal mit Bob einzurichten, abhängig machen.

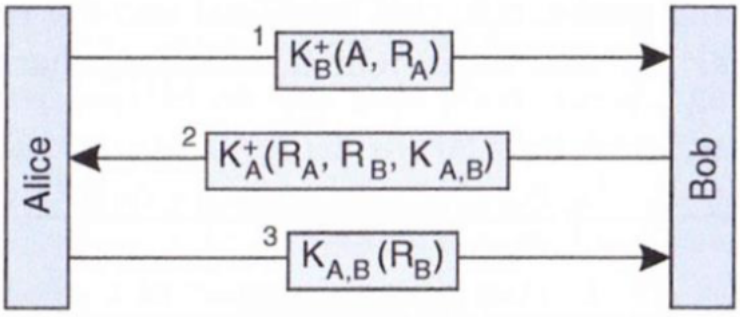


Der Trick besteht darin, eine Nonce in der von Alice an das KDC gesandten Anfrage einzuschließen. Diese Nonce muss jedoch von Bob stammen: Das überzeugt Bob, dass wer auch immer es sei, der einen sicheren Kanal mit ihm einrichten möchte, die entsprechenden Informationen vom KDC erhalten hat. Daher bittet Alice zunächst Bob, ihr eine Nonce RB1 zu senden, verschlüsselt mit dem Schlüssel, den Bob gemeinsam mit dem KDC nutzt. Alice schließt diese Nonce in ihrer Anfrage an das KDC ein, das diese dann entschlüsselt und das Ergebnis in das erzeugte Ticket einfügt. Auf diese Art und Weise ist Bob sicher, dass der Sitzungsschlüssel an die ursprüngliche Anfrage von Alice, mit Bob zu reden, gebunden ist.

5.2.3 Öffentlicher und Privater Schlüssel

Hierbei brauchen die kommunizierenden Parteien keinen gemeinsamen Schlüssel zu kennen. Ein Benutzer erzeugt ein Schlüsselpaar, welches aus einem öffentlichen und einem privaten Schlüssel besteht. Der öffentliche dient zur Verschlüsselung und der private zur Entschlüsselung.

Nehmen wir an, dass Alice einen sicheren Kanal zu Bob einrichten will und beide im Besitz des öffentlichen Schlüssels des anderen sind. Alice beginnt, indem sie Bob eine Aufgabe RA sendet, die sie mit seinem öffentlichen Schlüssel KB+ verschlüsselt. Es ist Bobs Aufgabe, die Nachricht zu entschlüsseln und die Aufgabe an Alice zurückzusenden. Da Bob der einzige ist, der die Nachricht (unter Verwendung des von Alice benutzten privaten Schlüssels) entschlüsseln kann, weiß Alice, dass sie es mit Bob zu tun hat.



Wenn Bob Alices Anfrage zur Einrichtung eines Kanals erhält, sendet er die entschlüsselte Aufgabe zusammen mit seiner eigenen Aufgabe RB zur Authentifizierung von Alice zurück. Darüber hinaus erzeugt er einen Sitzungsschlüssel KA,B, der zur weiteren Kommunikation genutzt werden kann. Bobs Antwort auf Alices Aufgabe, seine eigene Aufgabe und der Sitzungsschlüssel werden in eine Nachricht gepackt, die mit dem öffentlichen Schlüssel von Alice KA+ verschlüsselt wird. Nur Alice ist in der Lage, diese Nachricht mit dem zu KA+ zugehörigen privaten Schlüssel KA- zu entschlüsseln. Schließlich sendet Alice ihre Antwort auf Bobs Aufgabe unter Verwendung des von Bob erzeugten Sitzungsschlüssels KA,B an Bob zurück. Auf diese Art beweist sie, dass die Nachricht 2 entschlüsseln konnte und dass es somit tatsächlich Alice ist, mit der Bob spricht.

6 Desaster Recovery

6.1 Fehlertolerante Architekturen

6.1.1 Cold Standby

Bei Cold-Standby wird neben den eigentlichen Produktivsystem ein zweites identisches System aufgebaut. Dieses ist allerdings solange das Produktivsystem funktioniert nicht im Einsatz und bleibt deaktiviert. Sollte das Produktivsystem ausfallen, muss das Ersatzsystem manuell hochgefahren werden.
Vorteile:

- Keine Komplexitätserhöhung
- Geringe Kosten

Nachteile:

- Ersatzsystem muss ständig aktuell gehalten werden
- Ersatzsystem muss manuell aktiviert werden
- Daten müssen ggf. migriert werden

6.1.2 Hot Standby

Beim Hot Standby steht ebenfalls neben dem Produktivsystem ein Ersatzsystem zur Verfügung. Das Ersatzsystem wird allerdings neben dem Produktivsystem parallel betrieben. Beim Ausfall des Produktivsystems wird das Ersatzsystem aktiviert. Der Wechsel auf das Ersatzsystem kann ggf. auch automatisch stattfinden, dafür sind allerdings zusätzliche Funktionen, wie z.B. die automatische Erkennung von Ausfällen, notwendig.
Vorteile:

- Ausfallzeiten geringer als bei Cold-Standby
- Relativ Kostengünstig
- Ersatzsystem in Betrieb, daher Datenreplikation möglich

Nachteile:

- Ersatzsystem muss ständig aktuell gehalten werden

Ggf. manuelle Aktivierung notwendig

6.1.3 Cluster

Durch die Gruppierung von mehreren Rechnern zu einem Cluster, welche parallel arbeiten, führt zu einer Steigerung der Verfügbarkeit sowie der Performance. Die Applikation kann dabei aktiv auf einem Rechner oder parallel verteilt auf mehre Rechner ausgeführt werden. Man unterscheidet dabei je nach Funktionsart zwischen Load balanced Cluster und Failover Cluster.

Load balanced Cluster

Beim Load balanced Cluster wird die Applikation bzw. die Instanzen einer Applikation auf mehrere Server verteilt. Die Verteilung basiert meist auf der Auslastung der einzelnen Server. Durch diese Lastverteilung wird eine Performance-Steigerung erzielt. Außerdem

kann durch die geschaffene Redundanz ein Ausfall eines einzelnen Servers kompensiert werden.

Vorteile:

- Steigerung der Verfügbarkeit und Performance
- Alle Ressourcen werden dauerhaft genutzt
- Skalierbar
- Komplexität geringer als Failover Cluster

Nachteile:

- Nicht für alle Applikation einsetzbar

Failover Cluster

Bei einem Failover-Cluster, wird beim Ausfall eines der Cluster-Systeme, ein automatischer Wechsel des Betriebes auf einen anderen Teil des Clusters durchgeführt. Diese automatische Übernahme wird Failover genannt. Für diese Failover-Funktionalität, wird eine sogenannte „heartbeat“ Verbindung zwischen den Cluster-Servern benötigt, welche für die Kommunikation innerhalb des Clusters verwendet wird.
Vorteile:

- Erhebliche Steigerung der Verfügbarkeit
- Voll Automatisiert

Nachteile:

- Hoch komplex
- Nicht gut skalierbar
- Nur Teile der Ressourcen werden genutzt
- Hohe Kosten

7 Algorithmen und Protokolle

7.1 Pfadoptimierung

7.1.1 Dijkstra-Algorithmus

Oft möchte man wissen, welches der kürzeste Weg von a nach b ist. Dazu kann man Orte als Knoten in einem Graphen darstellen, welche durch Kanten verbunden sind. Diese Kanten stellen die Wege zwischen den Orten dar. Sie können Beschriftungen haben wie Weglängen in Metern oder Kilometern, aber auch Fahrzeiten oder Mautgebühren für den Weg.

Die Kantenbeschriftungen werden im Allgemeinen als Kantenkosten oder -gewichte bezeichnet.

Mit dem Algorithmus von Dijkstra kann man kürzeste beziehungsweise kostengünstigste Wege berechnen, sofern alle Kosten positive Zahlen sind. Falls auch negative Zahlen auftreten, funktioniert der Algorithmus nicht mehr.

er Dijkstra-Algorithmus berechnet die Kosten der günstigsten Wege von einem Startknoten aus zu allen anderen Knoten im Graph.

Der Algorithmus beginnt bei einem Startknoten und wählt schrittweise über die als nächstes erreichbaren Knoten die momentan günstigsten Wege aus. Dabei kann er auch Verbesserungen vornehmen. Das tut er so lange, bis alle Knoten besucht wurden und kein besserer Weg mehr gefunden wurde.

Um den Algorithmus auszuführen, braucht man eine Warteschlange. Dort werden alle Knoten, die man bereits gefunden hat, eingefügt. Der Knoten, zu dem momentan der günstigste Weg vom Startknoten ausführt, steht ganz vorne in dieser Warteschlange.

Zunächst wird der Algorithmus initialisiert:

- Der Weg zum Startknoten hat Kosten 0, denn seine Entfernung zu sich selbst ist natürlich 0.
- Zu den anderen Knoten ist noch kein Weg bekannt, darum werden ihre Kosten zunächst mit unendlich bewertet. Im Laufe des Algorithmus sollen diese Kosten verbessert werden, wobei sich der Algorithmus immer den bisher günstigsten gefundenen Weg zu jedem Knoten merkt.
- Der Startknoten wird in die Warteschlange eingefügt.

Nun wird der vorderste Knoten aus der Warteschlange entnommen. Zu Beginn kann dies nur der Startknoten sein. Dann werden alle Nachbarknoten des entnommenen Knotens mit den jeweiligen Kanten betrachtet und auf folgende Bedingung geprüft: Ist der Knoten in der Warteschlange enthalten und sind die Kosten über die neue Kante geringer als die bisherigen Kosten?

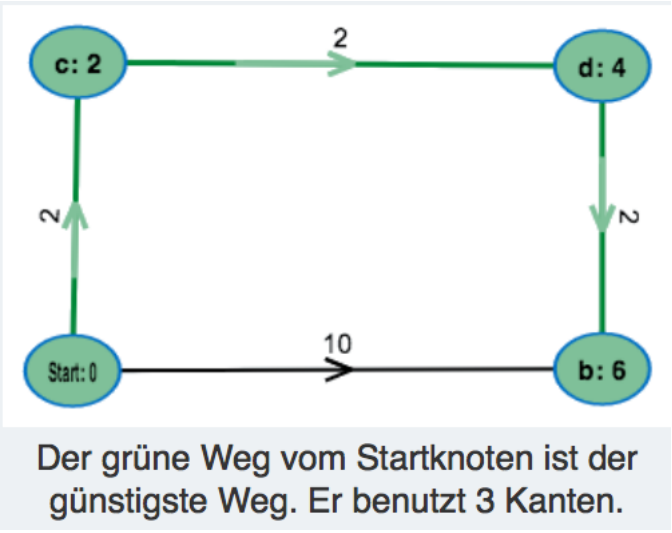
Falls dies der Fall ist, werden die Kosten für diesen Knoten auf den neuen Wert verringert. Ansonsten wird auf eine weitere Bedingung geprüft: Ist der Knoten bisher noch gar nicht besucht worden? Falls ja, wird er nun in die Warteschlange eingefügt, damit später auch die Kanten, die von ihm aus weggehen, betrachtet werden können. Außerdem erhält dieser Knoten die Kosten, die sich aus der Summe der Kosten zu seinem Vorgängerknoten und den Kosten der neuentdeckten Kante zu ihm selbst ergeben.

Es werden nun so lange Knoten aus der Warteschlange entnommen und ihre Nachbarknoten auf diese beiden Bedingungen geprüft, bis keine Knoten mehr in der Warteschlange enthalten sind. Dann ist der Algorithmus fertig und hat von dem Startknoten ausgehend zu allen anderen Knoten die jeweils günstigsten Wege gefunden.

Bei jeder Aktualisierung der Kosten speichert der Algorithmus die Kante, mit deren Hilfe er aktualisiert wurde, als Vorgängerkante des Knotens.

Am Ende des Algorithmus kann dann der günstigste Weg für jeden Knoten konstruiert werden, indem man so lange über Vorgängerkanten läuft, bis man am Startknoten angekommen ist.

Wenn es von dem Startknoten aus zu einem Knoten keinen Weg gibt, so bleiben seine Kosten unendlich, was bedeutet, dass dieser Knoten niemals erreichbar ist.



Der grüne Weg vom Startknoten ist der günstigste Weg. Er benutzt 3 Kanten.

7.1.2 A*-Algorithmus

Der A*-Algorithmus ist ein Algorithmus mit dem man den kürzesten Weg zwischen zwei Knoten bestimmen kann. Möchte man, wie beim Dijkstra-Algorithmus, alle kürzesten Wege von einem Startknoten zu allen anderen Knoten erhalten, so muss man den Algorithmus entsprechend oft ausführen.

Beim A*-Algorithmus spricht man auch von einem informierten Suchverfahren, da der Algorithmus im Gegensatz zum Dijkstra-Algorithmus nicht blind einfach die nächsten erreichbaren Knoten abarbeitet, sondern weitere Informationen hinzuzieht, um die Knoten auszuwählen, die wahrscheinlich schneller zum Ziel führen.

Als zusätzliche Information verwendet der A*-Algorithmus eine Schätzfunktion, das ist eine Art Daumenregel, die für jeden Knoten angibt, wie weit der Zielknoten ungefähr entfernt ist. Damit kann immer der Knoten als nächstes ausgewählt werden, der wahrscheinlich am schnellsten zum Zielknoten führt. Dadurch wird auch die Suche nach dem kürzesten Weg beschleunigt.

Beim A*-Algorithmus befindet sich jeder Knoten in einem von drei Zuständen:

- Der Knoten ist unbekannt: Er wurde noch nicht bearbeitet und somit ist auch noch kein Weg zu ihm bekannt.
- Der Knoten befindet sich in der Warteschlange: Zu ihm wurde bereits ein Weg gefunden. Es könnte aber noch einen kürzeren Weg geben.
- Der Knoten ist fertig abgearbeitet: Zu ihm wurde der kürzeste Weg vom Startknoten gefunden.

Bei der Ausführung des Algorithmus wird zunächst der Startknoten in die zuvor leere Warteschlange eingefügt. Jeder Knoten in dieser Warteschlange hat einen f-Wert, das ist die Summe aus der Distanz vom Startknoten zu diesem Knoten und der geschätzten Entfernung

von diesem Knoten zum Zielknoten. Der Knoten, der gerade den geringesten f-Wert hat, steht vorne in der Warteschlange und wird als nächstes bearbeitet.

Nun werden solange Knoten mit dem momentan kleinsten f-Wert aus der Warteschlange entnommen bis man den Zielknoten entnimmt oder die Warteschlange leer ist. Ist der entnommene Knoten der Zielknoten, so ist der Algorithmus fertig, da der kürzeste Weg zu diesem Knoten gefunden wurde. Endet der Algorithmus jedoch, weil die Warteschlange leer ist, ohne dass der Zielknoten entommen wurde, so wurde kein Weg zum Zielknoten gefunden, da dieser vom Startknoten aus unerreichbar ist.

Wenn ein Knoten aus der Warteschlange entnommen wurde, ist er fertig abgearbeitet und als nächstes werden seine Nachbarknoten betrachtet. Für jeden der Nachbarknoten werden drei Fälle unterschieden:

- Der Nachbarknoten ist bereits fertig abgearbeitet:
  In diesem Fall passiert nichts weiter mit dem Knoten.
- Der Nachbarknoten befindet sich bereits in der Warteschlange:
  Ist der Weg über die neue Kante kürzer als der bisherige Weg, so wird der f-Wert aktualisiert.
- Der Nachbarknoten war noch nicht in der Warteschlange:
  Der f-Wert des Knotens wird ermittelt und der Knoten wird anhand des f-Werts in die Warteschlange eingereiht.

Für jeden Knoten, zu dem ein kürzester Weg gefunden wurde, kann dieser Weg wie unten beschrieben konstruiert werden.

Der f-Wert eines Knotens, der bestimmt, wie weit vorne der Knoten in der Warteschlange steht, ist die Summe der Distanz des Knotens zum Startknoten und der geschätzten Distanz des Knotens zum Zielknoten. Die Distanz zum Startknoten hängt von den tatsächlichen Kantenkosten ab. Für die geschätzte Distanz zum Zielknoten wird eine Schätzfunktion benötigt.

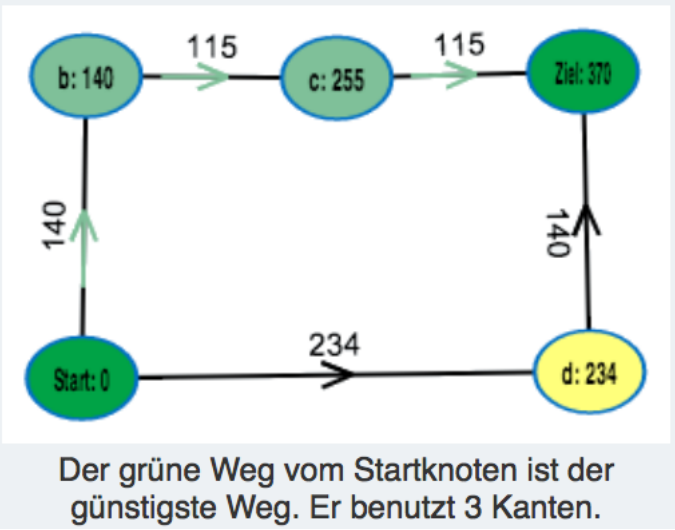
Als Schätzfunktion kann man zum Beispiel die Luftlinie von einem Knoten zum Zielknoten wählen. Mathematisch gesehen ist das dann der euklidische Abstand zwischen den beiden Knoten. Wenn die Koordinaten eines Knotens (x1, y1) sind und die Koordinaten des Zielknotens (x2, y2), dann ist der euklidische Abstand zwischen diesen beiden Knoten √((x1-x2)2+(y1-y2)2).

Der Algorithmus legt jedoch nicht fest, welche Schätzfunktion zu verwenden ist. Im Prinzip kann man diese frei wählen. Allerdings muss man darauf achten, dass die Schätzfunktion auch zulässig ist. Dies bedeutet, dass die Schätzfunktion niemals die Kosten für eine Strecke überschätzen darf. Wenn beispielsweise der Weg von Dorf A nach Dorf B 10 km lang ist. muss die geschätzte Distanz zwischen diesen beiden Dörfern kleiner als 10 km sein. Wenn diese Forderung nicht eingehalten würde, könnte es passieren, dass der Algorithmus keine korrekte Lösung findet, da der Zielknoten zu weit hinten in die Warteschlange eingereiht wird.

Bei jeder Aktualisierung der Kosten speichert der Algorithmus den Knoten, der direkt vor ihm auf dem bisher günstigsten Weg zu ihm liegt, als Vorgängerknoten des Knotens. Oft wird dieser Knoten auch Vaterknoten genannt.

Am Ende des Algorithmus kann dann der günstigste Weg vom Start- zum Zielknoten konstruiert werden, indem man vom Zielknoten aus so lange über Vorgängerknoten läuft, bis man am Startknoten angekommen ist.

Wenn es von dem Startknoten aus zu einem Knoten keinen Weg gibt, so bleiben seine Kosten unendlich. Ebenso bleiben die Kosten der Knoten unendlich, die zwar erreichbar sind, aber auf dem kürzesten Weg vom Startknoten zum Zielknoten nicht besucht werden.



Der grüne Weg vom Startknoten ist der günstigste Weg. Er benutzt 3 Kanten.

7.2 Paxos-Algorithmus

Paxos ist ein Algorithmus zur Konsens Findung. Paxos ermöglicht eine Gruppe von unverlässlichen processes gemeinsam über Dinge verlässlich zu entscheiden. Paxos ermöglicht es also deterministisch und sicher einen Konsenz zu finden sofern die Bedingungen dafür erfüllt werden bzw. sorgt dafür, dass das System konsistent bleibt wenn die Bedingung nicht erfüllt werden kann.

Ein System welches den Paxos Algorithmus implementiert erlaubt es mehreren processes Werte zu speichern und zu lesen auch wenn ein paar der processors ausfallen. Um einen Wert zu lesen muss die Mehrheit der processors zu stimmen, dass dieser Wert der wirklick richtige und aktuelle ist. Daher stört es nicht, wenn einzelne processors falsche Werte liefern.

Vorab sei gesagt, dass das System nur dann funktionieren kann, wenn min. N/2+1 processors funktionsfähig sind, da sonst keine Mehrheit im System gefunden werden kann.

7.2.1 Begriffs-Erklärung

- Process – Ein Computer innerhalb des Systems. Oftmals auch: Replika oder Node
- Client – Ein Computer welcher nicht Teil des Systems ist, der den Wert des Systems abfragt oder einen neuen Wert schreiben will.

7.2.2 Lesen

Beim Lesen fragt ein Client alle Processors des Systems nach ihrem gespeicherten Wert. Der Client nimmt dann den Wert der von der Mehrheit der Processors zurückgelifert wird. Sollte keine Mehrheit gefunden werden kann oder einfach nicht genügend Processors antworten schlägt der Lesevorgang fehl.

7.2.3 Schreiben

Nach dem ein Client einen neuen Wert an einen der Processors gesendet hat, sendet dieser ein Propose an alle anderen Processors im System. Dieses Propose beinhaltet eine Sequenznummer. Alle anderen Processors schicken sofern sie noch kein anderes Propose, mit höherer Sequenznummer erhalten haben, ein Promise an den ersten Processor zurück. Dieser Promise enthält ggf. auch die Sequenznummer und den bereits akzeptierten Wert zurück. Erhält der erste Processor eine Mehrheit an Promises zurück so sendet er ein Accept an die Processor, welche ein Promise gesendet haben. Sofern diese in der zwischen Zeit kein anderes Propose mit höherer Sequenunummer erhalten haben wird der neue Wert akzeptiert.

7.2.4 Konflikte

Sollte ein neues Propose ausgesendet werden, während schon min. ein Processor das vorherige akzeptiert hat, so muss der Processor der den neuen Wert proposed hat, den Wert auf jenen ändern den er von den anderen Processorn im Promise zurückbekommt.

8 Konsistenz und Datenhaltung