h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt
- Fachbereich Informatik -

Sichere Kommunikation über das Controller Area Network (CAN)

Abschlussarbeit zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

vorgelegt von

Jannis Priesnitz
729341

Referent:	Prof. Dr. Ronald Moore
Korreferent:	Prof. Dr. Hans-Peter Wiedling

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Griesheim, den 15. Juli 2015

Jannis Priesnitz

Abstract

Das Controller Area Network (CAN) ist ein umfassend standardisiertes System, was sich in vielen Anwendungsfällen in der Praxis, vor allem im Umfeld von Embedded Systems, bewährt hat. Trotz seiner Einschränkungen hinsichtlich Geschwindigkeit und nutzbarem Datenvolumen wird es bis heute in zahlreichen Systemen eingesetzt und ist immer noch Bestandteil neuer Entwicklungen. Dies ist u.a. auf die hohe Widerstandsfähigkeit gegenüber Störeinflüssen zurückzuführen.

Was passiert aber, wenn nicht nur informationstechnische Störungen vorliegen, sondern das Netzwerk von Angreifern kompromittiert wird? In einem gängigen CAN-Netzwerk kann jeder Teilnehmer beliebige Daten versenden, die von allen Teilnehmern ausgewertet werden. Dies kann zu schwerwiegenden Fehlfunktionen von technischen Anlagen führen.

Diese Arbeit beschäftigt sich mit der Frage, ob eine sichere Kommunikation über CAN grundsätzlich möglich ist, welche kryptografischen Algorithmen dafür besonders geeignet sind und welche Voraussetzungen an die Systeme gestellt werden müssen, um sicher zu kommunizieren. Des Weiteren wird der Vorschlag einer möglichen Softwareumsetzung in Form einer prototypischen Implementierung gemacht und anhand dessen Messungen durchgeführt. Mit Hilfe der Messergebnisse wird eine allgemeine Aussage über die Realisierbarkeit und den Aufwand einer sicheren Kommunikation getroffen.

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Motivation

In den meisten Bereichen unseres beruflichen und privaten Alltags ist die digitale Datenverarbeitung fester Bestandteil unseres Lebens. So haben auch Smartphone-Apps mittlerweile Zugriff auf Steuersysteme von Autos, um dem Besitzer möglichst bequem Informationen zur Verfügung zu stellen [Dai15]. Auch Fertigungssysteme können per Weboberfläche von nahezu jedem Punkt der Welt überwacht und angepasst werden [AD14].

All diese Möglichkeiten bieten jedoch auch Angreifern die Möglichkeit in diese Systeme einzudringen und schwerwiegenden Schaden zu verursachen. Hackern ist es bereits gelungen, sich über das Infotainmentsystem Zugriff auf Steuereinheiten von Automobilen zu verschaffen und elektronische Funktionen remote zu kontrollieren[Rei13].

Auch die von der Industrie angestrebten Zukunft der informellen Fertigungstechnik, die sogenannte „Industrie 4.0“, in der Fertigungsanlagen stark vernetzt werden, nutzt diesen Standard und muss sich vor Angriffen schützen[Hot13].

Das Controller Area Network (CAN) wird in diesen Bereichen eingesetzt, um Daten von einer zentralen Steuereinheit zu Aktoren zu übermitteln und Werte von Sensoren zu empfangen und ist somit das „letzte Glied“ in der Nachrichtenkette. Das zentrale Steuergerät bildet dabei oftmals den Übergang zwischen Protokollen, wie W-Lan und CAN, was einen leichten Zugriff von außen ermöglicht.

Nach aktuellem Stand der Technik wird das Controller Area Network komplett ohne eine Verschlüsselung der übertragenen Informationen und Authentifizierung der Teilnehmer verwendet. Die Sicherheit beruht derzeit ausschließlich auf der Abgeschlossenheit des Systems sowie des meist geheimgehaltenen Softwareprotokolls.

Aus diesem Grund kann eine infizierte Komponente ohne Weiteres andere Teilnehmer übernehmen und sich so im ganzen Netz ausbreiten, was bei Industrieanlagen zu einem Produktionsverlust oder sogar der Zerstörung der Anlage führen kann. Im Fall des Autos besteht darüber hinaus ein hohes Unfallrisiko, da infizierte Komponenten beispielsweise Motorsteuersysteme mit Informationen versorgen. Zudem kann es durch eine Störung zu

einem Versagen von Fahrassistenzsystemen und Sicherheitssystemen kommen, was zur einem erheblichen Risiko für die Insassen wird.

1.2 Thema und Ziel der Arbeit

Das Thema dieser Arbeit ist es, zu klären, ob eine sichere Kommunikation über das Controller Area Network (CAN)grundsätzlich möglich ist. Des Weiteren soll ermittelt werden Mittel dafür eingesetzt werden müssen und welche Einschränkungen dafür in Kauf genommen werden müssen.

Ziel dieser Arbeit ist es zu evaluieren, ob eine verschlüsselte und authentische Kommunikation zwischen einer Teilmenge der am Bus angeschlossenen Teilnehmer, nach dem Stand der Technik, über den CAN-Bus möglich ist. Dabei soll auf die besonderen Randbedingungen bei einem Einsatz im Bereich von Embedded Systems eingegangen werden, in dem CAN hauptsächlich zum Einsatz kommt. Außerdem sollen die Eigenschaften des Busses berücksichtigt und soweit wie möglich erhalten bleiben.

Um eine belastbare Aussage zu treffen, wird das geforderte Sicherheitsniveau auf Basis der kryptografischen Schutzziele eingegrenzt. Auf dieser Basis werden gängige kryptografische Mittel ausgewählt und ihr Zusammenspiel beschrieben.

Aus einem allgemeinen Architekturmodell wird eine prototypische Implementierung abgeleitet, die eine Teilmenge der maximalen Sicherheit darstellt. Anhand des Prototypen sollen Messwerte genommen werden, mit deren Hilfe eine differenzierte Aussage über die Realisierbarkeit einer sicheren Kommunikation getroffen werden kann. Aus diesen Ergebnissen wird auf die Realisierbarkeit und Performance des geforderten Sicherheitsniveaus geschlossen.

1.3 Aufbau der Arbeit

In Kapitel 2 werden zunächst die Grundlagen erläutert, die für das Verständnis der folgenden Kapitel notwendig sind. Dabei wird eine kurze Einführung in die wichtigsten Aspekte der Kryptologie gegeben. Des Weiteren wird das Controller Area Network beschrieben, wobei hier lediglich auf die für diese Arbeit wichtigsten Komponenten eingegangen werden sollen. Der dritte Abschnitt des Kapitels behandelt die Eigenschaften von Embedded Systems. Auch hier ist der Umfang auf die im Rahmen dieser Arbeit maßgeblichen Aspekte beschränkt. Schließlich werden allgemeine Betrachtungen zu einem Kommunikationsmodell vorgestellt, auf dem in den folgenden Kapiteln aufgebaut wird.

Kapitel 3 unterbreitet einen Vorschlag für ein Sicherheitsniveau. Hierbei werden die erforderlichen Schritte im Detail beschrieben, auf Vor- und Nachteile eingegangen und

Alternativen aufgezeigt. Dieses Kapitel hängt direkt mit Anhang A zusammen, welcher ein tabellarisches Schema des vorgestellten Sicherheitsniveaus enthält.

In Kapitel 4 folgt die Beschreibung einer im Rahmen dieser Arbeit erstellten prototypischen Implementierung. Hierbei wird neben den Anforderungen, den Rahmenbedingungen und der Architektur insbesondere auch auf den Nachrichtenfluss eingegangen.

Anschließend werden in Kapitel 5 Messwerte vorgestellt, die mit Hilfe des Prototypen ermittelt wurden. Hierfür erfolgt zunächst eine Beschreibung der eingesetzten Messmethode. Das Kapitel schießt mit einer Interpretation der Ergebnisse ab.

Den Abschluss dieser Arbeit bildet das Fazit in Kapitel 6. Darin befindet sich eine Zusammenfassung der Arbeit, eine Bewertung der Ergebnisse und ein Ausblick auf weitere Aspekte des Themas.

In Anhang B befindet sich ein komplettes Sequenzdiagramm welches zum besseren Verständnis des Prototypen beitragen kann.

Kapitel 2

Grundlagen

Dieses Kapitel legt das Fundament der Arbeit. Zunächst wird der Begriff einer sicheren Kommunikation beschrieben und darauf eingegangen, welche Sicherheitsaspekte im Vordergrund stehen, mit welchen Mitteln sie umgesetzt werden können und welche Vor- und Nachteile die einzelnen Verfahren haben.

Weiter wird das Controller Area Network (CAN) genauer beschrieben. Hierbei wird vor allem auf die für eine sichere Kommunikation relevanten Eigenschaften eingegangen. Da das CAN hauptsächlich im Bereich von Embedded Systems eingesetzt wird, sollen außerdem wichtige Randbedingungen aus diesem Bereich erläutert werden, die eine Entscheidungsgrundlage beim Design eines Systems bieten können.

Schließlich soll ein allgemeines Kommunikationsmodell skizziert werden, welches die grundsätzlich benötigten Komponenten und deren Einflüsse auf die Buskommunikation vorstellt. Dies erfolgt komplett unabhängig von einer konkreten Softwareumsetzung und den kryptografischen Verfahren, die zum Einsatz kommen.

2.1 Sichere Kommunikation

Um eine sichere Kommunikation zu erreichen ist es im Allgemeinen erforderlich ein gefordertes Sicherheitsniveau festzulegen und kryptografische Mittel mit entsprechenden Parametern auszuwählen. Im Folgenden werden einige allgemeine Grundsätze zu den Themen Kryptografie und IT-Sicherheit kurz erläutert und auf die für diese Arbeit wichtigen Verfahren genauer eingegangen.

2.1.1 Schutzziele

Wenn das Thema „Sichere Kommunikation“ behandelt wird, sollte zunächst definiert werden, was genau „Sicherheit“ bedeutet und welche „Arten von Sicherheit“ gewährleistet werden sollen. Dafür geben die Schutzziele[1] der Informationssicherheit eine gute

[1] In der Literatur werden die Worte Schutzziel und Sicherheitsziel synonym verwendet.

Übersicht. [Buc10, S. 1]

Vertraulichkeit: Stellt sicher, dass Informationen unautorisierten Personen nicht zugänglich sind. Dieses Schutzziel ist eines der Kernziele dieser Arbeit und wird durch Verschlüsselung realisiert.

Integrität: Integrität ist ebenfalls von großer Bedeutung für diese Arbeit. Daten dürfen nicht unbemerkt verändert werden und alle Änderungen müssen nachvollziehbar sein. Integrität gegenüber ungewollten Änderungen, wie z.B. Bit-Flips sind bereits im CAN-Protokoll enthalten. Integritätsprüfungen gegenüber gewollter Manipulation einer Nachricht werden durch Authentizität und Nichtabstreitbarkeit abgedeckt und sind somit gewährleistet.

Authentizität: Die Gewährleistung der Echtheit, Überprüfbarkeit und Vertrauenswürdigkeit einer Nachricht ist ein weiteres, wichtiges Ziel. Die Authentizität der Nachrichtenübertragung wird durch einen angefügten Message Authentification Code bewiesen. [Buc10, S. 1]

Nichtabstreitbarkeit / Zurechenbarkeit: Sie erfordert, dass „kein unzulässiges Abstreiten durchgeführter Handlungen“ möglich ist. Nichtabstreitbarkeit gewährleistet die eindeutige Identifizierung der Kommunikationsteilnehmer. Sie muss während der Initialisierung einer sicheren Kommunikation erreicht werden, um sicherzustellen, dass ein Teilnehmer eindeutig zugeordnet werden kann. Dieses Schutzziel wird durch digitale Signaturen erreicht. [Buc10, S. 2]

Verfügbarkeit: Ein nicht direkt im Bereich der Kryptografie angesiedeltes Schutzziel ist die Verfügbarkeit, die sich mit der Verhinderung von Systemausfällen beschäftigt. Der Zugriff auf Daten muss innerhalb eines vereinbarten Zeitrahmens gewährleistet sein. Obwohl Verfügbarkeit im Bereich der Embedded Systems eine wichtige Rolle spielt, kann im Rahmen dieser Arbeit aus zeitlichen Gründen nur am Rande darauf eingegangen werden. [PP09, S. 264f]

Die weiteren Schutzziele „Anonymität / Pseudonymität“ spielen in diesem Bereich eine untergeordnete Rolle und werden in dieser Arbeit nicht betrachtet. [PP09, S. 264f]

2.1.2 Verfahren zum Erreichen der Sicherheit

Wie bereits erwähnt, ist die Verschlüsselung der zu übertragenden Nachrichten eines der Hauptziele dieser Arbeit. Im Folgenden soll der Unterschied zwischen asymmetrischer

und symmetrischer Verschlüsselung erläutert und die jeweils in der Arbeit verwendeten Algorithmen genauer vorgestellt und deren Verwendungszweck beschrieben werden.

Asymmetrische Verschlüsselung

Asymmetrische Verschlüsselungen beruhen im Allgemeinen auf Einwegfunktionen, von denen angenommen wird, dass deren Berechnung relativ einfach im Gegensatz zu deren Umkehrung sind. Das Attribut „asymmetrisch“ bezieht sich dabei auf die Tatsache, dass zwei Schlüssel existieren: eine privater und ein öffentlicher Schlüssel.

Zum Verschlüsseln wird ein öffentlicher Schlüssel unverschlüsselt über einen öffentlichen Kanal an den Kommunikationspartner gesendet. Dieser ist nun in der Lage damit eine Nachricht, die kürzer sein muss, als der öffentliche Schlüssel selbst, zu verschlüsseln. Dazu wendet er die kryptografische Funktion, in Verbindung mit dem Schlüssel auf die Nachricht an. Die verschlüsselte Nachricht wird über den öffentlichen Kanal zurück an den Besitzer des Schlüssels gesendet. Dieser kann die Nachricht nun mit der gleichen kryptografischen Funktion, unter Verwendung seines privaten Schlüssels, entschlüsseln.

Der Vorteil einer asymmetrischen Verschlüsselung liegt darin, das kein sicherer Kanal zum Übertragen des privaten Schlüssels benötigt wird. Der öffentliche Schlüssel kann von jedem Kommunikationsteilnehmer zur Verschlüsselung der Daten oder Prüfung einer Signatur verwendet werden.

Als Nachteile dieses Verfahrens sind zum einen die sehr eingeschränkte, verschlüsselbare Nachrichtenlänge zu nennen, die höchstens so lang sein kann, wie der Schlüssel selbst. Zum anderen sind asymmetrische Kryptosysteme im Vergleich zu symmetrischen Systemen sehr inperformant. [Buc10, S. 60f / S. 135ff]

Das RSA-Kryptosystem Das RSA-Kryptosystem ist das am weitesten verbreitete asymmetrische Kryptosystem. Es basiert auf der Annahme der Schwierigkeit des Faktorisierens großer Primzahlen. Eine Primfaktorzerlegung von zwei hinreichend großen [2] Zahlen gleicher Länge ist nach dem aktuellen Stand der Erkenntnisse, praktisch nicht durchführbar, auch wenn dies bis heute nicht bewiesen werden konnte. Diese sogenannte Einwegfunktion, die sich in eine „Richtung“ leicht, in die andere jedoch schwer berechnen lässt, macht die Sicherheit des Systems aus. [Buc10, 207ff]

Zur Schlüsselgenerierung werden zwei gleich lange Primzahlen p und q gewählt (in der Praxis werden so lange Zufallszahlen generiert und mit probabilistischen Primzahltests geprüft, bis zwei gefunden wurden) und ein RSA-Modul $N = p*q$ errechnet. Danach wird die eulersche ϕ-Funktion von N : $(\phi(N) = (p-1)*(q-1))$ berechnet und ein e bestimmt, welches teilerfremd und kleiner als $\phi(N)$ ist. Der private Entschlüsselungsschlüssel wird nun durch $e * d \equiv 1(mod(\phi(N)))$ berechnet.

[2] „hinreichend“ bedeutet nach dem aktuellen Stand der Technik (RSA2048) 617 Ziffern

Die Verschlüsselung wird nun durch $cipher \equiv message^e (mod N)$ mit e als öffentlichen Schlüssel realisiert; das Entschlüsseln funktioniert mit $message \equiv cipher^d (mod N)$ wobei d der private Schlüssel ist.

Wichtig zu sehen ist, dass das RSA-Verfahren per se nicht sicher gegenüber der „Ciphertext Indistinguishability (Ununterscheidbarkeit der Geheimtexte) (IND-CPA)“ und Chosen-Ciphertext-Attacks ist. Um diese Angriffe auszuschließen, wird der RSA mit einem Padding versehen. Hierfür zumeist das in Public Key Cryptography Standards (PKCS)#1 standardisierte Optimal Asymmetric Encryption Padding (OAEP) verwendet, welches auf kryptografischen Hashfunktionen beruht. [Buc10, S. 149ff]

Das RSA-Verfahren kann sowohl zur Verschlüsselung als auch zur Signatur von Nachrichten verwendet werden. Hierzu wird ein Fingerprint (üblicherweise ein Hashwert) der Nachricht mit Hilfe des privaten Schlüssels authentisiert und zusammen mit der Nachricht versendet. Der Empfänger entschlüsselt (authentifiziert) nun anhand des öffentlichen Schlüssels die Nachricht und prüft, ob das Ergebnis mit dem Hashwert der empfangenen Nachricht übereinstimmt. [Buc10, S.207ff]

Symmetrische Verschlüsselung

Moderne symmetrische Verschlüsselungsverfahren bestehen aus Block- oder Stromchiffren, die ihre Eingabedaten mit einem Schlüssel über mehrere Runden substituieren und permutieren, sodass sie für einen Angreifer ohne Schlüssel nicht zu entschlüsseln sind.

Der Vorteil dieser Substitutions-Permutationsnetzwerke liegt darin, dass sie beliebig viele Daten in kurzer und nahezu konstanter Zeit ver- und entschlüsseln können und somit im Gegensatz zu asymmetrischen Systemen sehr performant sind. Der Nachteil jedoch ist, dass dafür der selbe Schlüssel verwendet wird, welcher über einen sicheren Kanal übertragen werden muss. [Buc10, S. 60f]

Der Advanced Encryption Standard (AES) Ein bekanntes und sehr häufig eingesetztes Verfahren zur symmetrischen Verschlüsselung ist der Advanced Encryption Standard . Dies beruht auf der Standardisierung des Algorithmusses durch das National Institute of Standards and Technology (NIST). Der AES verwendet eine Blockchiffre, basierend auf einem Substitutions-Permutations-Netzwerk. Hierbei wird eine Eingabe fester Länge über mehrere Runden substituiert („ersetzt“) und permutiert („vermischt“), sodass ein möglicher Angreifer keine Rückschlüsse auf den eingegeben Klartext ziehen kann.

Der dabei verwendete Schlüssel sollte eine Länge von 128, 192 oder 256 Bit aufweisen und wird zu Beginn des Verfahrens zu Rundenschlüsseln weiterverarbeitet, die am Ende einer Runde auf das Ergebnis „addiert“ werden. [PP09, S.99ff]

Betriebsarten und Paddingverfahren Da der AES zunächst nur eine feste Länge gleich der Schlüssellänge verschlüsseln kann, werden Betriebsarten dazu verwendet, um beliebig lange Nachrichten zu verschlüsseln. Diese haben die Aufgabe gleiche Klartextblöcke auf verschiedene Chiffrate abzubilden. Dafür nutzt eine Blockchiffre einen Initialisierungsvektor, der wie der Schlüssel zufällig und geheim sein muss. Als Betriebsart kommen der Counter Mode, dessen Spezialisierung der Galois-Counter-Mode und das Cipher-Block-Chaining in Frage.

Da die Betriebsart nur die Verschlüsselung eines Vielfachen des Schlüssels unterstützt, muss noch ein Paddingverfahren eingeführt werden, welches eine Nachricht auf die nächst größere Blockgröße erweitert. Hierbei bietet sich das ISO-Padding an, welches die verbleibenden Bits mit Nullen auffüllt. [PP09, S.30f, S.124ff]

Neben Blockchiffren gibt es außerdem Stromchiffren, die in diesem Anwendungsfall des AES keinen Sinn ergeben, da nur Nachrichten mit fester Größe übertragen werden. Daher werden diese hier nicht weiter betrachtet.

Hybride Verschlüsselung Die Verbindung eines symmetrischen Verfahrens zum Verschlüsseln der Datenmasse mit einem asymmetrischen Verfahren zum Übertragen eines symmertrischen Schlüssels nennt man im Allgemeinen eine hybride Verschlüsselung. Sie ist Grundbestandteil vieler Kryptosysteme, wie z.B. W-LAN. [Buc10, S. 136f]

Man unterscheidet zwischen Daten- und Instanzauthentizität:

Datenauthentizität

Mit Datenauthentizität wird gewährleistet, dass die gesendeten Daten auf dem Weg nicht verändert wurden. Dies schließt sowohl die ungewollte Veränderung der Daten durch technische Fehler als auch die mutwillige Veränderung durch einen Angreifer ein.

Message Authentication Code (MAC) Dies wird über einen MAC realisiert. Dazu wird aus der zu sendenden Nachricht und einem zuvor vereinbarten geheimen Schlüssel ein Hashwert gebildet (Keyed-Hash Message Authentication Code (HMAC)) oder durch Anwendung einer Blockchiffre ein Prüfwert berechnet (Cipher-Based Message Authentication Code (CMAC)). Dieser Hash- oder Prüfwert kann nun von jedem, der in Besitz des Schlüssels ist, verifiziert werden, wodurch die Authentizität der Daten bewiesen wird. [PP09, S.319ff]

Instanzauthentizität

Instanzauthentizität folgt der gleichen Idee wie Datenauthentizität, geht jedoch noch einen Schritt weiter. Hier wird nicht nur die Authentizität der Daten gewährleistet, sondern zusätzlich die Authentizität der sendenden Instanz bewiesen.

Dies geschieht im Allgemeinen über ein asymmetrisches Verfahren, wobei der private Schlüssel zum Signieren der Nachricht verwendet wird. Der Empfänger kann mit der empfangenen Signatur die Nachricht verifizieren und so die Identität des Senders feststellen. Normalerweise wird eine Signatur nicht direkt über eine Nachricht gebildet, sondern über deren Hashwert.

Um die Identität des Teilnehmers sicherzustellen, ist zunächst ein Authentifizieurngsverfahren erforderlich, welches eine bestimmte Eigenschaft des Teilnehmers beweist. Oftmals ist diese Eigenschaft das Wissen eines Geheimnisses (Master Secret / Pre Shared Secret).

Challenge Response Authentifizierung Das Challenge Response Protokoll ist ein relativ simples Verfahren, zum authentifizieren eines Teilnehmers auf der Basis von Wissen. Wie der Name vermuten lässt, stellt ein Teilnehmer eine „Aufgabe“ (challenge), die ein anderer lösen muss (response). Hier darf jedoch das gemeinsame Geheimnis nie übertragen werden.

Das Grundprinzip dahinter basiert auf dem Senden einer Nonce [3] vom Prüfer. Der Teilnehmer konkateniert[4] die Nonce mit dem Pre Shared Secret und hasht das Ergebnis mit einer kryptografisch starken Hashfunktion. Das Ergebnis wird nun an den Prüfer gesendet, welcher ebenfalls aus der Nonce und dem Pre Shared Secret einen Hashwert bildet. Der Prüfer ist nun in der Lage die beiden Hashs abzugleichen und so das Geheimnis zu verifizieren.

Da auf dieses Verfahren eine Vielzahl von Angriffen möglich ist, müssen weitere Absicherungen getroffen werden. Um Wörterbuchangriffe und Known-Plaintext-Angriffe erfolglos zu machen, ist es erforderlich die Übermittlung der Nonce zu verschlüsseln. Außerdem sollte zudem ein Timestamp mit kurzer Gültigkeitsdauer verwendet werden, um Replay-Attacken und Choosen-Plaintext-Attacken auszuschließen. Grundsätzlich sollte für jede Verbindung immer eine neue Nonce verwendet werden.

Ein großer Nachteil des Verfahrens ist, dass das Per Shared Secret in beiden Teilnehmern im Klartext vorliegen muss. [Eck13]

Digital Signature Algorithm (DSA) Der DSA stellt einen standardisierten Algorithmus zum digitalen Signieren von Daten dar. Er beruht auf der Schwierigkeit des diskreten Logarithmierens in endlichen Körpern und stellt daher eine Einwegfunktion bzw. ein asymmetrisches Kryptosystem dar.

Um einen Schlüssel zu erzeugen werden hier zwei Primzahlen p und q gewählt. Die

[3] Eine Nonce ist in der Kryptografie eine „number used only once“, also eine zufällige Nummer, die nur einmal verwendet werden darf[Wik15c]. Hinzu kommt hier noch der Timestamp, um Replay-Attacken zu verhindern. Hier wird der Timestamp zusammen mit der Nonce als ein Datensatz gesehen.

[4] „Konkatenieren“ bedeutet in diesem Zusammenhang das kryptografische Verbinden der Nonce mit dem Pre Shared Secret

Länge von p muss zwischen 512 und 1024 Bit betragen und ein Vielfaches von 64 sein. Außerdem muss q über eine Länge von 160 Bit verfügen, wobei q ein Teiler von $p-1$ sein muss. Nun wird ein h mit $1 < h < p-1$ gewählt und $g = h^{\frac{(p-1)}{q}} \mod p$ berechnet. Ist dies 1, so muss ein anderes h gewählt werden. Schließlich wird ein x mit $1 < x < q$ gewählt und $y = g^x \mod p$ berechnet. Der öffentliche Schlüssel besteht nun aus p, q, g und y wobei x der geheime Schlüssel ist.

Eine Signatur wird mit Hilfe eines zufälligen s erstellt, wobei $1 < s < q$ gelten muss. Nun wird $s_1 = (g^s \mod p) \mod q$ berechnet und anschließend $s_2 = s^{-1} * (m + s_1 * x) \mod q$. Als Signatur wird nun s_1, s_2 zusammen mit der zu signierenden Nachricht übertragen. Hierbei ist zu beachten, dass sowohl s_1 als auch s_2 nicht 0 sein dürfen. Überprüft werden kann die Nachricht mit $w = s_2^{-1} \mod q$ und anschließend $u_1 = m * w \mod q$ und parallel $u_2 = s_1 * w \mod q$. Schließlich wird $v = (g^{u1} * y^{u2} \mod p) \mod p$ berechnet. Nun ist $v = s_1$, wenn die Signatur richtig ist. Auch hier wird statt der direkten Nachricht im Allgemeinen ein Hashwert der Nachricht verwendet.

Ohne jeden Schritt im Detail nachvollziehen zu müssen ist ersichtlich, dass der Algorithmus aufgrund der Vielzahl von Operationen, der langen Schlüssel und vor allem dem möglichen Misslingen durch ein Nullergebnis beim Signieren für den aktuellen Anwendungsfall eher ungeeignet ist.

In 2.1.2 wird eine Erweiterung um Elliptic Curvess (ECs) vorgestellt. [Buc10, S. 217ff]

RSA-Signaturverfahren Das RSA-Signaturverfahren funktioniert analog zur Verschlüsselung mit dem RSA-Algorithmus, mit dem Unterschied, dass hier nicht mit dem öffentlichen Schlüssel des Kommunikationspartners verschlüsselt, sondern mit dem eigenen privaten Schlüssel signiert wird. Wie bereits beschrieben, kann der Kommunikationspartner mit dem öffentlichen Schlüssel die Signatur gegenüber der Nachricht verifizieren.

Mit Instanzauthentizität wird das Schutzziel Nichtabstreitbarkeit gewährleistet.

Schlüsseleinigungsverfahren

Bei Verfahren zur Schlüsseleinigung geht es darum, ein gemeinsames Geheimnis zwischen zwei Kommunikationspartnern über einen unsicheren Kanal zu erstellen.

Diffie Hellman Schlüsseltauschverfahren Beim Diffie-Hellman Schlüsseltauschverfahren wird jeweils eine Nachricht zwischen den Kommunikationsteilnehmern übertragen, wodurch beide Kommunikationsteilnehmer in der Lage sind, ein gemeinsames Geheimnis zu berechnen. Das Verfahren beruht auf dem Diffie-Hellman-Problem, welches sehr ähnlich zu dem Problem des diskreten Logarithmierens ist.

Dazu müssen sich beide Kommunikationspartner vorab auf eine zyklische Gruppe p einigen, deren Ordnung prim ist. Außerdem wird ein Erzeuger g der Gruppe, also

eine Zahl, aus der die gesamte Gruppe generiert werden kann, benötigt. Beide Partner wählen nun jeweils eine Zufallszahl a und b aus der Gruppe p und berechnen $A = g^a \mod p$, $B = g^b \mod p$ und übertragen A, B . Schließlich berechnet der Empfänger des jeweiligen A, B den Schlüssel $K = A^b \mod p$ bzw. $K = B^a \mod p$, wobei K für beide Kommunikationspartner gleich und das gemeinsame Geheimnis ist. [PP09, S. 206ff]

Hashverfahren

Hashverfahren haben die Aufgabe eine beliebig lange Nachricht auf einen eindeutigen Hashwert (Fingerprint) fester Größe abzubilden. Diese Einwegfunktionen erhalten keinen Schlüssel und sind daher lediglich für das Beweisen der Datenauthentizität geeignet. Außerdem werden Hashfunktionen auch als Funktion zur Ableitung von Schlüsseln (Key Derivation Function) aus einem gemeinsamen Geheimnis verwendet. Hashfunktionen basieren zumeist auf Blockchiffren, auf Abwandlungen der Merkle-Damgård-Konstruktion oder anderen Berechnungsvorschriften.

Hashfunktionen müssen drei wichtige Eigenschaften bereitstellen, damit sie nicht leicht kompromitierbar sind und somit einen Angriffspunkt bieten. So muss eine starke Hashfunktion „preimage resistant“ sein, was bedeutet, dass es unmögliche sein muss, aus der Ausgabe auf die Eingabe in die Funktion zurück zu schließen. Außerdem darf zu einem bekannten Paar aus Eingabe und Hashwert keine zweite Eingabe mit dem gleichen Hashwert gefunden werden (Second preimage resistance). Drittens dürfen keine verschiedenen Eingaben zum gleichen Hashwert führen (Collision resistance). [PP09, S. 303]

Dies schränkt die Anzahl der möglichen Hashfunktionen stark ein.

Secure Hash Algorithm (SHA) Der Secure Hash Algorithm (SHA) ist das am häufigsten eingesetzte kryptografische Hashverfahren. SHA existiert mittlerweile in drei verschiedenen Versionen, wobei SHA1 nicht mehr als sicher gilt. Darüber hinaus können mit SHA2 und SHA3 Hashs mit einer Länge von 224, 256, 384 und 521 Bit erstellt werden, die folglich unterschliedliche Sicherheitsniveaus bieten.

Exemplarisch soll hier SHA1 mit einem 160 Bit Hashwert vorgestellt werden, welche in ähnlicher Form auch in SHA2 verwendet wird. Hierbei werden die Quelldaten ggf. gepadded, in Blöcke mit je 512 Bit unterteilt, und iterativ mit 4 Konstanten verrechnet. Dafür werden vier Stufen mit je 20 Runden der Cipher Block Chaining (CBC)-Blockchiffre durchgeführt. Dabei wird sowohl die Nachricht, als auch das Ergebnis der letzten Runde als Eingabe der nächsten Runde verwendet. Jede der Stufen wird dabei mit anderen Konstanten und Abwandlungen des CBC durchgeführt. Die Initialisierungsvektoren sind dabei ebenfalls unterschiedlich. Die konkatenierten Ausgaben eines Teils des Ergebnisses der Blockchiffre bildet nun das Ergebnis.

Auch wenn die hohe Anzahl an Runden und das Verfahren an sich relativ kompliziert

erscheint, werden überwiegend Operationen benutzt, die für einen Prozessor schnell ausführbar sind und somit eine schnelle Berechnung eines Hashwertes zulassen. Auf größeren System kommt hier auch eine Hardwareimplementierung zum Einsatz. [PP09, S.307ff]

Elliptische-Kurven-Kryptographie

Das EC-Kryptosystem ist wesentlich jünger als der RSA-Algorithmus. Die Sicherheit des Elliptic Curves-Verfahrens beruht, wie die Sicherheit des DSA und des Diffie-Hellman Schlüsseltausches, auf dem Problem des diskreten Logarithmus in endlichen Körpern. Dieser nutzt die Annahme, dass die diskrete Expotentialfunktion einfach berechnet werden kann, der diskrete Logarithmus dagegen nur schwierig berechenbar ist. Dies führt zu der benötigten Einwegfunktion.

Da sich der diskrete Logarithmus auf einer endlichen Gruppe Z_p z.B. mit Hilfe des Babystep-Giantstep-Algorithmusses relativ effizient (Polynomialzeit) berechnen lässt, gelten Verfahren, die das Problem des diskreten Logarithmus als Grundlage haben, nur noch mit sehr langen Schlüsseln (also großen Primzahlen) als sicher.

Hier hilft jedoch die Definition einer elliptischen Kurve über einem endlichen Körper. Dabei werden die Multiplikations- und Exponentationsoperationen in Z_p , z.B. des Diffie Hellman Key Establishments (DHKEs), auf Punktadditions- und Punktmultiplikationsoperationen auf einer elliptischen Kurve übertragen.

Was wir zum Aufbau der Kurve benötigen ist, wie bei den vorangegangenen Kryptosystemen, ein endlicher Körper primer Ordnung Z_p, worauf eine elliptische Kurve definiert wird. Diese Kurve ist anschaulich ausgedrückt nichts anderes, als eine Menge an Punkten, welche Lösungen der Gleichung $y^2 \equiv x^3 + a * x + b \mod p$ sind (dies folgt aus der kurzen Weierstraß-Gleichung). Außerdem wird ein imaginärer Punkt im Unendlichen benötigt, der, äquivalent zu Operationen im gesamten Z_p, als neutrales Element bei den Operationen fungiert. Um eine Singularität ausschließen zu können, muss die Bedingung $4 * a^3 + 27 * b^2 \neq 0 \mod p$ gelten. In [PP09, 247] ist definiert und genauer begründet, dass das Problem des diskreten Logarithmus ebenfalls auf elliptischen Kurven gilt, was an dieser Stelle nicht weiter beschrieben werden soll. Zusammengefasst stellt sich das diskrete Logarithmus-Problem auf elliptischen Kurven so dar, dass auf einer gegebenen Kurve E zwei Punkte P, T gewählt werden. Das diskrete Logarithmus-Problem besteht nun darin, mit d Punktadditionen und -verdopplungen den Punkt T zu erreichen. (Die Kombination von Punktaddition und -verdopplung wird auch Punktmultiplikation genannt.) Hierbei wird P zuvor öffentlich vereinbart, T stellt den öffentlichen und d den privaten Schlüssel dar. d ist dabei die Anzahl von Operationen, um von P nach T zu kommen. Dadurch lässt sich die Punktaddition und -multiplikation auf die Multiplikation und Exponentation in konventionellen Verfahren beziehen.

Die Elliptic Curves-Cryptography stellt somit kein neues Verfahren dar, sondern erweitert bestehende Verfahren, die auf der Schwierigkeit des diskreten Logarithmus in

endlichen Körpern basieren auf einer elliptischen Kurve innerhalb des Körpers. So ergibt sich auf einigen Kurven mit kurzen Schlüsseln ein Sicherheitsniveau, welches nach dem Stand der Technik ein akzeptables Maß an Sicherheit bietet.

Da die Kurve bestimmten Anforderungen entsprechen muss, welche genau überprüft werden müssen, wird im Allgemeinen eine bereits bestehende, zertifizierte Kurve gewählt.

Elliptic Curves Diffie Hellman (ECDH) Um einen gemeinsamen Schlüssel mit Hilfe des DHKE basierend auf elliptischen Kurven zu generieren, werden, wie zuvor beschrieben, zunächst die Domainparameter definiert. Hierfür benötigen wir ein p der Größe von 160 Bit, welches prim ist und welche die endliche Gruppe definiert, eine elliptische Kurve E und ein Element der Kurve P.

Die Schlüsselgenerierung geschieht nun analog zum konventionellen Diffie-Hellman Protokoll: Beide Teilnehmer wählen je eine Primzahl a, b aus Z_p und berechnen $A = a*P$ bzw. $B = b * P$ (hier kommt die Punktmultiplikation zum Einsatz) und teilen sich gegenseitig diese Werte mit. Nun wird das gemeinsame Geheimnis $T_{AB} = a * B$ bzw. $T_{AB} = b * A$ von beiden Teilnehmern berechnet.

Elliptic Curves Digital Signature Algorithm (ECDSA) Das ECDSA Verfahren ist, wie bereits beschrieben, ebenfalls lediglich eine Erweiterung des Digital Signature Algorithm. Diese folgt der gleichen Idee wie bereits in den vorangegangenen Abschnitten beschrieben und soll hier nicht weiter besprochen werden.

2.1.3 Schlüsselverwaltung

In den meisten Kryptosystemen kommt eine Vielzahl von Schlüsseln zum Einsatz. Hierbei haben bestimmte Schlüssel bestimmte Aufgaben und „Lebzeiten“, in der sie gültig sind, welche hier näher beschrieben werden sollen.

Schlüsselhierarchie

Eine Schlüsselhierarchie beschreibt das Zusammenwirken mehrerer Schlüssel mit unterschiedlichen Aufgaben und Lebenszeiten. Diese Hierarchie setzt sich im Allgemeinen aus einem Masterkey oder Pre-Shared Secret, Langzeitschlüsseln und Sitzungsschlüsseln zusammen.

Master Secret Der Masterkey oder Master Secret ist die Wurzel der sicheren Kommunikation. Er dient der Prüfung eines Teilnehmers und aus ihm können weitere Schlüssel abgeleitet werden. Auch, wenn es nicht genau der Definition entspricht, kann im Rahmen dieser Arbeit anschaulich das in 2.3.4 vorgestellte Passwort als Masterkey betrachtet werden. Hierbei ist jedoch zu beachten, dass es zwar im Rahmen des Challenge-Respnonse-

Verfahrens getrüft wird, jedoch keine Schlüsselableitung aus dem Passwort abgeleitet. [Eck13, S. 400f]

Langzeitschlüssel Langzeitschlüssel oder Verbindungsschlüssel dienen nur dem Austausch von gemeinsamen Geheimnissen, wie z.B. den Sitzungsschlüsseln. Langzeitschlüssel werden bei der Initiierung der Kommunikation vereinbart und sind über einen langen Zeitraum gültig, der je nach System individuell festgelegt werden kann. Bei Systemen mit einer beschränkten Laufzeit ist es durchaus üblich, den Langzeitschlüssel zur Laufzeit nicht zu tauschen. [Eck13, S. 400f], [BSI05, S. 51] Das zugrunde liegende Kryptosystem ist hier im Allgemeinen asymmetrisch.

Sitzungsschlüssel Mit den Sitzungsschlüsseln wird die komplette Kommunikation verschlüsselt, was den Großteil der übertragenen Daten ausmacht. Daher sollten diese innerhalb einer kürzeren Zeitspanne ausgetauscht werden (der sogenannte Key Update), um einer Kompromittierung keine Möglichkeit zu bieten. Das hier zugrunde liegende Kryptosystem ist ein symmetrisches. [Eck13, S. 400f]

Schlüsselupdate

Ein Schlüsselupdate beschreibt den Austausch der Sitzungsschlüssel während der Laufzeit des Systems. Dies erhöht das Sicherheitsniveau deutlich, da die häufig verwendeten Sitzungschlüssel ein großes Angriffsziel bieten. Würden sie nicht getauscht hat ein Angreifer beliebig lange Zeit den Sitzungsschlüssel zu kompromittieren. Für das Schlüsselupdate gibt es unterschiedliche Verfahren, wobei im Allgemeinen entweder die Langzeitschlüssel zur Verschlüsselung der Sitzungsschlüssel verwendet werden oder ebenfalls neue Langzeitschlüssel erstellt werden. [Eck13, S. 400f]

Perfect Foreward Secrecy (PFS)

Die Eigenschaft der Perfect Foreward Secrecy eines Kryptosystems ist gegeben, wenn ein Angreifer alle Langzeitschlüssel aufdeckt, er jedoch nur die aktuelle Sitzung kompromittieren kann. Das heißt, dass er keinen Zugriff auf folgende Sitzungsschlüssel erhält.

In Abgrenzung dazu ist eine Backward-Secrecy erreicht, wenn lediglich die Sitzungsschlüssel nicht voneinander abgeleitet werden und die vergangenen Schlüssel gelöscht werden. Es gibt so keine Möglichkeit den Schlüssel zu rekonstruieren und aufgezeichnete Kommunikationen damit zu entschlüsseln.

Die Foreward-Secrecy leistet dagegen die Sicherheit, dass alle folgenden Kommunikationen nicht entschlüsselt werden können, was auf der Annahme basiert, dass die gewählten Schlüssel nicht voneinander abgeleitet werden können und somit aus dem kompromittierten Schlüssel nicht auf die folgenden geschlossen werden kann. Ein Pro-

blem bleibt jedoch, dass mit einem gebrochenen Langzeitschlüssel jeder Sitzungsschlüssel entschlüsselt werden kann.

Bei der Perfect Foreward Sercrecy schließlich wird auch diese Lücke behoben, in dem für jeden Sitzungsschlüssel ein neuer Langzeitschlüssel generiert wird. Dies erfolgt im Allgemeinen über das Diffie Hellman Key Establishment oder den Elliptic Curves Diffie Hellman.[Eck13, S. 402]

Reihenfolge von Verschlüsselung und Signatur

Ein leicht zu realisierender, jedoch sehr wichtiger Punkt ist die Reihenfolge von Verschlüsselung und Signatur. Es sollte immer erst verschlüsselt und danach signiert werden. Da ein Angreifer ohne weiteres Kenntnis des öffentlichen Verschlüsselungsschlüssel hat, ist er in der Lage Schadcode mit diesem verschlüsseln, der nach der Entschlüsselung das Zielsystem kompromittieren kann. Verifiziert der Empfänger jedoch zunächst die empfangene Nachricht bemerkt er vor dem Entschlüsseln, dass die mitgesendete Signatur nicht mit der errechneten zusammen passt. Damit erkennt er dei Fälschung und entschlüsselt die Nachricht nicht. [Eck13, S.400f]

Teile, wo du teilen kannst

Ein Grundprinzip der Kryptologie ist es, möglichst für jedes Schutzziel einen eigenen Schlüssel zu verwenden, um ein möglichst hohes Maß an Sicherheit zu bieten. So ist es erforderlich, jeweils einen eigenen Schlüssel zum Verschlüsseln und zum Signieren zu verwenden. Anderenfalls könnte ein Angreifer, der einen Schlüssel bricht, sowohl verschlüsseln als auch signieren und hätte die komplette Sicherheit des Systems gebrochen. Bei zwei unterschiedlichen Schlüsseln ist dies jedoch nicht möglich, wodurch das Sicherheitsniveau steigt. [Eck13, S.400ff]

2.2 Das Controller Area Network (CAN)

Das Controller Area Network gehört zu Kartegorie der Feldbusse, dessen Hauptaufgabe es ist, die Kommunikation zwischen Sensoren und Aktoren in technischen Systemen zu vereinfachen. CAN wird bereits seit 1987 primär im Bereich der Automobilindustrie und Automatisierungstechnik eingesetzt, ist jedoch auch im Bereich der Medizintechnik, in Zügen und Flugzeugen zu finden [Wik15a]. Es ist bis heute eines der am weitesten verbreiteten Bussysteme, was auf seine große Flexibilität, sein einfaches Design und geringe Anfälligkeit gegenüber Störeinflüssen zurückzuführen ist [Wik14a]. Seit 2012 existiert mit CAN-FD eine Weiterentwicklung des bereits existierenden Standards, die über einen höheren Datendurchsatz und eine bessere Fehlererkennung verfügt[Wik15a].

Wie bei Feldbussen üblich, ist das Ziel beim Einsatz von CAN im Allgemeinen den Verkabelungsaufwand zu reduzieren und somit Platz und Gewicht einzusparen. Auch

führt die Verkabelung über ein Bussystem zu einem übersichtlicherem Aufbau und weniger fehleranfälligen Systemen. Auf der anderen Seite schafft dies jedoch einen Single Point of Failure, da bei elektronischen oder physischen Störungen oftmals der komplette Bus ausfällt.

In diesem Kaptiel soll die allgemeine Funktionsweise, erläutert und für die Arbeit relevante Sachverhalte genauer beschrieben werden.

2.2.1 Grundsätzliche Daten und Funktionsweise

Ein CAN-Bus-System ist theoretisch in der Anzahl der Teilnehmer unbegrenzt, wird jedoch praktisch aus elektrotechnischen Gründen auf 128 Teilnehmer an einem Bus begrenzt. Dabei werden keine Verwaltungsinformationen über die Teilnehmer, wie zum Beispiel IP-Adressen bei Ethernet, verwendet[Ets01, S.38]. Dies gewährleistet eine hohe Systemflexibilität, da Knoten einfach zum Bus hinzugefügt werden können. Des werden wird damit die Multicastfähigkeit erreicht, da eine Nachricht auf Basis eines Identifiers nahezu synchron von allen „interessierten“ Empfängern empfangen wird. Auf der anderen Seite führt dies jedoch auch zu einer leichteren Angreifbarkeit des Systems von außen, da ein Angreifer lediglich einen validen Identifier benötigt, um eine Nachricht zu versenden und gegebenenfalls zahlreiche Teilnehmer gleichzeitig erreicht[Bos91].

2.2.2 Topologie

Wie bei Feldbussen üblich, wird in der Regel eine Bustopologie verwendet, welche das Netzwerk resistent gegen den Ausfall einer Komponente macht, geringe Mengen an Kabelleitung und keine aktiven Komponenten, wie z.B. Switches, erfordert und sich einfach erweitern lässt. [Wik14b]

Im Netzwerk wird, ähnlich wie bei Ethernet, das Multi-Master-Prinzip angewendet, bei dem jeder Teilnehmer auf den Bus schreiben darf, wenn dieser frei ist. Dies führt unweigerlich zu Kollisionen, welche durch das Carrier Sense Multiple Access / Collision Resolution-Verfahren nicht nur erkannt, sondern auch mit Hilfe einer Arbitierung aufgelöst werden können. Bei einer Kollision zweier Nachrichten wird nur die Nachricht geringerer Priorität verworfen, während die höher priorisierte Nachricht fehlerfrei übertragen wird [Ets01][Bos91].

2.2.3 Nachrichtenformate

Das Controller Area Network (CAN) spezifiziert folgende Nachrichtenformate:

Datenframe Übertragung von 8 Byte Datenvolumen verbunden mit einem priorisierenden Identifier. Die sichere Übertragung des Informationsgehaltes dieses Frames ist Kern dieser Arbeit.

Remoteframe Aufforderung eines Teilnehmers Daten zu senden. Hierbei ist die CAN-ID des Remoteframes gleich der ID der angeforderten Daten. Der Empfänger eines Remoteframes sendet schnellst möglich die mit der ID verbundenen Nutzdaten.

Errorframe Der Errorframe signalisiert einen Fehler eines der Teilnehmer im Netz. Der Empfänger, der einen Fehler in einer Nachricht erkennt, sendet einen Error-Frame aus. Busteilnehmer, die einen Error-Frame empfangen, reagieren darauf mit einer Fehlerbehandlung und ggf. des vorübergehenden oder kompletten Aussetztens der Kommunikation. Dies wird über einen Errorcounter des jeweiligen Teilnehmers geregelt.

Overloadframe Der Overloadframe wird genutzt, um eine Pause zwischen Daten- und Remoteframes herzustellen.

Das Senden von Remote-, Error-, und Overloadframes spielt im Rahmen der sicheren Kommunikation keine Rolle, da diese keine sicherheitskritischen Daten enthalten. Eine Übersicht der im Rahmen dieser Arbeit gewonnen Erkenntnisse über Angriffe mit Hilfe dieser Frames ist im Abschnitt 6.3 zu finden. [Bos91, S. 5]

2.2.4 Object-Identifier und K-Matrix

Wie aus dem Abschnitt 2.2.2 hervorgeht, verfügt CAN über keine Identifizierung der Busteilnemer, jedoch über Object Identifier, sogenannte CAN-IDs. Dabei handelt es sich um eine Zahl zwischen 1 und $7FFF_{16}$ (Länge: 11Bit) die *eine Nachricht* eindeutig identifiziert. Ein Teilnehmer kann also beliebig viele Nachrichten mit unterschiedlichen CAN-IDs versenden und empfangen, jedoch kann eine Nachricht immer nur von einem Teilnehmer versendet werden. Dies folgt direkt aus der Idee, die Nachrichtenarbitierung und -priorisierung mit dem gleichen Datenfeld zu realisieren. Würde es zwei Frames der gleichen CAN-ID auf dem Bus geben, würde Arbitierung fehlschlagen und das System nicht mehr funktionieren. [Ets01]

Die Verteilung Objekt-Identifier wird im Allgemeinen beim Design des technischen Systems festgelegt und ist über den kompletten Lebenszyklus des Systems gleich. „Die Liste der Objekt-Identifier einschließlich Sender und Empfänger ist Bestandteil der sog. Kommunikationsmatrix oder K-Matrix, die außer für den direkt an der Steuergeräteentwicklung beteiligten Personenkreis als besonders vertrauliches Dokument für niemanden einsehbar ist.“[Wik15a]

Priorisierung der Nachrichten

Wie bereits erwähnt wird die Priorisierung der Nachricht mit gleichen Feld, wie die Arbitierung realisiert. Das heißt, dass automatisch eine höher priorisierte Nachricht die

Arbitierung[5] gegen eine geringer priorisierte Nachricht „gewinnt“ und somit u.U. schneller übermittelt wird. Da Nachrichten geringerer Priorität bei einer Kollision verworfen werden, muss das Design der K-Matrix sorgfältig durchdacht werden, da ansonsten wichtige Nachrichten, die eine zu geringe Priorität haben, bei hoher Buslast immer wieder verworfen werden. Eine dynamische Umpriorisierung ist hier nicht vorgesehen. [Ets01]

Multi-Master-Prinzip

Das Controller Area Network kennt keine zentrale Autorität, die den Buszugriff steuert. Jeder Knoten kann die Übertragung von Nachrichten starten, sobald der Bus frei ist. Kollisionen werden, wie bereits beschrieben, mit Hilfe der Arbitierung aufgelöst. Dadurch wird die Buslast und die Übertragungszeit, insbesondere von Nachrichten mit hoher Priorität, signifikant gesenkt. [Wik15a]

Babling Idiot

Dabei ist es jedoch theoretisch möglich, dass alle Teilnehmer, ohne eine Überwachung, miteinander kommunizieren, was im Normalfall durch die K-Matrix unterbunden wird. Ein Busteilnehmer, der sich allerdings nicht an die Matrix „hält“ ist in der Lage Nachrichten mit beliebiger Priorität auf den Bus zu senden. Dies führt zum Problem des Babling Idiot, bei dem ein fehlerhaftes oder bösartiges Bauteil den Bus mit Nachrichten hoher Priorität flutet und somit keine anderen Nachrichten mehr ankommen. Diese Problem ist nur mit Hilfe von zusätzlichen Hardwarekomponenten zu lösen.

Datenkonsistenz

Das Multi-Master-Prinzip führt zu einer busweiten Datenkonsistenz, da es gewährleistet ist, dass jeder Knoten eine Nachricht nahezu gleichzeitig empfängt. Dies wird lediglich durch die Ausbreitungslatenz auf dem Bus limitiert. Die Ausbreitung einer Nachricht nur über einen Teil des Busses ist dabei durch die Arbitierung ausgeschlossen.

2.2.5 Senden und Empfangen

Das Senden von Nachrichten auf dem Bus ist sehr einfach. Hierfür muss lediglich der Object-Identifier bekannt bzw. vorgegeben sein und die zu versendenden Daten komplett bereitstehen. Der Treiber konstruiert nun den kompletten Frame, der aus dem Arbitierungsfeld, einer Checksumme, und weiteren, für die Arbeit nicht relevanten Feldern besteht.

[5] Es reicht für das Verständnis dieser Arbeit aus die Arbitierung als eine „Black-Box“ zu sehen, die eine Priorisierung der Nachrichtenübertragung liefert und das Carrier Sense Multiple Access / Collision Resolution (CSMA/CR)-Verfahren unterstützt. Für die detailliertere Informationen soll an dieser Stelle auf die Literatur verwiesen werden.

Prinzipiell können alle Teilnehmer alle Nachrichten empfangen, da es keine Adressierung der Daten gibt. Da dies in den meisten Netzen jedoch nicht gewünscht wird, „hören“ einzelne Teilnehmer lediglich auf eine Teilmenge der Object-Identifier, die für sie bestimmte Daten bereitstellen. Diese Teilmenge wird im Allgemeinen über die K-Matrix statisch festgelegt und wird nicht verändert. Dabei ist es sowohl möglich, dass mehrere Datensätze in nur einem Frame, wie auch ein Datensatz in mehrere Frames übertragen werden. Die Empfangsroutine muss im zweiten Fall jedoch so designt sein, dass sie in der Lage ist entsprechende Frames zu identifizieren und wieder richtig zusammen zu setzen.

2.2.6 Zeitverhalten

Der CAN-Standard spezifiziert zwei verschiedene Datenübertragungsraten. Die Highspeedvariante überträgt bis zu 1 MBit/s, wohingegen Lowspeed-CAN nur bis zu 125 KBit/s überträgt.

Wichtig ist hier jedoch, dass dies lediglich die Ausbreitungszeit auf dem Bus, nicht aber Treiberaufrufe und andere zeitrelevante Aktivitäten im Sender oder Empfänger einschließt.

2.2.7 Bereits vorhandene Sicherungssysteme

Als Absicherung der Nachrichten steht eine CRC-Checksumme bereit, die Bestandteil jedes Frames ist. Diese ist ausreichend zur Identifizierung von Veränderungen während der Übertragung, jedoch kann sie eine Datenauthentizität nicht gewährleisten, da ganze Frames verloren gehen können. Über weitere Sicherheitsmechanismen verfügt das Netzwerk nicht.

2.2.8 Einordnung in das ISO/OSI-Modell

Aus der CAN-Spezifikation geht lediglich eine Implementierung der ISO/OSI-Schichten 1 (Bitübertragungsschicht) und Schicht 2 (Sicherungsschicht) hervor, jedoch gibt es bspw. Hersteller von Routern, die auf der Schicht 3 (Vermittlungsschicht) arbeiten. Außerdem existieren auch Implementierungen auf der Anwendungsschicht. In dieser Arbeit werden die Schichten eins bis drei von einem Treiber übernommen und daher als Black-Box betrachtet.

Anschaulich kann das in dieser Arbeit beschriebene Sicherheitskonzept als eine Art Transport Layer Security betrachtet werden, da es eine ähnliche Idee wie das gleichnamige TLS-Protokoll bezogen auf das Internet verfolgt. Auch hier soll es das Ziel sein die Sicherheit auf der Schicht 4 zu realisieren und der darüber liegenden Schicht 5 eine einheitliche Sicht auf die Daten zu bieten. Darüber hinaus soll die „Entscheidung“, welche Daten sicher übertragen werden, transparent zur Schicht 5 sein. Eine detailierte

Ausarbeitung dieser Problemstellung konnte im Rahmen dieser Arbeit jedoch nicht mehr geschehen.

2.3 Eigenschaften von Embedded Systems

In diesem Abschnitt soll zunächst definiert werden, was ein Embedded System ausmacht und speziell auf die Eigenschaften und Anforderungen bei der Entwicklung von Software für Embedded Systems eingegangen werden. Weiter sollen konkrete Probleme bei der Kommunikation zwischen Embedded Systems beschrieben und Lösungsmöglichkeiten abgesteckt werden.

2.3.1 Beschreibung eines Embedded Systems

„Der Ausdruck eingebettetes System (auch englisch embedded system) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist. Dabei übernimmt der Rechner entweder Überwachungs-, Steuerungs- oder Regelfunktionen oder ist für eine Form der Daten- bzw. Signalverarbeitung zuständig, beispielsweise beim Ver- bzw. Entschlüsseln, Codieren bzw. Decodieren oder Filtern.“ [Wik15b]

Embedded Systems übernehmen jedoch auch komplexere Aufgaben, wie die Steuerung von Sensor-Aktor-Netzwerken in Fertigungsanlagen oder Infotainmentsystemen im Auto. Hierbei kommunizieren sie mit einer Vielzahl von anderen Geräten beispielsweise über den CAN-Bus.

Die Haupteigenschaften eines Embedded Systems sind im Allgemeinen ein günstiger Stückpreis, eine deutlich geringere Performance gegenüber einen Personal Computer, ein geringerer Platzbedarf und ein kleinerer Energiebedarf. Daraus folgt zumeist auch eine komplett eigene Hardwarearchitektur.

Ein weiterer, wichtiger Aspekt von Embedded Systems ist, dass sie normalerweise über keine direkten Ein- und Ausgabegeräte verfügen, sondern diese gegenüber dem Anwender gekapselt erscheinen. [Wik15a]

2.3.2 Arbeitsspeicher

Embedded Systems verfügen prinzipiell über wenig Speicher, der bei komplexen Systemen, wie etwa einer Headunit im Auto, von vielen Prozessen oder Threads intensiv genutzt wird [WT06, S. 31ff]. Die zusätzliche Realisierung eines Kryptosystems beansprucht den Hauptspeicher des Systems stark. So wird bei der Initiierung einer sicheren Kommunikation eines Teilnehmers mit Hilfe eines RSA-Kryptosystems nach aktuellem Stand der Technik bis zu 4352Bit Speicher für die Schlüsselmasse benötigt.

Eine dynamische Speicherverwaltung, wie sie im Bereich der Desktop-PC eingesetzt wird, kommt in Embedded Systems selten zum Einsatz, da der Speicherverlust durch Fragmentierung, Overhead-Informationen und Alignment mit zunehmender Laufzeit erheblich wird. Außerdem ist es mit einer dynamischen Speicherverwaltung schwierig, Echtzeitanforderungen, wie sie in vielen Embedded Systems gelten, gerecht zu werden [WT06, S. 55ff].

Embedded Speicherkonzepte erfordern es häufig, den kompletten, benötigten Speicher beim Aufstarten des Systems zu reservieren. Bei dieser Praxis müsste die zentrale Instanz sehr viel Speicher für die Schlüssel bei paralleler Initiierung einer sicheren Kommunikation mehrerer Teilnehmer vorhalten oder mit der Regel, keine Deallokation zu betreiben, brechen.

Eine sinnvolle Alternative ist hier die Object-Pool-Allokierung, „die man benutzen kann, wenn die zu allokierenden Objekte alle feste und gleiche Größe haben und nach einer Zeit einzeln nicht mehr gebraucht werden“[WT06, S. 78], was bei dem Szenario der Authentisierung genau der Fall ist. Es wird je ein Pool für jeden Schlüsseltyp statisch allokiert, der zur Laufzeit dynamisch mit unterschiedlichen Schlüsseln gleicher Länge beschrieben werden kann. Hierbei muss beim Design des Systems eine sinnvolle[6] Tiefe des Pools gewählt werden.

So wird zum einen das Nachallokieren von Speicher zur Laufzeit vermieden und zum anderen kann eine sinnvolle Anzahl von parallelen Initiierungen der sicheren Kommunikation stattfinden.

2.3.3 Multithreadingeigenschaften

Ähnlich wie bei Speicher gelten für viele Embedded Systems Einschränkungen im Bereich des Multitaskings. So spezifiziert der OSEK-Standard, welcher häufig im Bereich von automotiven Echtzeitsystemen eingesetzt wird, dass alle Task- und Speicheranforderungen vor dem Kompilieren bekannt sind. [G+05]

Deshalb sollte die Anzahl der verwendeten Threads und deren Priorität bereits beim Design des Systems genau überlegt werden und auf die Eigenschaften des jeweiligen Prozessors abgestimmt werden.

So ist der Performancegewinn beim Auslagern von kryptografischen Algorithmen in eigene Threads nur dann sinnvoll, wenn das Ergebnis vorberechnet werden kann, wie z.B. das Berechnen eines symmetrischen Schlüssels zum Zeitpunkt der Anfrage einer sicheren Kommunikation. Hierbei ist wiederum zu beachten, dass die Vorberechnungen geringer priorisiert sind, als die Berechnungen auf dem kritischen Pfad der Authentifizierung, um diese nicht zu verzögern.

[6] „sinnvoll“ hängt in diesem Kontext von vielen Parametern wie Anzahl der Teilnehmer, Performance, Multithreading und Speicherverfügbarkeit ab.

2.3.4 Passwort / Pre-Shared Secret / Device Secret

Eine wichtige Frage, die sich beim Design eines Systems mit Bezug zu Sicherheitsthemen stellt, ist die Frage, wie sich die Kommunikationspartner gegenseitig authentifizieren. In Systemen mit einer Benutzerinteraktion ist dies gewöhnlicherweise ein Passwort oder eine PIN, die vom Nutzer eingegeben wird und von der Instanz, an die der Nutzer die Anfrage gerichtet hat, mit einem zuvor übermittelten Geheimnis verifiziert wird.

Dies ist in einem Embedded System natürlich nicht möglich, da im Allgemeinen keine direkten Eingabegeräte verfügbar sind und das Embedded Device gekapselt vom Anwender funktionieren soll. Daher ist es erforderlich, die Teilnehmer mit einem Geheimnis auszustatten. Dies kann in diesem Anwendungsfall nur beim Einrichten des Systems erfolgen.

So müssen einzelne Komponenten in einem Auto oder einer Fertigungsanlage mit einem Passwort ausgestattet und dieses der zentralen Steuereinheit mitgeteilt werden. Kommt ein Bauteil nachträglich ins System, so muss dieses ebenfalls mit der zentralen Steuereinheit bekannt gemacht werden. Um die Sicherheit zu gewährleisten, kann dies nur von einer autorisierten Person durchgeführt werden. Hier wäre z.B. eine Dongle-Lösung denkbar, bei der der Schlüsselbereich des Servers nur veränderbar ist, wenn eine zusätzliche Hardware (etwa ein USB-Stick) im System vorhanden ist.

In der Literatur wird anstelle eines Passworts hier auch der Begriff Pre-Shared-Secret oder Device Secret verwendet. [Eck13]

2.3.5 Hardwareunterstützung

Je nach Größe und Ausstattung der Embedded Systems verfügen diese über unterschiedliche Hardwarebausteine und Berechnungseinheiten, die für kryptografische Verfahren erforderlich sind oder diese wesentlich erleichtern.

Zufallszahlengenerator

Für die Generierung von starken Schlüsseln ist es wichtig „echte“ Zufallszahlen zu verwenden, da die Schlüssel direkt aus Zufallszahlen generiert werden. Diese kryptografisch sicheren Zufallszahlengeneratoren sind im Allgemeinen Hardwarebausteine, die z.B. auf der Basis von radioaktivem Zerfall beruhen. Jedoch verfügen nicht alle Embedded Devices über diese starken Zufallszahlengeneratoren und müssen ggf. eine Zufallszahl softwareseitig erstellen. Dieses Verfahren kann die Systemperformance und -sicherheit jedoch erheblich reduzieren [PP09, S. 34f].

Bei der Auswahl der Hardware sollte darauf geachtet werden, dass diese über einen geeigneten Zufallszahlengenerator verfügt, um die Möglichkeit, ein System durch das Erraten von Schlüsseln zu kompromittieren, auszuschließen.

Hardwareimplementierungen von Algorithmen

Hardwareimplementierungen von Algorithmen sind Implementierungen von Algorithmen, die direkt in dem Hardwarechip realisiert werden. Hardwareeinheiten für die AES Verschlüsselung und das SHA-Hashverfahren finden sich beispielsweise auf Chips, die für größere Embedded Systems geeignet sind [ARM14, Abs.: 2-2].

Der Einsatz von Hardware mit Kryptographieeinheiten beschleunigt die Ausführung erheblich und sollte vor allem für Komponenten mit zeitkritischen Aufgaben überdacht werden.

2.3.6 Zeitverhalten

Eine wichtige Frage ist, wann die sichere Kommunikation gestartet werden soll. Viele Embedded Sytems stellen strenge Zeit-Constraints [7] auf, um Sicherheitsanforderungen gerecht zu werden. Ein Automotive Embedded System muss beispielsweise nach zwei Sekunden voll gestartet sein, um z.B. den Fußgängerschutz durch Rückfahrkameras zu gewährleisten. Dies ist unter keinen Umständen mit der sicheren Kommunikation vereinbar, da allein der Schlüsseltausch ein Drittel der Startzeit beanspruchen würde. Hier müssen andere Strategien gefunden werden.

So ist zunächst eine unsichere Kommunikation, die sich zu einem sinnvollen Zeitpunkt in eine sichere wandelt, denkbar. Dies bringt jedoch die große Unsicherheit mit, dass es bereits zum Zeitpunkt der Authentifizierung zu spät sein kann, weil ein Bauteil in der Zeit bereits kompromittiert wurde. Außerdem ist der Zeitpunkt, *wann* ein sinnvoller Wechsel stattfinden kann, in zeitkritischen Eventsystemen u.U. nicht leicht festzulegen. Wird bspw. der Rückwärtsgang eingelegt, während eine Authentifikation stattfindet, vergeht auch hier wertvolle Zeit, um den Prozess abzuschließen oder abzubrechen.

Auch wenn diese Punkte wichtig sind, kann in dieser Arbeit aus zeitlichen Gründen nicht weiter darauf eingegangen werden. Weitere Ideen zu diesem Punkt werden in 6.3 beschrieben.

2.3.7 Starten der sicheren Kommunikation

Zunächst ist es offensichtlich, dass die zentrale Authorisierungsinstanz als erstes kommunikationsbereit sein muss, damit sie Teilnehmer authentifizieren kann. Dies kann leicht über eine Nachricht übermittelt werden.

Die Anfragen der Clients werden auf unterschiedlichen Object-Identifiern durchgeführt, was, wie in 2.2.4 beschrieben, dazu führt, dass die Authentifizierung mit der höchsten ID zuerst beim Server eintrifft und authentifiziert wird. Hierbei sollte im Client ein größerer Timeout vorgesehen werden als der normale Timeout bei einer Kollision, da der

[7] Ein Zeit-Constraint ist eine Anforderung an das System eine bestimmte Aufgabe innerhalb einer festgelegten Zeit abgeschlossen zu haben.

Authentifizierungsprozess wesentlich länger dauert als eine normale Nachricht. Dies gilt natürlich nur, wenn der Server Authentifizierungen ausschließlich sequenziell bearbeiten kann.

2.3.8 Vorberechnung von Schlüsseln

Grundsätzlich ist es möglich bestimmte Schlüssel, die nicht kollaborativ erstellt werden, vorzuberechnen, wodurch der Authentifizierungsprozess u.U. deutlich beschleunigt wird. Es kann jedoch nicht von den erreichten Zeiten mit vorberechneten Schlüsseln ausgegangen werden, da die zentrale Steuereinheit direkt nach der erfolgreichen Authentifierzierung eines Teilnehmers eine weitere Anfrage bekommt, noch bevor sie einen neuen Schlüssel vorberechnet hat (einen Singlethreaded Ansatz vorausgesetzt). Dadurch müsste der Teilnehmer genau so lange warten, wie bei nicht vorberechneten Schlüsseln. Auch muss sichergestellt sein, dass keine unvollständig berechneten Schlüssel genutzt werden, was vor allem bei der Auslagerung der Schlüsselberechnung in einen weiteren Thread wichtig ist.

Clientseitig führt die Vorberechnung von Schlüsseln jedoch zu einem direkten Performancegewinn, da dieser die Authentifizierungsanfrage erst stellt, wenn die Schlüssel berechnet wurden und somit Wartezeit in der zentralen Steuereinheit eingespart werden kann.

2.3.9 Ausfall von Kommunikationspartnern

Besonders automotive Embedded Systems sind während des Betriebs starken Störungen, wie Temperaturunterschieden oder elektromagnetischen Störungen ausgesetzt und müssen dennoch in nahezu jedem auf der Erde fahrbaren Bereich zuverlässig funktionieren.

Hierfür müssen die Systeme eine Robustheit aufweisen, die einen Ausfall eines Kommunikationsteilnehmers transparent zum Anwender erscheinen lässt und vor allem keinerlei Interaktion erfordert[8].

Ausfall eines Clients Der Ausfall eines Clients ist hierbei aus Sicht der sicheren Kommunikation eher unkritisch, da der Standard keine Zusagen hinsichtlich einer Verbindung macht [Bos91]. Lediglich die zusätzliche Zeit, um den Client zu reauthentifizieren, kommt beim Neustarten des Clients hinzu (siehe 2.3.7).

Der Fall eines Ausfalls des Clients, nachdem ein Teil einer Nachricht gesendet wurde, ist hier noch von größerer Bedeutung. Hierbei kann es, je nach Implementierung der Empfangsroutine, bei anderen Clients, zu Verklemmungen kommen. Diese erwarten den Rest der Nachricht des ausgefallenen Clients, welcher jedoch ausbleibt. Hier muss eine

[8] Dies gilt natürlich nur für Automotive Embedded Systems und nicht für Produktionsanlagen.

Routine implementiert werden, die timeout gesteuert, einen Request verschickt und ggf. die empfangene Teilnachricht verwirft.

Ausfall der zentralen Steuereinheit Der Ausfall des zentralen Steuerelements stellt ein wesentlich größeres Problem dar, als der Ausfall eines Teilnehmers. Für diesen Fall sollen drei denkbare Szenarien dargestellt werden. Welche Lösung eingesetzt wird, muss beim Design des Systems entschieden werden.

Zu den genannten Verfahren gelten zusätzlich die bei einem Ausfall eines Clients beschriebenen Schwierigkeiten. Die Anfrage eines Clients während des Ausfalls eines Servers stellt nur hinsichtlich der Zeit-Constraints ein Problem dar, da der Client nach einem Timeout seine Anfrage über eine sichere Kommunikation einfach wiederholen kann.

Reauthentifizierung aller Teilnehmer Der sicherste, jedoch auch aufwändigste Weg ist es, eine neue Kommunikation zu initiieren, indem alle Clients neu authentifiziert werden. Dies ist jedoch nur möglich, wenn die vorgegeben Zeitconstraints sehr große Spielräume ermöglichen.

Vogel-Strauß-Algorithmus Die einfachste Möglichkeit nach einem Ausfall ist die Vogel-Strauß-Technik. Hier wird sprichwörtlich der „Kopf in den Sand gesteckt“. Dies heißt in unserem Fall, dass die Steuereinheit wieder startet, jedoch keine weiteren Maßnahmen unternimmt. Dieses Verfahren kann nur in statischen Systemen nach Authentifizierung aller Teilnehmer und auch nur bis zum nächsten Key-Update (vgl. Kapitel 3)erfolgreich sein. Denkbar wäre der Vogel-Strauß-Algorithmus in Verbindung mit einer Reauthentifikation aller Teilnehmer bei einem Key-Update. [Tan09, S.520]

Protokollierung der Kommunikationen Eine weitere Möglichkeit besteht darin, dass die Steuereinheit alle Authentifizierungen und Schlüssel protokolliert und nach einem Neustart wieder einliest. Dieses Verfahren muss jedoch aus kryptografischer Sicht genau untersucht und ggf. weiter abgesichert werden, um die Sicherheit der Schlüssel zu gewährleisten. Zudem entsteht sowohl bei der Authentifizierung als auch beim Neustart der Steuereinheit ein weiterer Overhead für das Speichern und Lesen der Schlüssel.

Mögliche Mittel und Verfahren, um dies zu realisieren, konnten im Rahmen dieser Arbeit jedoch nicht mehr recherchiert werden.

2.4 Allgemeines Kommunikationsmodell

Um eine sichere Kommunikation zwischen beliebig vielen Busteilnehmern durchzuführen, ist ein grundsätzliches Konzept zur Kommunikationsstrategie, unabhängig von einer konkreten Implementierung oder gewählten kryptografischen Algorithmen, erforderlich.

In diesem Kapitel sollen die Voraussetzungen beschrieben und ggf. Alternativen erwähnt werden.

2.4.1 Client - Server Architektur

Aufgrund der gegeben Anforderungen an das System bietet sich das Clients - Server Modell an. Hierbei werden nach einer beim Design des Systems festgelegten Abfolge, Algorithmen durchlaufen. Dessen Ergebnis ist die Authentifizierung eines Clients, gegenüber einem Server und die Zuordnung zu einer Kommunikationsgruppe.

Server

Um eine sichere Kommunikation mehrerer Teilnehmer zu initiieren und durchzuführen, ist es notwendig, eine Instanz zu haben, die in der Lage ist Teilnehmer authentifizieren. Dies übernimmt in der gewählten Architektur das Hauptsteuergerät im Bus, welches im Folgenden Server genannt wird. Der Server gilt per se als vertrauenswürdige Instanz im Netzwerk und stellt den Dienst der sicheren Kommunikation bereit. Das heißt, ein Teilnehmer, der sich authentisieren möchte, wendet sich immer an den selben Server. Wird der Server kompromittiert, so ist das komplette Netzwerk als unsicher anzusehen. Der Server hält sämtliche Passworte aller Teilnehmer sowie eine Domain - Client Zuordnung.

Außerdem ist der Server für die ggf. zyklische Generierung und Verteilung der Domainschlüssel verantwortlich.

Weitere Aufgaben, wie das Loggen von Authentifizierungsvorgängen und das Führen von Blacklisten für auffällige Teilnehmer, sind denkbar.

Der Server nimmt selbst nicht an der sicheren Kommunikation teil, um eine Kompromittierung über diesen Weg auszuschließen. Es ist jedoch möglich, auf der gleichen Hardware sowohl Client als auch Server zu betreiben.

Clients

Busteilnehmer, die an der sicheren Kommunikation teilnehmen, werden als Clients bezeichnet. Sie fordern den bereitgestellten Dienst der sicheren Kommunikation an und authentisieren sich gegenüber dem Server. Das Ergebnis des Verfahrens ist die Übergabe des Domainschlüssels, womit der Client berechtigt ist, in der Gruppe zu kommunizieren.

Ein Client kennt lediglich die Gruppen, in der er sicher kommunizieren möchte, inklusive der für ihn interessanten CAN-IDs zum Senden und Empfangen. Es wäre auch denkbar, diese Informationen nach einer erfolgreichen Authentisierung vom Server mitgeteilt zu bekommen.

2.4.2 Domainprinzip

Damit eine Kommunikation zwischen beliebig vielen Clients möglich ist, wird ein Domainprinzip eingeführt. So soll es z.B. möglich sein, anhand der Kommunikationsmatrix festzulegen, ob drei Kommunikationspartner in einer oder in drei Gruppen (Domains) kommunizieren können. Eine dynamische Änderung der Gruppierungen ist jedoch nicht vorgesehen.

Um in einer Gruppe sicher kommunizieren zu können, ohne die Identität aller Empfänger feststellen zu müssen, wird eine „Schlüsseldomain“ für jede Gruppe erzeugt. Das heißt, jeder Teilnehmer, der Mitglied in dieser Gruppe ist und sich erfolgreich authentifiziert hat, erhält den gleichen Domainschlüssel. Somit kann er die verschlüsselten Nachrichten der anderen Teilnehmer entschlüsseln und selbst in der Gruppe kommunizieren.

Es ist ohne weiteres möglich, dass ein Client Mitglied in zwei Domains ist, wobei der Authentifizierungsvorgang natürlich für jede Domain separat durchzuführen ist. Etwaige „Interessenkonflikte“ im Sinne der Weiterleitung von sensitiven Daten zwischen den beiden Domains müssen beim Design des System bedacht werden.

Prinzipiell wäre auch ein Subdomainmodell denkbar, in dem eine Domain eine oder mehrere Subdomains enthalten kann und ein Client erst in der Oberdomain authentifiziert worden sein muss, um die Subdomain zu betreten. Jedoch erhöht sich die Kommunikationsaufwand hierdurch immens. Auch dies sollte beim Design eines Systems abgewogen werden.

Einschränkung der Kommunikationsmatrix

Um das Domainprinzip umsetzen zu können, muss jedoch die Kommunikationsmatrix angepasst werden. Um den Grundsatz, dass eine CAN-ID nur einem von Teilnehmer versendet werden darf, zu erfüllen, braucht jeder Client eine eigene CAN-ID, um die sichere Kommunikation mit dem Server zu initiieren. Umgekehrt benötigt der Server für jeden Client eine CAN-ID, um zu antworten. [Bos91]

Ein Client benötigt jedoch nur eine CAN-ID für den Aufbau der sicheren Kommunikation, und nicht für jede Domain eine eigene, da das Datenfeld der Kommunikationsanfrage für die Identifizierung der Domain genutzt werden kann.

IDs, die nicht für eine sichere Kommunikation verwendet werden, können weiterhin als unverschlüsselte Kommunikationswege genutzt werden.

Beispiel einer Kommunikationsmatrix In der Tabelle 2.1 ist eine beispielhafte Kommunikationsmatrix mit drei Clients, die in zwei Domains kommunizieren, und einem Server dargestellt, um die Zusammenhänge zu erläutern. 2.1 zeigt eine dazu passende schematische Darstellung.

Hierbei ist Client 1 in beiden Domains angemeldet, der Client 2 in Domain 1 und der Client 3 in Domain 2. Der Server ist, wie bereits beschrieben, kein Mitglied einer Domain.

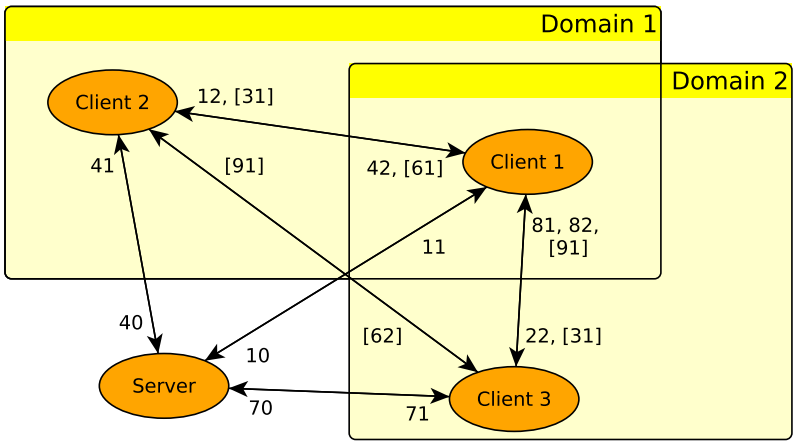


Abbildung 2.1: Schematische Darstellung der beispielhaften K-Matrix

Er lauscht auf die 10, 40 und 70, auf denen die jeweiligen Clients Anfragen für eine sichere Kommunikation stellen können. Die Antwort sendet der Server nun auf der um eins erhöhten CAN-ID. Die Initiierung der sicheren Kommunikation findet ausschließlich auf diesen beiden IDs statt, auch wenn während dieser Phase verschiedene Nachrichtentypen mit der gleichen ID übertragen werden (s.u.). Innerhalb der jeweiligen Domain können die Clients nach der Initiierung auf den ihnen zugewiesenen IDs beliebig empfangen und senden.

	ID_SEND				ID_RECEIVE			
	SECREQ	DOM1	DOM2	PLAIN	SECRES	DOM1	DOM2	PLAIN
Server	11,41,71				10,40,70			
Client 1	10	12, 13	21, 22	[31]	11	42	81, 82	[61], [91]
Client 2	40	42		[61], [62]	41	12		[31], [91]
Client 3	70		81, 82	[91]	71		22	[51], [61]

Tabelle 2.1: Tabellarische Übersicht der beispielhaften K-Matrix

Dynamischere Ansätze Durch den Server als zentrale Verwaltungsinstanz ist es denkbar, Clients innerhalb der sicheren Kommunikation dynamisch Sende- und Empfangs-IDs zuzuweisen, was interessante Anwendungsfälle mit sich bringt.

Auswirkungen auf die Priorisierung

Wie in 2.2.4 beschrieben, wird über die CAN-ID auch die Arbitierung auf dem Bus und so auch Priorisierung der Nachrichten realisiert. In Zusammenhang mit einem großen Datenaufkommen, vor allem bei der Initiierung der sicheren Kommunikation, kann es

zum Verwerfen von zahlreichen Nachrichten kommen. Authentifiziert sich ein Client mit einer ID geringer Priorität, so kann es zu erheblichen Verzögerungen bei der Nachrichtenübertragung kommen.

2.4.3 Kommunikationsprotokoll

Da es unterschiedliche Nachrichten mit unterschiedlichen Längen und Aufgaben in der Initiierung der sicheren Kommunikation gibt, ist es erforderlich ein Kommunikationsprotokoll einzuführen, welches eine strukturierte Übertragung langer Nachrichten ermöglicht. Dieses Protokoll ist unterhalb der CAN-IDs angesiedelt und nutzt ausschließlich das Datenfeld des CAN-Frames.

Der Overhead eines solchen Protokolls sollte möglichst gering sein, da die Bandbereite des CAN-Busses ohnehin recht gering ist. Im Wesentlichen bieten sich zwei verschiedene Protokolle an:

Präfixprotokoll

Bei dem Präfixprotokoll werden alle für die Nachricht relevanten Verwaltungsinformationen am Anfang der Nachricht übertragen. Diese Informationen sind im Falle des Prototypen die ID der übertragenen Nachricht, die dem Empfänger signalisiert, wie er die enthaltenen Daten interpretieren muss. Außerdem enthält er die Anzahl der folgenden Daten in Byte, worauf die Datenmasse des Frames folgt.

Der Vorteil dieses Verfahrens ist, dass sehr wenig Verwaltungsoffset entsteht, welcher sich genau vorhersagen lässt. Nachteilig ist, dass mit der Größe des Längenfeldes eine feste Obergrenze an Nachrichtenlänge bestimmt wird.

Streamprotkoll

Ein weiterer Ansatz einer Protokollierung ist ein Streamprotokoll nach dem Vorbild des High-Level Data Link Control (HDLC)-Formates. Hierbei werden Bitmuster als Signal für den Anfang und das Ende der Nachricht genutzt (z.B. 01110). Um die Nachricht nicht vorzeitig abzubrechen, muss der Sender ein Bitstuffing vornehmen, indem er nach jeder zweiten direkt aufeinander folgenden 1 eine 0 einfügt. Der Empfänger löscht diese Nullen wieder um die ursprüngliche Nachricht zurück zu gewinnen.

Das Streamprotokoll kann im Gegensatz zum Präfixprotokoll eine beliebig lange Nachricht übertragen, die auch bei der Initierung der Übertragung noch nicht feststehen muss. Dies ist jedoch mit einem schwer absehbaren Offset verbunden, der abhängig von den Eingabedaten ist. Daher sollte dieses Verfahren nur dann eingesetzt werden, wenn sehr große Datenmengen über CAN übertragen werden sollen, wie z.B. beim Flashen von Daten auf Geräte, die nur über CAN erreicht werden können.

Weitere Ansätze

Ansätze, wie z.B. eine Base64 Codierung der Datenmasse, würden zwar eine Protokollierung der Länge überflüssig machen, da sie sich einfach nullterminieren. Dies führt jedoch ebenfalls zu einem erheblichen Overhead an zu übertragenen Daten und ist daher ungeeignet.

Kapitel 3

Vorschlag eines Sicherheitskonzeptes

In diesem Kapitel soll ein Vorschlag für ein möglichst hohes Sicherheitsniveau erstellt werden, welches sich in das Architekturmodells aus Abschnitt 2.4 einbettet. Dazu werden die in 2.1.2 beschriebenen Verfahren so zusammengefügt, dass sich eine durchgängige Sicherheitskette bildet. Eine schematische Darstellung der erforderlichen Schritte befindet sich außerdem im Anhang 1. Das Schema wird in diesem Kapitel als „roter Faden“ genutzt, indem die Schritte hier in der Reihenfolge beschrieben werden und hier auf das Schema referenziert wird. Außerdem soll zu jedem Verfahren kurz auf mögliche Optimierungen und deren Limitierungen eingegangen werden.

Vorbemerkung zu diesem Kapitel

Auch wenn die in diesem Kapitel entwickelten Verfahren sorgfältig geprüft wurden, kann im Rahmen dieser Arbeit keine Verifizierung des Systems geboten werden. Dies folgt allein aus der Regel, dass ein kryptografisches System nicht von demjenigen verifiziert werden kann, der es entwickelt hat. Es bleibt daher dem Leser überlassen die Richtigkeit und Vollständigkeit des vorgestellten Sicherheitskonzepts zu überprüfen.

3.1 Voraussetzungen und grundsätzliches Vorgehen

Um die in diesem Kapitel beschriebenen Sicherheitsziele umzusetzen, müssen einige Voraussetzungen erfüllt sein, die im Kapitel 2 aufgeführt sind und hier noch mal kurz zusammengefasst werden sollen.

Kryptografisch starke Verfahren Um ein Sicherheitsniveau nach dem Stand der Technik zu gewährleisten ist es essentiell, die eingesetzten Verfahren und die Länge der verwendeten Schlüssel nach aktuellen Erkenntnissen auszuwählen.

Sinnvolles Zusammenspiel der Verfahren Wie bereits im Abschnitt 2.1 beschrieben, verfügen symmetrische und asymmetrische Verfahren über unterschiedliche Vor-

und Nachteile. Daher werden asymmetrische Verfahren zur Initiierung eingesetzt, um die Eigenschaften der öffentlichen und privaten Schlüssel auszunutzen. Symmetrische Verfahren werden dagegen für die Kommunikation eingesetzt, da sie über bessere Performance verfügen.

Erst Verschlüsseln dann Signieren Um eine Kompromittierung durch Schadcode auszuschließen ist es erforderlich, dass jede Kommunikation zunächst verschlüsselt und danach signiert wird. So kann Schadcode vor der Entschlüsselung entdeckt werden und keinen Schaden anrichten.

Schlüssel Zunächst folgt aus dem Abschnitt 2.1.3, dass für jedes Verfahren jeweils einen eigener Schlüssel benötigt wird. Das heißt, dass sowohl Server als auch Client einen Schlüssel für Verschlüsselung und Signatur der Initiierung erstellen müssen. Der Server muss außerdem die Schlüssel für die Domain bereitstellen, die ebenfalls unterschiedlich sein müssen. Der Client berechnet also zwei Schlüssel, der Server insgesamt vier. Die Schlüssellängen und deren Generierung muss aus kryptografisch starkem Zufall erfolgen und hinreichend lang sein.

Masterpasswort Um ein sicheres Verfahren zu gewährleisten, ist es erforderlich, ein Preshared Secret oder Masterpasswort vorab auf sicherem Wege auszutauschen.

3.2 Vorschlag für Algorithmen

Grundsätzlich eignet sich eine Vielzahl von Algorithmen zur Umsetzung des Sicherheitsniveaus, wobei der Ablauf der einzelnen Verfahren teilweise stark unterschiedlich sein kann. Daher wird ein Vorschlag für ein möglichst hohes Sicherheitsmaß während der Initiierung am Beispiel des RSA-Kryptosystems und einem System, basierend auf elliptischen Kurven, erläutert. Für die sichere Kommunikation wird der Advanced Encryption Standard (AES) als standardisiertes Verfahren alternativlos betrachtet. Für weitere, mögliche Verfahren wird auf die Literatur (insbesondere [BSI14]) verwiesen.

Hintergrundverfahren, wie Hashfunktionen oder Zufallsgeneratoren, können im Allgemeinen beliebig ausgetauscht werden, sofern sie den jeweiligen Sicherheitsanforderungen genügen.

3.2.1 Einsatzzweck der ausgewählten Algorithmen

Die Tabelle 3.1 zeigt eine Übersicht der in diesem Kapitel eingesetzten Verfahren in Verbindung mit deren Einsatzzweck. Hieraus wird ersichtlich, dass für die sichere Kommunikation lediglich der AES betrachtet wird, der sowohl beim Verschlüsseln als auch beim Signieren als CMAC zum Einsatz kommt.

Bei der Initiierung der Kommunikation wird hingegen das RSA-Verfahren dem auf elliptischen Kurven beruhenden Verfahren ECDH zum Erstellen eines gemeinsamen, symmetrischen Schlüssels und dem ECDSA zur Signaturbildung gegenüber gestellt. Bei dem Verfahren mit ECDH wird nach dem Erstellen des Geheimnisses ebenfalls der AES zum Verschlüsseln gewählt. Es ist jedoch wichtig zu sehen, dass hier keine Message Authentication Codes für die Signatur verwendet werden können, da diese nur Datenauthentizität liefern, jedoch eine Instanzauthentizität gefordert wird.

Das Mischen von RSA und ECC ist theoretisch möglich, macht in diesem Anwendungsfall jedoch keinen Sinn und wird somit nicht betrachtet.

Einsatzzweck	Klasse	Algorithmus	Schlüssellänge
Initiierung	Asymmetrische Verschlüsselung	RSA / ECDH	2048 / 160
Initiierung	Asymmetrische Signatur	RSA / ECDSA	2048 / 160
Kommunikation	Symmetrische Verschlüsselung	AES	256
Kommunikation	Symmetrische Signaturver	AES	256
Initierung und Kommunikation	Hashenverfahren	SHA2	256

Tabelle 3.1: Überblick über die eingesetzten kryptografischen Verfahren und deren Schlüssellängen

3.3 Vorbereitungen

Um eine Initierung der sicheren Kommunikation zu beschleunigen, können die eingesetzten Schlüssel vorberechnet werden. Das Szenario geht davon aus, dass alle Schlüssel, die vorberechenbar sind, zur Initiierung zur Verfügung stehen.

Zudem ist ein gemeinsames Geheimnis im Vorfeld ausgetauscht worden, was als Master-Secret genutzt wird.

3.3.1 Vorberechnung der Schlüssel

Als ersten Schritt, noch bevor eine Anfrage einer sicheren Kommunikation eintrifft, berechnet der Server zwei symmetrische Schlüssel mit einer Länge von 256 Bit. Hierbei dient einer zur Verschlüsselung der Daten und der andere zu deren Signierung. Zudem wird ggf. je ein Initialisierungsvektor benötigt. Dies ist der Schritt *AuthRSA1* bzw. *AuthEC1* in der schematischen Darstellung, wobei Schlüssel und Initialisierungsvektor (IV) zusammengefasst werden.

Rivest, Shamir, Adelman Kryptoverfahren (RSA)

Sofern bei der Initiierung der RSA zum Einsatz kommt, werden im Rahmen der Vorbereitung vom Client und vom Server je zwei RSA-Schlüsselpaare erzeugt (Schritt *AuthRSA2*). Diese Schlüssel müssen für ein akzeptables Sicherheitsniveau mindestens 2048 Bit lang sein und von einem Zufallsgenerator erstellt werden. Auch hier ist ein Schlüssel für die Verschlüsselung vorgesehen, der andere dient Instanzauthentifizierung durch die RSA-Signatur.

Elliptic Curves Digital Signature Algorithm (ECDSA)

Bei der Wahl des ECDSA zum Erstellen von digitalen Signaturen kann der Schlüssel ebenfalls vorberechnet werden. Hierfür wird jedoch ein wesentlich kleinerer Schlüssel von 160 Bit benötigt (Schritt *AuthEC2*).

3.4 Initiierung der sicheren Kommunikation

Wie bereits beschrieben wird die Initialisierung eines Clients lediglich einmal durchgeführt. Hierzu sendet ein Client, der sich gegenüber dem Server authentisieren möchte, eine Anfrage für eine sichere Kommunikation (sec_com_req im Schritt *Auth3*). Diese wird mit dem privaten Schlüssel des Clients (sk_{c1a}) signiert, damit eine Veränderung der Anfrage, z.B. Änderung der angefragten Domain, ausgeschlossen werden kann. Damit der Server diese und weitere Anfragen verifizieren kann, sendet der Client den dazugehörigen öffentlichen Schlüssel pk_{c1a} mit.

3.4.1 RSA-Kryptosystem

Außerdem sendet der Client im Falle des RSA-Kryptosystems den öffentlichen Schlüssel pk_{c1e} zum Verschlüsseln einer Nachricht für den Server mit der Kommunikationsanfrage (ebenfalls *AuthRSA3*).

Hat der Server die Anfrage zur sicheren Kommunikation empfangen, so sendet er seine beiden öffentlichen Schlüssel pk_{sa} und pk_{se} an den Client. Außerdem sendet er eine Authentifizierungsanfrage (auth_req) kombiniert mit einer Nonce, welche mit dem privaten Schlüssel des Servers verschlüsselt ist [1] (*AuthRSA4*).

Der Client verifiziert diese mit dem erhaltenen öffentlichen Schlüssel des Servers, womit er von der Echtheit des Authentifizierungsrequests ausgehen kann. Anschließend entschlüsselt er den auth_req und die Nonce (*AuthRSA5*). Diese konkateniert er mit seinem Passwort und bildet einen Hashwert über dem Ergebnis. Der Client sendet nun

[1] Der auth_req selbst müsste nicht verschlüsselt werden, der er keine sensitiven Daten enthält. Sofern die Nonce nicht größer als die Länge des asymmetrischen Schlüssels ist, macht dies jedoch keinen Unterschied.

die berechneten Daten zurück an den Server (*AuthRSA6*). Eine Verschlüsselung dieser Nachricht ist nicht erforderlich, solange eine kryptografisch starke Hashfunktion zum hashen der Daten verwendet wird, da die Ursprungsdaten hieraus nicht zurückgewonnen werden können.

Dieser verifiziert die Daten mit seinem geheimen Signaturschlüssel und kann anschließend mit Hilfe der Nonce und dem Passwort ebenfalls das Geheimnis berechnen und somit die vom Client gesendeten Daten verifizieren (*AuthRSA7*).

In Schritt *AuthRSA8* versendet der Server die symmetrischen Domainschlüssel zur Verschlüsselung und Authentifizierung in der angefragten Domain. Diese werden wiederum zunächst mit dem öffentlichen Schlüssel des Clients verschlüsselt und danach mit dem eigenen privaten Schlüssel signiert. Die verschlüsselten Schlüssel werden vom Server nacheinander an den Client gesendet.

Die eingesetzten, symmetrischen Verschlüsselungverfahren und Authentifikationsverfahren benötigen u.U. darüber hinaus noch einen Initialisierungsvektor. Dieser kann als eigene Nachricht versendet werden, wobei hier das gleiche Verfahren wie bei den Schlüsseln angewendet werden sollte oder kann mit Hilfe eines Verfahrens zur Schlüsselableitung aus dem übertragenen Schlüssel hergeleitet werden.

Schließlich verifiziert der Client die empfangenen Daten jeweils mit seinem geheimen Signaturschlüssel und entschlüsselt sie mit dem öffentlichen Schlüssel des Servers. Damit ist der Vorgang der Authentifizierung abgeschlossen und der Client kann mit den zur Verfügung gestellten Schlüsseln in der angefragten Domain sicher kommunizieren.

3.4.2 Elliptic Curves Cryptography

Die Authentifizierung mit Hilfe von elliptischen Kurven unterscheidet sich im wesentlichen durch das Generieren eines gemeinsamen Geheimnisses vom RSA-Kryptosystem.

Nachdem auch hier der in Unterabschnitt 3.3.1 beschriebene Punkt ausgeführt wurde, wird in Schritt *AuthEC3*, wie bereits beim RSA, vom Client eine Anfrage zur sicheren Kommunikation gesendet, welche mit einer Signatur versehen wird. Einen asymmetrischen Schlüssel zur Verschlüsselung gibt es jedoch nicht.

Der Server verifiziert diese Anfrage mit dem erhaltenen öffentlichen Schlüssel (*AuthEC4*)

Im nächsten Schritt beginnt der Server mit dem Generieren eines gemeinsamen Geheimnisses, welches er gemeinsam mit dem Client entwickelt. Hierzu generiert er seinen Teil des Shared Secret und sendet dies an den Client (*AuthEC5*).

Der Client führt das Generieren fort, indem er auf Basis des erhaltenen Teilgeheimnisses sowohl das gesamte Shared Secret als auch sein Teilgeheimnis berechnet. Das Teilgeheimnis des Clients wird wiederum an den Server übertragen, damit dieser in der Lage ist, ebenfalls das Shared Secret zu berechnen (*AuthEC6*). Um aus dem Shared Secret einen validen Schlüssel zu erzeugen, sollte dieses mit Hilfe einer Key Derivation

Function abgleitet werden. Falls erforderlich muss ebenfalls ein Initialisierungsvektor aus einem anderen Teil des Shared Secrets abgeleitet werden. Diese Funktion muss natürlich im Client und im Server gleich sein, damit sie zu einem identischen Ergebnis kommen. Client und Server haben nun einen gemeinsamen, symmetrischen Schlüssel, der für die Verschlüsselung eingesetzt werden kann.

Auf Basis des eigenen und des erhaltenen Teilgeheimnisses berechnet der Server ebenfalls das volle Shared Secret. Beide Teilgeheimnisse müssen bei diesem Verfahren nicht verschlüsselt oder signiert werden, da die Schlüsseleinigung mit veränderten Werten kein valides Ergebnis liefert und von den Teilnehmern wiederholt wird.

Die folgenden Schritte sind sehr ähnlich zu der Version auf Basis des RSA, abgesehen davon, dass hier für die Verschlüsselung der Daten ein symmetrisches Verfahren eingesetzt wird.

In Schritt *AuthEC7* sendet der Server eine Authentisierungsanfrage kombiniert mit einer Nonce, die wie beschrieben symmetrisch verschlüsselt, jedoch asymmetrisch authentifiziert ist. Dies wird in *AuthEC8* vom Client symmetrisch entschlüsselt und asymmetrische authentifiziert. Auf Basis der Nonce und des Master Secrets bildet der Client einen Hashwert, welcher signiert wird und an den Server gesendet wird (*AuthEC9*). Der Server gleicht die Antwort mit dem von ihm ermittelten Hashwert auf Nonce und Master Secret ab (*AuthEC10*) und beginnt darauf hin die verschlüsselte und signierte Übertragung des Domainschlüssels (*AuthEC11*). Der Client entschlüsselt und verifiziert die gesendeten Daten und installiert schließlich die Schlüssel, womit er zur Kommunikation in der angefragten Domain berechtigt ist.

3.4.3 Optimierung der Verfahren

Aufgrund des voraussichtlich hohen Zeitbedarfs der vorgestellten Initiierungsverfahren stellt sich die Frage, ob diese optimiert werden können, um sie performanter zu gestalten.

Dabei kommt eine Verringerung der Schlüssellänge nicht in Frage, da diese zu einer leichteren Angreifbarkeit des System führt. So wurden bereits erfolgreich Angriffe auf RSA1024 durchgeführt.

Auch das Verwenden von Pre Shared Keys[2] geht mit einem massiven Verlust an Sicherheit einher. Ein Angreifer hat hier quasi unbegrenzt lange Zeit einen Schlüssel zu brechen und erhält mit steigender Laufzeit immer mehr Informationen, die ihm dabei helfen.

Optimierung, beispielsweise durch Weglassen der Authentifizierung einzelner Nachrichten, gewährleistet keine Integrität mehr und ist daher aus kryptografischer Sicht indiskutabel.

[2]In Abgrenzung zu Pre Shared *Secrets* werden hier asymmetrische Schlüssel vorab verteilt und direkt verwendet.

3.5 Key Update aller Clients einer Domain

Um ein möglichst hohes Sicherheitsniveau zu realisieren, ist es erforderlich, die verwendeten Domainschlüssel zyklisch und bei jedem Eintritt eines Clients in die Domain auszutauschen.

Hierzu muss zunächst die komplette Kommunikation in der Domain gestoppt werden. Dazu sendet der Server eine Anfrage zum Stoppen an alle Clients der Domain. Diese sollte mit dem privaten Schlüssel des Servers signiert werden, wie in Schritt *KeyUpdate1* gezeigt. Daraufhin unterbrechen die Clients die sichere Kommunikation, um ihr Key Update durchzuführen[3] (*KeyUpdate2*).

Der Server signiert und verschlüsselt die neu generierten Schlüssel für die Verschlüsselung und Authentifizierung, wie bereits in Schritt *AuthRSA8* beschrieben und sendet diese an alle Clients der Domain (*KeyUpdate3*).

Die Clients verifizieren die Nachricht mit dem öffentlichen Schlüssel des Servers und entschlüsseln sie mit ihren jeweiligen privaten Schlüsseln (*KeyUpdate4*) und ersetzen die aktellen Schlüssel mit den erhaltenen (*KeyUpdate5*).

3.5.1 „Perfektes“ Key Update

Das zuvor beschriebene Verfahren bietet jedoch noch keine Perfect Foreward Secrecy. Hierzu ist es erforderlich, mit jedem Client der Domain zuvor neue Langzeitschlüssel auszuhandeln, woraus eine komplette Reauthentifizierung (also *AuthEC1* bis *AuthEC12*) *aller* Clients folgt. In Abhängigkeit der Domaingröße wird dies je nach Wahl der Nachrichtenprioritäten zu einer erheblichen Verzögerung der Kommunikation in der Domain oder auf dem gesamten Bus führen.

3.5.2 Einfacheres Keyupdate über die alten Sitzungsschlüssel

Ein weiterer Weg des Schlüsselupdates wäre es, die neuen Sitzungsschlüssel mit den aktuellen Sitzungsschlüsseln zu verschlüsseln und zu authentifizieren. Dies ist möglich, da der Server diese erstellt und verteilt, und wesentlich effizienter, da den Sitzungsschlüsseln ein symmetrisches Verfahren zugrunde liegt. Allerdings lässt sich bei dem Vorgehen aus einem gebrochenen Sitzungsschlüssel der nächste Sitzungsschlüssel ableiten und somit kann weder aclPFS noch die bloße Foreward Secrecy gewährleistet werden.

3.5.3 Notwendigkeit von Key Updates bei kurzer Betriebszeit

Angesichts dieser Einschränkungen einerseits hinsichtlich der Sicherheit, andererseits der Realisierbarkeit, muss die Frage gestellt werden, ob die Forderung der PFS und der dar-

[3] Wie genau dies geschieht, z.B. über Acknowledgement der Clients an den Server oder einen Timeout, muss im konkreten Fall bedacht werden.

aus resultierende Aufwand gerechtfertigt ist. Vor allem in Systemen mit einer begrenzten Betriebszeit sollte dies genau abgewogen werden, was im Rahmen dieser Arbeit nicht möglich ist.

3.6 Sichere Kommunikation

Wie bereits beschrieben, wird die sichere Kommunikation zwischen Clients in einer Domain mit symmetrischen Mitteln realisiert und ist in jeder Hinsicht deutlich simpler, als die anderen Verfahren.

Dazu wird im sendenden Client 1 eine Nachricht zunächst mit dem Domainschlüssel zur Verschlüsselung verschlüsselt. Über diese Daten wird eine Datensignatur gebildet (Schritt: *SecCom1*). Dies geschieht mit Hilfe des symmetrischen Schlüssels zum Bilden eines Message Authentication Code. Die verschlüsselte Nachricht wird konkateniert mit der Signatur an die anderen Clients gesendet (*SecCom2*). Die empfangenden Clients bildeen ebenfalls einen MAC über die verschlüsselten Daten und vergleichen diese mit dem empfangenen MAC. Schließlich werden die Daten mit dem symmetrischen Schlüssel zur Verschlüsselung [4] entschlüsselt und weiterverarbeitet (*SecCom3*).

3.6.1 Optimierung der Kommunikation

Bei der sicheren Kommunikation besteht nach aktuellen Erkenntnissen keine Möglichkeit einer Optimierung der Laufzeit oder zusätzlichen Buslast. Beide Verfahren sind essentiell notwendig, um die Sicherheit des Systems zu gewährleisten. Auch eine Reduzierung der Schlüssellänge würde aufgrund des symmetrischen Verfahrens voraussichtlich zu keiner nennenswerten Beschleunigung führen.

[4] Da es sich um ein symmetrisches Verfahren handelt, kann mit dem Schlüssel zur Verschlüsselung auch entschlüsselt werden.

Kapitel 4

Prototypische Implementierung

Um die Betrachtungen, die in Kapitel 3 beschrieben werden, zu entwickeln und einen Anhaltspunkt über deren Realisierbarkeit und zusätzlichen Aufwand[1] zu bekommen, wurde ein Prototyp aus dem allgemeinen Kommunikationsmodell aus Abschnitt 2.4 abgeleitet. In diesem Kapitel soll die erstellte prototypische Implementierung beschrieben und auf Besonderheiten und Limitierungen eingegangen werden.

4.1 Zentrale Anforderung

Um eine möglichst realistische Implementierung zu erreichen, wurden im Vorfeld folgende Anforderungen im Rahmen einer kurzen Anforderungsanalyse erarbeitet, die bei der Entwicklung des Prototypen berücksichtigt werden sollen:

> *Zentrale Anforderung an die prototypische Implementierung ist es, eine belastbare Aussage darüber treffen zu können, ob es effizient möglich ist, eine verschlüsselte und authentische Kommunikation beliebig vieler Teilnehmer über CAN durchzuführen und welche Mittel dafür eingesetzt werden müssen.*

4.1.1 Weitere Anforderungen

Weitere Anforderungen sollen außerdem berücksichtigt werden, um mit Hilfe deren eine Aussage über die Realisierbarkeit zu treffen.

Anforerungen:

- Die Zeit zur Initiierung einer verschlüsselten Kommunikation soll den Systemstart und auch den Start der Netzwerkkommunikation nicht nennenswert verzögern.

[1] „Aufwand“ beschreibt in diesem Zusammenhang die Zeit, die zusätzlich benötigt wird und die benötigten CAN-Frames.

- Durch die verschlüsselte Kommunikation soll keine nennenswerten Verzögerungen in der Übertragung einer Nachricht entstehen. Auch die Buslast soll sich nicht drastisch erhöhen.
- Weiter ist eine wichtige Anforderung, den Prototypen so zu entwickeln, dass er auf Embedded Systems lauffähig ist. Besonders zu beachten ist dabei der Speicherverbrauch und die Multithreadingeigenschaften der Anwendung.
- Hinsichtlich der Implementierung ist eine Programmiersprache zu wählen, die weitreichende Kontrollmöglichkeiten bezüglich der Regeln der embedded Programmierung bietet.
- Der Prototyp soll so plattformunabhängig wie möglich gestaltet werden. Insbesondere bei der Auswahl der Bibliotheken ist dies zu beachten.
- Kommunikationsprotokolle sollen leicht austauschbar gestaltet werden.
- Der Prototyp soll die Möglichkeiten von CAN voll ausnutzen können. Dabei sollen beliebig viele Teilnehmer mit beliebig vielen Untergruppen verschlüsselt kommunizieren können. (M:N-KOmmunikation)

Von der Ausarbeitung eines umfangreichen Lastenheftes wurde jedoch aus zeitlichen Gründen abgesehen. Außerdem wurde auf weitere Anforderungen, wie z.B. im Bereich der Benutzerinteraktion oder der Zuverlässigkeit, verzichtet, um ein möglichst aussagekräftiges Ergebnis für die Kernanforderung zu erreichen.

4.2 Rahmenbedingungen

In diesem Abschnitt sollen die Rahmenbedingungen, die der prototypischen Implementierung zugrunde liegen, kurz erläutert werden.

Hardware Als Testdevice wurde ein Beagle Bone Black Einplatinencomputer verwendet, welcher von seinen Eigenschaften und seiner Performance ungefähr der Headunit eines Autos oder der zentralen Steuerkomponente einer Fertigungsanlage entspricht.

Der Beagle Bone Black wurde mit einem CAN-Controller ausgestattet, um auf einem realen Bussystem testen zu können.

Betriebsystem Als Betriebssytem wurde ein Standard-Linux auf Basis einer Raspian Distribution, welches wiederum auf einem üblichen Debian Linux mit der Kernelversion 3.13 basiert, verwendet.

SocketCAN Treiber Für die Anbindung der CAN-Hardware wurde der Open Source SocketCAN Treiber verwendet, der einen Berkley Socket für die CAN-Hardware zur Verfügung stellt und somit einfach zu nutzen ist. [Tea15]

OpenSSL Um die kryptografischen Algorithmen bereitzustellen, wurde die renommierte Entwickler-API „Libcrypto“, auf der OpenSSL basiert und die vom OpenSSL-Team entwickelt wird, verwendet. Diese stellt alle relevanten Algorithmen, Verfahren und Parameter bereit und lässt sich einfach anwenden. Es sind keine Einschränkungen hinsichtlich der Peformance aufgrund der Bibliothek zu befürchten. Siehe dazu auch [VMC02].

POSIX Da eine möglichst plattformunabhängige Lösung angestrebt wird, wurde die POSIX-Bibliothek eingesetzt. Hiermit ist der Prototyp auf Linux, Windows, QNX und vielen weiteren Betriebssystemen und Derivaten lauffähig. Darüber hinaus weist die POSIX-Bibliothek gute Perfornamceeigenschaften auf und ist im Embedded Umfeld weit verbreitet. [WT06]

Programmiersprache Im Bereich vom Embedded System ist es im Allgemeinen üblich C oder C++ als Programmiersprache zu wählen, da hiermit eine hardwarenahe Programierung möglich ist. Außerdem laufen C-Programme sehr performant und Speicherschonend, was vor allem auf Systemem mit geringen Hardwareressourcen von Vorteil ist.

Für die Entwicklung des Prototypen wurde aufgrund von Vorgaben C als Programmiersprache gewählt.

4.3 Konzept

Im Folgenden soll das grundlegende Konzept des Prototypen erläutert werden und die daraus resultierenden Erkenntnisse beschrieben werden.

4.3.1 Vorgehen

Bei der Entwicklung des Prototypen wurde ein vertikales Prototyping als Vorgehen ausgewählt, da viele Randbedingungen um die Kernanforderung unklar sind.

Dieses Prototyping stellt nur eine begrenzte Anzahl von Eigenschaften (Features) bereit, bietet jedoch volle Funktionalität der Eigenschaften. Dies wird aufgrund des kurzen Zeitraums mit einem evolutionären Modell kombiniert, sodass nach und nach neue Funktionalitäten hinzugefügt werden können, jedoch jederzeit ein valider Prototyp zur Verfügung steht.[Nie94, S.92ff]

Bei der Auswahl der zu realisierenden Komponenten wurde der Schwerpunkt auf die kryptografischen Funktionen gelegt, um einen möglichst viele, verschiedene Messungen an diesen durchzuführen zu können. Dies ergab sich auch aus dem Umstand, dass während der Bearbeitungszeit nur zwei Testdevices zur Verfügung standen.

Ergebnis des Vorgehens ist eine Eins-zu-eins-Kommunikation, die ein nahezu vollständiges Sicherheitsniveau bietet.

4.3.2 Architektur

Aufgrund der nicht vorhandenen Objektorientierung von C wurde eine sehr einfache Architektur gewählt, wie sie in 4.1 dargestellt ist[2].

Beschreibung der Klassen

Im Folgenden sollen die wichtigsten Aufgaben und Funktionen kurz beschrieben und deren Zusammenspiel erläutert werden.

Main, Client und Server Um die Entwicklung möglichst einfach zu gestalten und Coderedundanz zu vermeiden, wurden sowohl Client als auch Server in einer Implementierung realisiert. Die Unterscheidung, welche „Rolle“ gestartet werden soll, muss mit Hilfe von Terminalparametern getroffen werden. Abhängig davon werden weitere Parameter, wie das Passwort und die Kanäle, auf denen kommuniziert werden soll als Eingabe erwartet. Auf dieser Basis wird entweder der Client oder Servercode ausgeführt. Außerdem werden aus der Mainroutine zwei Threads gestartet.

Der „RXThread“ hat die Aufgabe auf den Bus zu hören. Der „TXThread“ ist dafür zuständig, eingegangene Nachrichten weiterzuverarbeiten und ggf. Antworten darauf zu versenden. Beide Threads blockieren, sofern es nichts für sie zu tun gibt, auf Semaphore. Der Mainthread sorgt dafür, dass das Programm weiterhin steuerbar bleibt, auch wenn die anderen Threads auf Semaphore oder Mutexes blockieren.

Die Klassen Client und Server verwalten zum einen die zur Laufzeit wichtigen Informationen. So verwaltet der Server eine Liste von Domainschlüsseln und ggf. Informationen zu den Clients, wohingegen der Client lediglich sein eigenes Passwort „weiß“.

Zum anderen verfügen Client und Server über die zentralen Methoden, um die sichere Kommunikation aufzubauen und zu verwalten. Diese sind z.B. „handleAuthentificationResponse“ im Server oder „handleEncryptedData“ im Client.

RXThread, Transmission und CANDriver Wie bereits angedeutet ist der „RXThread“ nur damit betraut, Nachrichten vom SocketCAN-Treiber entgegen zu nehmen und wei-

[2]Dieses Architekturbild folgt zwar dem entwickelten Prototypen, entspricht diesem jedoch nicht genau, da anschaulichere Namen verwendet wurden und nur die für das Verständnis wichtigen Funktionen dargestellt werden.

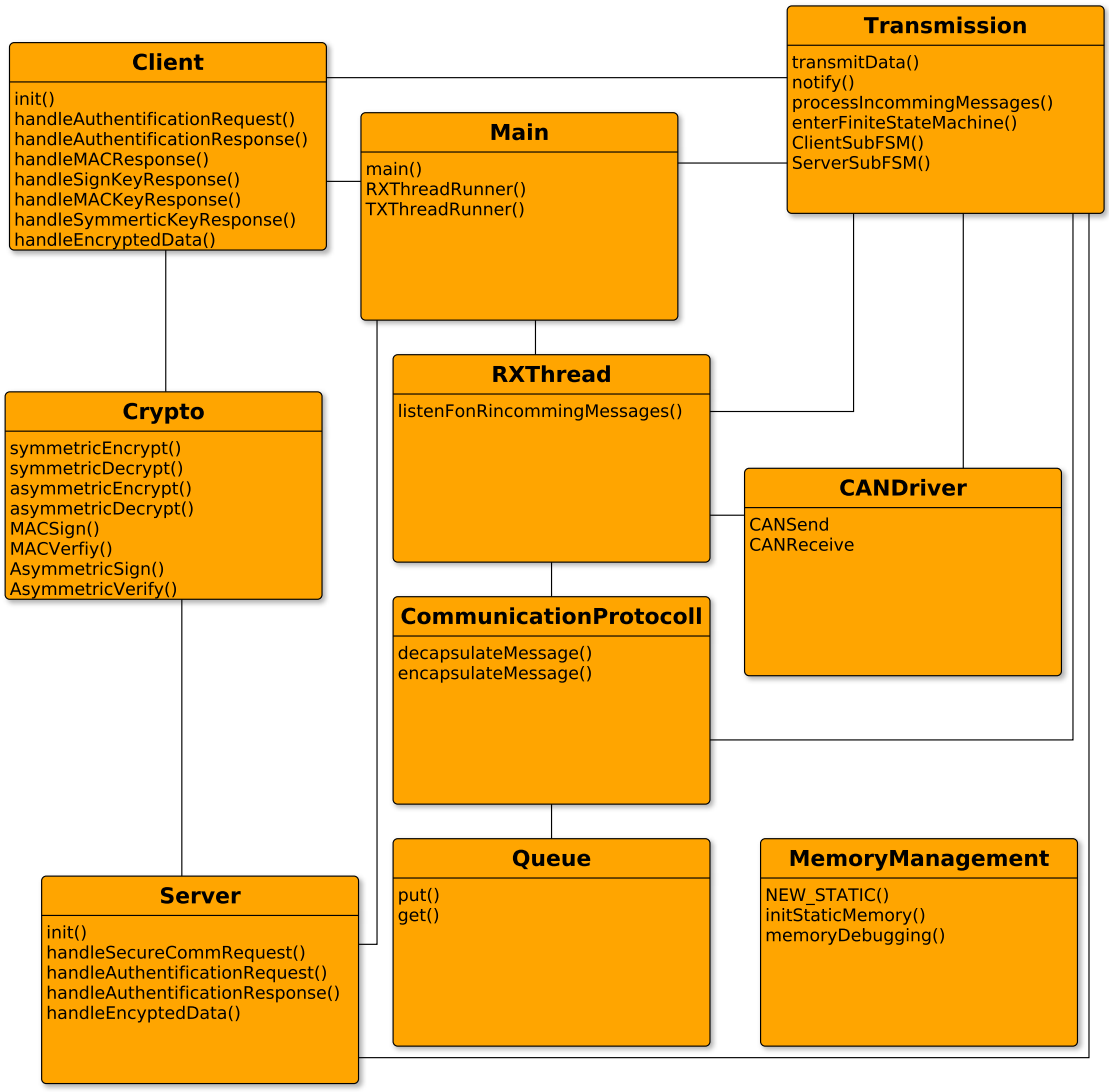


Abbildung 4.1: Architekturbild des Prototypen mit Zuordnung der wichtigsten Funktionen.

terzuverarbeiten. Hierbei werden die empfangenen CAN-Frames mit Hilfe des Kommunikationsprotokolls zu Nachrichten zusammengesetzt, die mehrere CAN-Frames lang sein können. Die fertige Nachricht wird anschließend an „notify“ weitergeleitet.

In der Klasse „Transmission“ [3] werden Nachrichten verteilt und mit der Funktion „transmitData“, die wiederum den CAN-Treiber aufruft, versendet. Die Bearbeitung eingehender Nachrichten über „notify“ wird mit Hilfe eines endlichen Automaten realisiert. Dieser leitet die empfangenen Daten aufgrund seiner Rolle (Client oder Server), seines aktuellen Zustands und des Identifiers im Nachrichtenkopf an eine Methode in der Klasse Client oder Server weiter, welche die Daten weiterverarbeitet.

CommunicationProtocol und Queue Die Klasse „CommunicationProtocol“ ist für die Verarbeitung einer Nachricht zu einer Abfolge von CAN-Frames bzw. von einem Strom von CAN-Frames zu einer Nachricht zuständig. Dies geschieht mit Hilfe der Queue-Klasse, die zwei verschiedene Implementierungen eines Ringpuffers zur Verfügung stellt.

Hierzu wird das in Unterabschnitt 2.4.3 beschriebene Präfixprotokoll mit Hilfe der zwei Methoden „encapsulate“ und „decapsulate“ implementiert. Dabei bekommt „encapsulate“ eine Nachricht und liefert eine Queue-Instanz zurück, die die Nachricht aufgeteilt in einzelne 8 Bit Sektionen und versehen mit dem Header des Präfixprotokolls enthält. „Decapsulate“ hingegen erhält eine Queue bestehend aus empfangenen Frames und rekonstruiert daraus eine Nachricht.

Dafür stellt die Klasse „Queue“ zwei verschiedene Instanzen der Zugriffsmethoden „put“ und „get“ zur Verfügung. Außerdem steht noch die Funktion „getFullInformation“ zur Verfügung, welche den kompletten Inhalt einer Queue zurück gibt. Dies führt in diesem Fall zu einer Performanceverbesserung der Zugriffe, da eine Nachricht sequenziell im Speicher liegt und daher mit dem Wissen der Anzahl der Elemente komplett zurückgegeben werden kann. Dies geht jedoch zu Lasten des Speicherverbrauchs.

Crypto Die Klasse „Crypto“ stellt die Schnittstelle zur OpenSSL-Bibliothek dar. Um möglichst klare Methodenaufrufe für die kryptografischen Algorithmen und einfache Ausgabeformate zum Austausch der Schlüssel zu bekommen, wurden die verwendeten OpenSSL-Aufrufe hier noch mal sauber gekapselt. Dies erhöht darüber hinaus die Lesbarkeit und Wiederverwendbarkeit des Codes erheblich. An dieser Stelle werden die asymmetrischen uns symmetrischen Verschlüsselungs- und Authentifizierungsmethoden, Schlüsselexport- und Hashfunktionen bereitgestellt. Diese sollen hier nicht im einzelnen beschrieben werden.

MemoryManagement Wie bereits in Unterabschnitt 2.3.2 beschrieben sollte eine statische Speicherverwaltung verwendet werden. Diese wurde auch in der prototypischen

[3] Auch wenn C natürlich keine Klassen im Sinne der Objektorientierung bereitstellt, erklärt der Begriff die Zusammenhänge hier aus meiner Sicht am anschaulichsten.

Implementierung zum Teil umgesetzt und befindet sich in der Klasse „MemoryManagement“. Von der Implementierung eines “DynamicPools„ wurde jedoch abgesehen, da das Testsystem mit 1 GB Arbeitsspeicher gut ausgestattet ist und die Anzahl der Schlüssel im Prototypen aufgrund der 1:1-Kommunikation überschaubar bleibt.

global.h In dem Header „global.h“ befinden sich wichtige Strukturen, sowie Precompiler Directives und globale Variablen. Dieser Header wird von jeder anderen Klasse implementiert, und stellt damit eine einheitliche Schnittstelle zu den globalen Datenobjekten dar. Hier können außerdem die kryptografischen Verfahren, Schlüssellängen, Queuegrößen und die Protokollierung umgestellt werden.

4.3.3 Nachrichtenverlauf

Das Sequenzdiagramm in Anhang B zeigt eine Übersicht über den vollständigen Nachrichtenverlauf im Protoytpen. Dieses unterteilt sich in drei Komponenten: Eine zum Empfangen einer Nachricht, eine zur Verarbeitung und schließlich eine Komponente, die für das Senden einer Nachricht verantwortlich ist.

Empfangen

Abbildung 4.2 zeigt die Empfangsroutine der Implementierung. Diese „hört“ ständig auf den SocketCAN-Treiber, was bei Inaktivität durch ein Semaphor blockiert wird. Wird eine Nachricht empfangen, so wird diese aktiv und leitet die empfangenen Daten an den „RXThread“ weiter. Dieser entkapselt den Frame, um daraus eine Nachricht zu bauen, wofür außerdem die Queue benötigt wird. Die „decapsulation“ Routine der Klasse „CommunicationProtocol“ gibt entweder *NULL* oder eine Queue-Instanz zurück. Wird eine Queue-Instanz zurückgeliefert, so wird diese mit Hilfe von „getFullInformation“, der Queue-Klasse in eine Struktur der Form „GlobalMessageFormat“ gewandelt. Die Funktion „notify“ legt nun die Nachricht in eine Queue, wodurch ein Semaphor inkrementiert wird und dem „TXThread“ signalisiert, dass er die Nachricht weiterverarbeiten kann.

Weiterverarbeitung

In Abbildung 4.3 ist die Verarbeitung des Frames dargestellt. Nachdem eine Nachricht in der „GlobalMessageQueue“ abgelegt wurde, holt sich der „TXThread“ diese Nachricht aus der Queue. Dies übernimmt die „processIncommingMessages“-Funktion, die im Diagramm oben links ohne Beschriftung dargestellt ist. Diese ruft einen endlichen Automaten (Finite State Machine (FSM)) auf, welcher zunächst aufgrund der aktuellen Rolle zwischen Server und Client unterscheidet und in entsprechende SubFSMs verzweigt. Dies wurde im Sequenzdiagramm aus Gründen der Übersichtlichkeit abstrahiert. Aus dem jeweiligen SubFSM wird nun basierend auf der aktuellen Rolle, dem Zustand

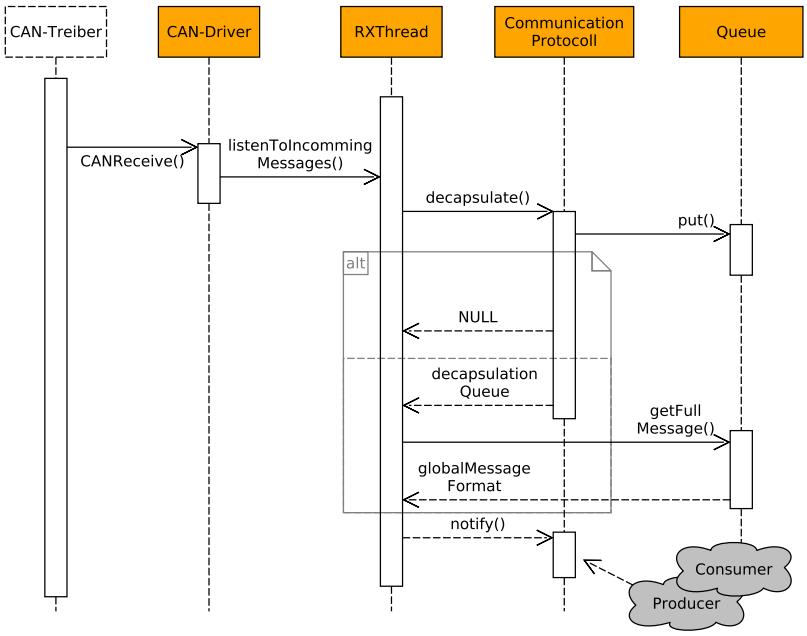


Abbildung 4.2: Teil des Sequenzdiagramms, welcher das Empfangen eines Frames beschreibt.

der Kommunikation [4] und der Eingabenachricht eine Funktion aus der „Crypto'-Klasse aufgerufen. Dies wird im Sequenzdiagramm ebenfalls verallgemeinert dargestellt. Das Ergebnis dieser wird nun je nach Inhalt der Nachricht intern weiterverarbeitet oder versendet.

Senden

Das Senden einer Nachricht ist in 4.4 dargestellt. Hierbei wird die Nachricht aus dem FSM mit Hilfe der „encapsulate“ und „put“ Methode in CAN-Frames zerlegt. Nach diesem Vorgang wird die Queue mit „get“ sequenziell geleert und die einzelenen CAN-Frames mit „CANSend“ an den Treiber übergeben, welche sie mit „write“ auf den Bus schreibt. Dies wird sowohl in der Server- als auch in der Clientinstanz ausgeführt.

Multithreading

Aktuell besteht der Prototyp aus insgesamt drei Threads, die in der Initphase gestartet werden. Dabei übernimmt der „RXThread“ das Empfangen eines Frames und der „TXThread“ das Weiterverarbeiten und Senden einer Nachricht. Der Mainthread hat hierbei eine eher untergeordnete Rolle und gewährleistet lediglich die Steuerbarkeit des Programms über Benutzereingaben.

Während der Implementierung des Prototypen hat sich herausgestellt, dass eine Unterteilung des „TXThreads“ in zwei weitere Threads zu einem strukturierteren Nachrich-

[4] In der derzeitigen Version der Prototypen wird der aktuelle Zustand nur vereinzelt genutzt.

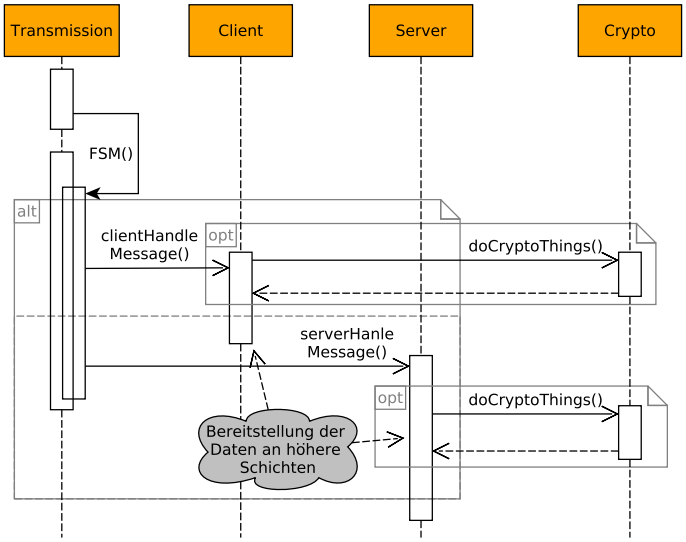


Abbildung 4.3: Teil des Sequenzdiagramms, welcher das Weiterverarbeiten eines Frames beschreibt.

tenverlauf geführt hätte. Da kryptografische Berechnungen und das Senden von Frames parallelisiert worden wären hätte dies u.U. zu einer höheren Sytemperformance geführt. Dies ließ sich jedoch in der gegebenen Zeit nicht mehr umsetzen.

Des Weiteren wird in der Version mit ECs ein weiterer Hilfsthread benötigt. Dies wird in 4.3.5 nochmals näher beschrieben.

Queues

Es werden zwei verschiedene Queues verwendet, um einen geregelten Nachrichtenverlauf zu gewährleisten. Dabei ist eine, wie bereits in der Beschreibung der Klassen erwähnt, für das Zusammensetzen der Nachrichten aus Frames (bzw. umgekehrt) zuständig. Eine weitere Queue verwaltet die Nachrichten innerhalb des Progamms. Hier werden die empfangenen und verarbeiteten Nachrichten vom „RXThread“ in eine Queue gelegt, welche vom „TXThread“ ausgelesen wird.

CAN-IDs

Da der Prototyp lediglich ein 1:1-Kommunikation bereitstellt, wurde nur mit jeweils einer CAN-ID kommuniziert und auf weitere Auswertungen der Daten auf Basis der CAN-IDs im Prototyp verzichtet.

4.3.4 Speicher

Um den Vorgaben im Bereich des embdedded Engineering gerecht zu werden, wurde eine vereinfachte Version der in 2.3.2 vorgestellten Speicherverwaltung realisiert. So wird

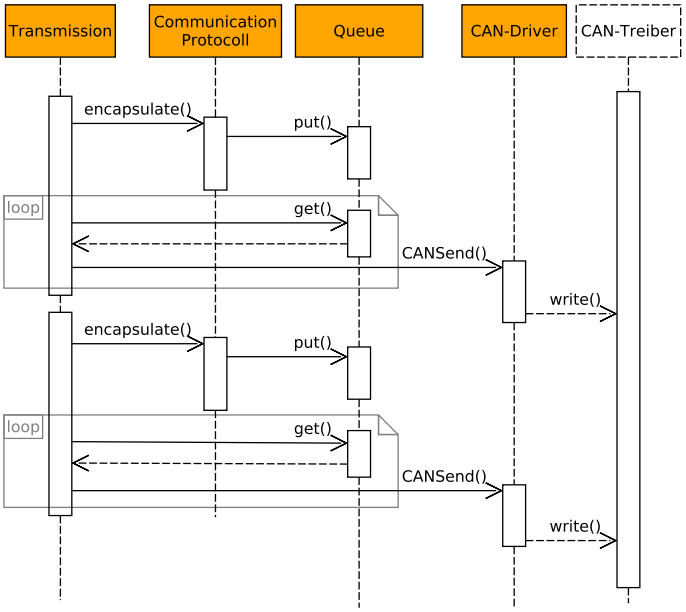


Abbildung 4.4: Teil des Sequenzdiagramms, welcher das Senden eines Frames beschreibt.

während der Initialisierung sämtlicher Speicher allokiert und an die Methoden übergeben, sodass fast ganz auf eine Allokierung nach der Initphase verzichtet werden konnte. Aufgrund des dynamischen Entwicklungsprozesses und des engen Zeitrahmens konnten einige kleinere Variablen nicht mehr in die Initphase migriert werden, daher „leckt“ das Programm leicht hinsichtlich des Speichers.

4.3.5 Realisierung des ECDH im Prototypen

Eines der Kernstücke der prototypischen Implementierung stellt die Realisierung der PFS mit Hilfe des Elliptic Curves Diffie Hellman (ECDH)-Verfahrens am Prototypen dar.

Wie in Abschnitt 2.1.3 beschrieben ist, ist das Austauschen des gemeinsamen Geheimnisses zur Laufzeit des Algorithmus erforderlich, da beide Teilnehmer auf Basis eines Teilgeheimnisses des gemeinsame Geheimnis errechnen.

Die Implementierung durch einen endlichen Automat ist in diesem Fall hinderlich für den Austausch des gemeinsamen Geheimnisses. Bekommt der Server eine Anfrage über eine sichere Kommunikation, durchläuft diese den Automaten und ruft die Routine des ECDH-Verfahrens auf, welche dem Client den Serverteil des Geheimnisses sendet. Der Client berechnet auf dieser Basis das gemeinsame Geheimnis und sendet den Clientteil des Geheimnisses an den Server. Diese Nachricht wartet nun endlos auf den Eintritt in den endlichen Automat, welcher noch durch das Warten auf eben diese Nachricht blockiert ist. Wir haben also anschaulich gesprochen einen „ECDH-Deadlock“ gebaut.

In dem sequenzartigen Diagramm 4.5 wird eine einfache Möglichkeit gezeigt, wie dieses Problem gelöst werden kann. Hierfür wird der Aufruf des ECDH-Verfahrens in

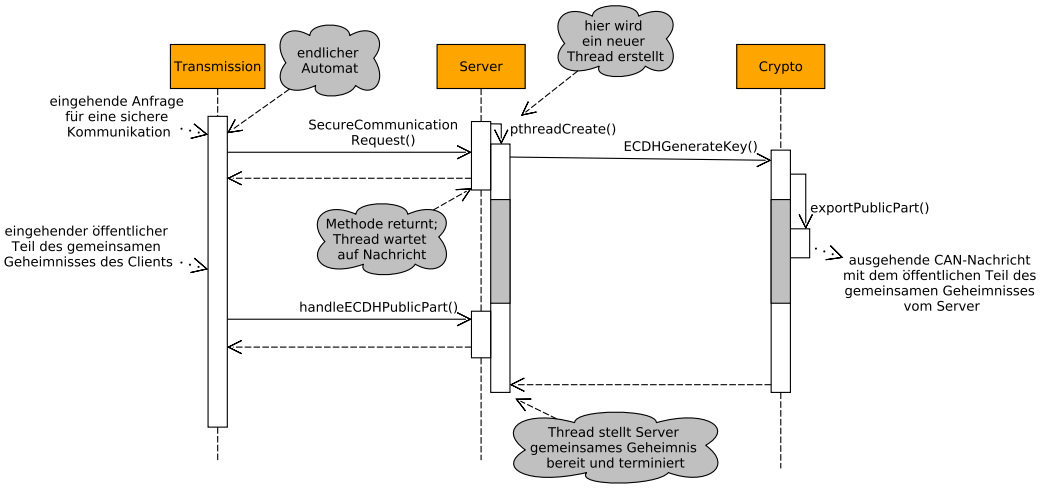


Abbildung 4.5: Veranschaulichung des Ablaufes eines ECDH-Verfahrens mit Hilfe eines Threads

einen weiteren Thread ausgelagert, welcher das Servergeheimnis versendet und auf die Antwort vom Client wartet. Somit kann der endliche Automat weitere Nachrichten, wie das Clientgeheimnis, verarbeiten. Die Übergabe des Geheimnisses wird dabei durch eine globale Variable realisiert. Signalisiert wird der Eingang durch ein binäres Semaphor, auf welches der ECDH-Thread wartet und was bei Nachrichteneingang freigegeben wird.

Dieses Vorgehen des Erstellens eines neuen Threads zur Laufzeit widerspricht jedoch den in 2.3.3 beschrieben Verfahren für Embedded Systems. Für eine Implementierung empfiehlt sich daher einen Workerthread einzuplanen, der vom Start weg läuft und u.a. diese Funktion übernimmt. Zudem könnte dieser auch die Vorberechnung von anderen Algorithmen, wie in 2.3.3 beschrieben, übernehmen.

4.3.6 Limitierungen der Implementierung

Aufgrund der zur Verfügung stehenden Zeit, konnten nicht alle Anforderungen realisiert werden.

Es ist mit dem Prototypen möglich, verschiedene Verschlüsselungs- und Signaturverfahren hinsichtlich Laufzeit, Speicher und benötigter CAN-Frames zu analysieren. Konkret ist ein Verfahren basierend auf elliptischen Kurven und ein RSA-Verfahren realisiert, wobei sich dieses mit und ohne Vorberechnung der Schlüssel einsetzen lässt. Als sichere Kommunikation ist der AES-Standard realisiert worden. Somit kann eine Aussage über die Effizienz einer verschlüsselten und authentischen Kommunikation auf Basis der jeweils gängigen Algorithmen getroffen werden.

Der Prototyp ist durch die folgenden Punkte limitiert.

- Die Challenge Response Authentifikation fehlt im Moment, dafür wird das Passwort verschlüsselt und signiert übertragen. Hinsichtlicht der Laufzeit fehlt lediglich

die Verschlüsselung und Übertragung der Nonce.

- Eine Routine zur Durchführung eines Schlüsselupdates wurde nicht implementiert.
- Es wird bei der Inittiierung kein MAC-IV übertragen. Dieser ist hartkodiert. Dies betrifft alle Initiierungverfahren gleichermaßen.
- Die Übertragung der öffentlichen RSA-Schlüssel verläuft nicht ganz optimal. Hierbei wird ein Frame zu viel übertragen.

Die genannten Punkte sollten jedoch zu keinen nennenswerten Veränderungen der Messergebnisse führen, bzw. deren Folgen können aus den Ergebnissen errechnet werden.

Die zentrale Anforderung kann jedoch lediglich in Bezug auf eine 1:1-Kommunikation evaluiert werden. So stellt der Prototyp aktuell lediglich eine Kommunikation zwischen einem Server und einem Client bereit. Hierbei simuliert der Server, anders als in Kapitel 3 gezeigt, nach dem Authentifizierungsprozess einen zweiten Client. Die meisten Teile der Logik, für eine M:N-Kommunikation, sind bereit implementiert und können direkt für weitere Tests herangezogen werden.

Aufgrund der 1:1-Kommunikation können nicht alle weiteren Anforderungen analysiert werden. So ist ein Monitoring der Netzwerkkommunikation, zum Beispiel bei einem Schlüsselupdate, nur bedingt möglich, da hierfür mehrere Teilnehmer kommunizieren müssten.

Hinsichtlich Plattformunabhägigkeit, Austauschbarkeit der Kommunikationsprotokolle und der Lauffähigkeit auf einem Embedded System konnten die Anforderungen weitestgehend erfüllt werden. Dies ist jedoch stark vom jeweiligen System, Compiler und anderen Faktoren abhängig und muss im Einzelfall analysiert werden.

Kapitel 5

Messungen

In diesem Kapitel sollen die durchgeführten Messungen beschrieben werden und daraus Erkenntnisse über die benötigte Zeit und die erforderlichen CAN-Frames gewonnen werden. Dies Messungen unterteilen sich in die Initiierung und Nachrichtenübertragung des Prototypen. Um einen Anhaltspunkt über die Aussagekraft zu geben wird zunächst das eingesetzte Messverfahren beschrieben.

5.1 Beschreibung der eingesetzten Messmethode

Im Folgenden sollen die gewählte Technik und die Methode erläutert werden, mit der die Messungen erstellt wurden. Außerdem soll eine Abschätzung der Aussagekraft der Messungen auf Basis der technischen Gegebenheiten getroffen werden.

5.1.1 Timer

Für die Messungen der Laufzeit von Algorithmen und Abläufen im Programm werden POSIX High Resoluion Timer verwendet. Diese lösen mit einer Präzision von Nanosekunden auf und sind weitestgehend auf allen POSIX kompatiblen Systemen einsetzbar. [Ker13] Um möglichst unverfälschte Ergebnisse zu erhalten wird der Clock_Monotonic-Timer verwendet. Dieser ist frei von Zeitsprüngen basierend auf Justierungen der Systemzeit jedoch, wie alle Timer im System, betroffen von Zeitverzögerungen durch Interrupt Service Routinen. [Wie12, S.113 / S.124f]

5.1.2 Systematischer Fehler und Abschätzung der Präzision

Da es sich um in Software implementierte Timer handelt, benötigen diese natürlich eine gewisse Zeit, zum Starten und Beenden. Um diesen Offset zu bestimmen, wurde bei jeder Messung ein Referenztimer gesetzt, der zwischen Start und Ende keinen Code enthält.

Der Mittelwert aller Referenztimer beträgt $\approx 7,25\,\mu s$. Dieses Offset kann als mittlerer systematischer Fehler betrachtet werden und tritt als Verzögerung bei jeder durchge-

führten Messung auf. Die angegebenen Messwerte wurden nicht um den systematischen Fehler bereinigt.

Die Präzision der Messungen, wird durch die Standardabweichung aller Referenztimer bestimmt und beträgt $\approx 0,42\,\mu s$.

Es ist zu beobachten, dass tiefe Schachtelungen (> drei) von Timern zu einer reproduzierbaren Verzögerung des äußeren Timers führt. Um dies zu umgehen, wurde auf eine Schachtelung von einer Tiefe größer als zwei verzichtet.

5.1.3 Auswahl der gemessenen Aktivitäten

Im folgenden Abschnitt sollen die für die Messungen gewählten Aktivitäten beschrieben werden und ggf. auf Besonderheiten im Messverfahren hingewiesen werden. Dazu zeigt die Grafik in Abbildung 5.1 einen Authentifizierungsvorgang mit dem RSA-Verfahren, in 5.2 die Authentifizierung mit ECDH und ECDSA und schließlich in 5.3 die verschlüsselte Kommunikation mit dem AES. Die orangen Kästchen zeigen eine einzelne Aktivität, wie z.B. das Verschlüsseln einer Nachricht an. Dabei wurde auf die Darstellung aller vermutlich zeitintensiven Aktivitäten geachtet.

Die gelben Kästen stellen Gruppen von Aktivitäten dar, die ein konkretes Ergebnis liefern und dessen Zeitverhalten von Interesse ist. Diese unterteilen sich während der Initiierungsphase noch in Obergruppen, welche die Dauer der kompletten Initiierung umfassen und Untergruppen, die Teile der Initiierung messen. Die Pfeile zwischen den Gruppen signalisieren den Austausch von CAN-Nachrichten.

Die Messungen werden so angesetzt, das für alle Gruppen von Aktivitäten eine Aussage über die Laufzeit gemacht werden kann. Dabei werden die Obergruppen jeweils durch Graphen dargestellt, die das Zeitverhalten von 100 Messungen widerspiegeln. Die Messwerte der Untergruppen werden in Form von Kreisdiagrammen miteinander verglichen. Hier wurde der Mittelwert über jeweils 15 Messungen gebildet, um ein möglichst aussagekräftiges Ergebnis zu erhalten. Während der Messungen hat es sich als hilfreich herausgestellt, nicht nur die Laufzeit der Untergrupperen zu messen, sondern auch die Wartezeit. Diese kann anschaulich als der Abstand zwischen den Untergruppen verstanden werden und ist in Abbildung 5.1 beispielhaft dargestellt.

Die Aktivitäten werden teilweise parallel ausgeführt, weshalb die Summe aller Untergruppen größer ist, als die Laufzeit zur Initiierung der sicheren Kommunikation.

Initiierung der sicheren Kommunikation

Für die Initiierung der sicheren Kommunikation wurden zwei verschiedene Verfahren untersucht. Zum einen das weit verbreitete RSA-Verfahren, welches in Abschnitt 2.1.2 beschrieben wurde und zum anderen das auf elliptischen Kurven basierende Verfahren, erklärt in Abschnitt 2.1.2.

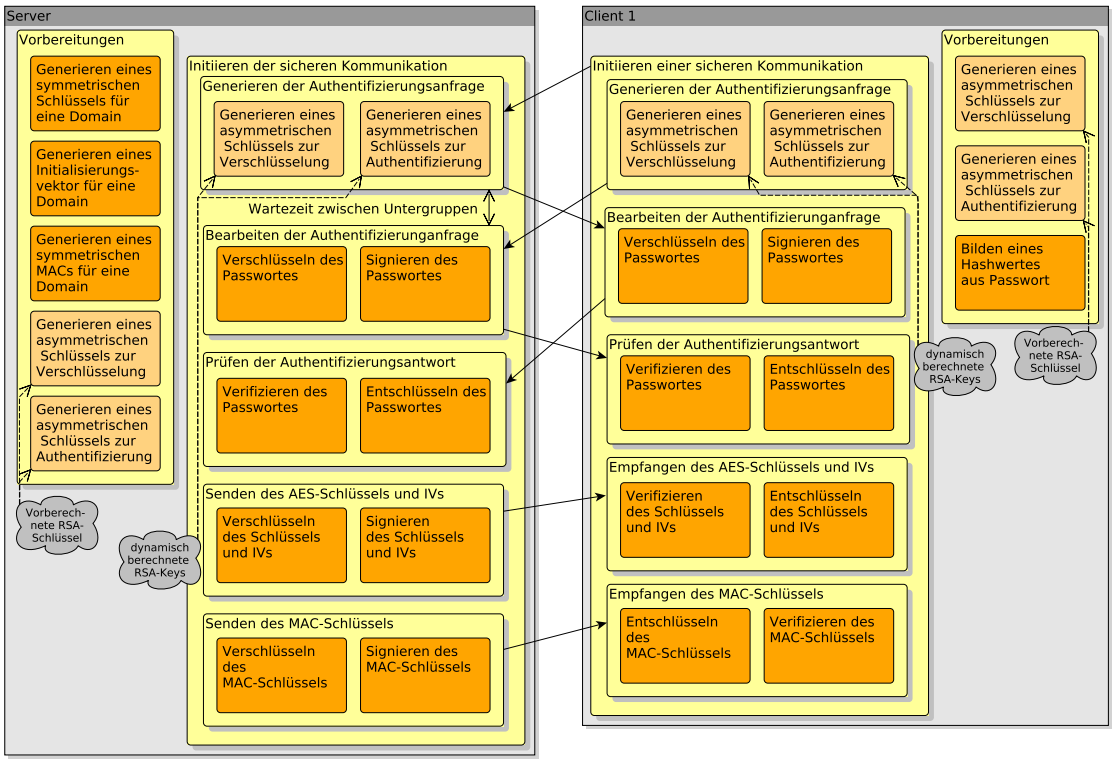


Abbildung 5.1: Übersicht über die Aktivitäten, die für die Initiierung der sicheren Kommunikation mit dem RSA ausgeführt werden müssen.

Die mit „Vorbereitung“ beschrifteten Kästen müssen sowohl vom Client als auch vom Server ausgeführt worden sein, bevor eine Anfrage gesendet wird. Da der Prototyp lediglich eine 1:1-Kommunikation unterstützt, ist das in Unterabschnitt 2.3.8 beschriebene Zeitverhalten für die Vorberechnung von Schlüsseln hier irrelevant. Wie in Abschnitt 3.5 beschrieben ist es erforderlich, die symmetrischen Schlüssel bei jedem Eintritt eines Clients in eine Domain neu zu vergeben, ist es ebenfalls erforderlich, auch diese Schlüssel jedes mal neu zu berechnen. So muss also jeweils nach der Aktivität „Initiieren der sicheren Kommunikation“ die Aktivität „Vorbereitung“ ausgeführt werden. Da dies jedoch keinen direkten Zusammenhang hat, sind die Aktivitäten voneinander getrennt.

RSA Zur Initiierung mit dem RSA werden zwei Schlüsselpaare mit einer Länge von 1024 Bit[1] generiert und getauscht. Dies ist sowohl während der Initiierung als auch im Vorfeld möglich. Jedoch bringen beide Möglichkeiten Vor- und Nachteile mit sich, welche in 5.2.2 beschrieben sind. Die Algorithmen sind in Abbildung 5.1 etwas heller dargestellt und mit einem Hinweis (in Wolkenform) beschriftet.

[1]Wie bereits beschrieben, gilt das RSA-Verfahren mit 1024 Bit nicht mehr als sicher. Aufgrund der hohen gemessenen Laufzeiten bei der Schlüsselgenerierung wurde jedoch dieses Schlüssellänge gewählt, um eine Vergleichbarkeit zu gewährleisten.

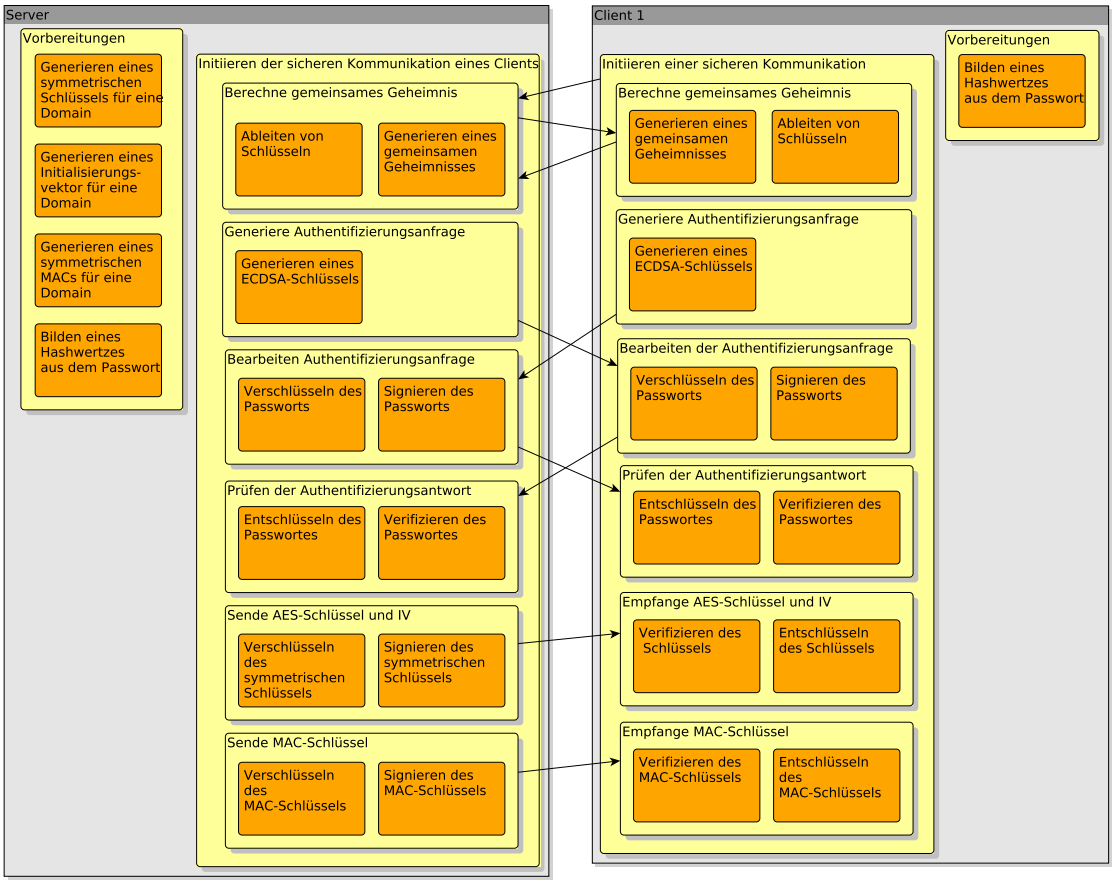


Abbildung 5.2: Übersicht über die Aktivitäten, die für die Initiierung der sicheren Kommunikation mit dem EC-Verfahren ausgeführt werden müssen.

Elliptic Curves Bei der Initiierung mit EC wird sowohl die asymmetrische Verschlüsselung als auch die Signatur durch Verfahren mit elliptischen Kurven realisiert. Wie in 2.1.2 beschrieben, wird dadurch eine kürzere Schlüssellänge benötigt, weshalb eine schnellere Authentifizierung erwartet wird. Zudem fällt eine Primzahlgenerierung gegenüber dem RSA weg.

Die Initiierung unterscheidet sich im Wesentlichen in den ersten beiden Schritten vom RSA-Kryptosystem. Hier wird mit dem ECDH das gemeinsame Geheimnis ermittelt und daraus ein Schlüssel abgeleitet (Abbildung 5.2). Dabei ist zu erwähnen, dass mit dem gemeinsamen Geheimnis keine asymmetrische Verschlüsselung mehr durchgeführt wird. Aus dem gemeinsamen Geheimnis wird stattdessen ein symmetrischer Schlüssel abgeleitet, der in den folgenden Schritten in einem symmetrisches Verschlüsselungsverfahren eingesetzt werden kann (vgl. 2.1.2). Der ECDSA funktioniert jedoch nach dem gleichen Prinzip wie der RSA.

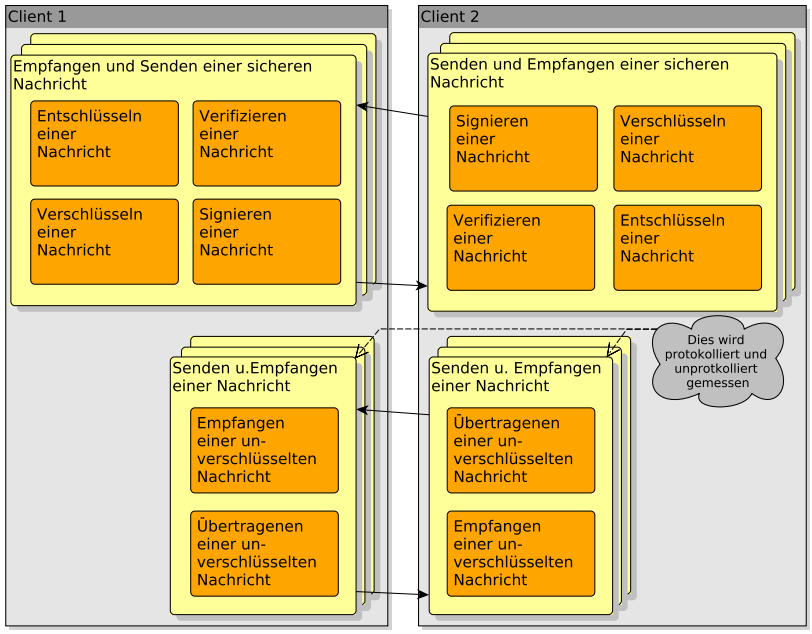


Abbildung 5.3: Übersicht über die Aktivitäten, die für eine sicheren Nachrichten mit dem AES ausgeführt werden müssen im Vergleich zu einer unverschlüsselten Übertragung

Nachrichtenübertragung

Für die Nachrichtenübertragung wurde der AES gegenüber der unverschlüsselten Kommunikation betrachtet.

AES Die sichere Kommunikation wird mit dem AES realisiert und wird sowohl für die Verschlüsselung als auch zur Signatur der Nachricht mit Hilfe des CMAC-Verfahrens verwendet. Ein sendender Client verschlüsselt und signiert eine Nachricht mit den jeweiligen Domainschlüsseln, die auch dem Empfänger bekannt sind, welcher diese auf gleichem Wege wieder verifiziert und entschlüsselt. Dieses Verfahren wird der unverschlüsselten Übertragung gegenübergestellt, um den zusätzlichen Aufwand für die Verschlüsselung, gegenüber der Klartextübertragung zu ermitteln. Die unverschlüsselte Kommunikation wird außerdem in eine protokolliere und eine unprotokollierte Nachrichtenübertragung unterteilt.

5.2 Vorstellung der Messergebnisse

In diesem Abschnitt sollen die durchgeführten Messungen vorgestellt und deren Ergebnisse interpretiert werden. Darüber hinaus sollen Schlüsse aus den Messungen auf die gewählten Verfahren gezogen werden. In Abbildung 5.4 wird eine Übersicht der durchgeführten Messungen in Zusammenhang mit den gemessenen Verfahren dargestellt. Wie darin zu sehen ist, unterteilen sich die Messergebnisse in den Aufwand, zur Initiierung

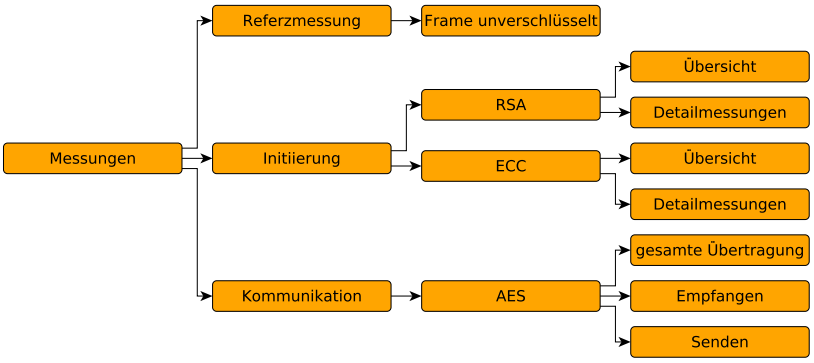


Abbildung 5.4: Übersicht über die durchgeführten Messungen

der sicheren Kommunikation und den Aufwand zur sicheren Nachrichtenübertragung.

Im Fall der Initiierung unterteilen sich die gemessenen Aktivitäten wiederum in die beiden Verfahren RSA und ECC. Hier wird jeweils sowohl eine grundsätzliche Übersicht über die Laufzeit des Verfahrens gegeben als auch eine detaillierte Aussage über die Laufzeit einzelner Komponenten getroffen.

Für die sichere Kommunikation wird lediglich der AES gemessen. Hier werden Messungen des Sendens, Empfangens und der Übertragung einer Nachricht inkl. Antwort vorgestellt. Diese werden zusätzlich in Gegenüberstellung zu einer unverschlüsselten Kommunikation betrachtet und mit dieser verglichen.

Der *Aufwand* beschreibt hierbei in erster Linie die erforderliche Zeit, jedoch wird auch auf die benötigten CAN-Frames eingegangen.

5.2.1 Grundlegende Messungen

Um die folgenden Messungen besser einordnen zu können, soll zunächst die grundsätzliche Performance des Testsystems vorgestellt werden. Diese wurde durch das Senden von zuvor zufällig generierten Datenframes ermittelt. Dafür wird ein Frame 100 mal vom Client gesendet, welcher vom Server empfangen und direkt zurück gesendet wird. Dabei wird die Wartezeit nach dem Senden bis zum Empfang der Antwort im Client und die Bearbeitungszeit zwischen dem Empfangen und dem Senden im Server gemessen. Um die Übertragungszeit eines Frames zu ermitteln wird nun die Bearbeitungszeit im Server von der Wartezeit im Client abgezogen und das Ergebnis durch zwei geteilt.

In Abbildung 5.5 ist die Laufzeit eines Frames dargestellt. Als Zeitraum wurde das Senden der ersten 10 Frames gewählt. Hierbei ist zu sehen, dass die ersten Frames eine deutlich längere Übertragungszeit benötigen, als der Durchschnitt. Die Ursache dafür liegt vermutlich im „SocketCAN“-Treiber und konnte nicht ermittelt werden. Weitere Tests zeigen, dass dieses Verhalten nur nach dem Starten des Programms auftritt.

Im Durchschnitt benötigt ein Frame eine Übertragungszeit von rund 289 μs, wobei die Maximalzeit beim ersten Frame des Clients bei 479 μs liegt.

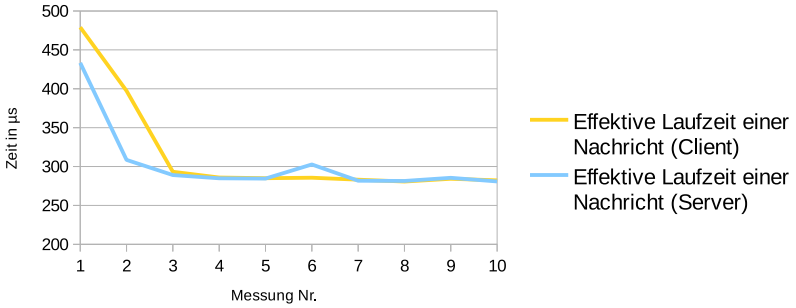


Abbildung 5.5

Rechnerisch ergibt sich aus einer Übertragungszeit von 289 μs eine Übertragung von 3460 Frames pro Sekunde und dadurch eine Übertragungsgeschwindigkeit von 27,031 KB/s.

5.2.2 Gegenüberstellung der Initiierungsverfahren

In diesem Abschnitt soll die Laufzeit der Initiierungsverfahren EC und RSA gegenübergestellt und interpretiert werden. Hierbei wird außerdem zwischen dem RSA mit und ohne vorberechneten Schlüsseln unterschieden. Die jeweiligen Verfahren werden zudem durch Detailmessungen in Form von Kreisdiagrammen genauer analysiert.

Elliptic Curves Cryptography

Der zeitliche Verlauf von 100 Durchläufen des Initiierungsverfahrens mit Hilfe der Elliptic Curves Cryptography (ECC) ist in Abbildung 5.6 dargestellt. ECC ist in der Lage, in $\approx 75,05\,ms$ eine sichere Kommunikation aufzubauen und verfügt über eine sehr geringe Standardabweichung von $\approx 0,99\,ms$ (1,32 %). Diese Werte sind jeweils im Client gemessen worden, spiegeln jedoch auch die Messungen im Server wider.

Dabei ist auffällig, dass die Initiierungszeit im Client jeweils um ca. 80 μs größer ist als im Server. Dies erklärt sich dadurch, dass der Client die Kommunikation beginnt und zum Schluss Daten erhält, die er noch entschlüsseln muss, während der Server den Prozess bereits abgeschlossen hat.

Abbildung 5.7 stellt den zeitlichen Verlauf einer Initiierung im Detail dar. Hierbei ist direkt ersichtlich, dass die Erstellung des gemeinsamen Geheimnisses im Server wesentlich länger dauert als im Client (Server: 30,33 % / 25,44ms Client: 18,18 % / 11,95 ms). Dies ist darauf zurückzuführen, dass der Server darauf warten muss, dass der Client sein Teilgeheimnis erstellt. Im Client stellt sich die Wartezeit in „Warte auf Teilgeheimnis“ (15,09 % / 9,93 ms) und zum Teil in „Warte auf Authentifizierungsanfrage“ (9,71 % / 6,39 ms) dar. Insgesamt nimmt das ECDH-Verfahren ungefähr ein Drittel der gesamten Zeit ein.

Die relativ lange Wartezeit auf eine Authentifizierungsanfrage (9,71% /6,39 ms) , das Generieren einer Authentifizierungsanfrage (17,46% / 11,48ms) im Client und die

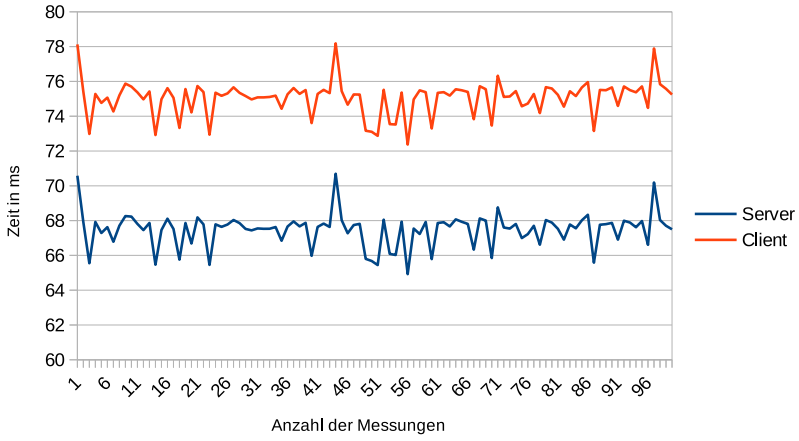


Abbildung 5.6: Darstellung der benötigten Zeit zur Initiierung der sicheren Kommunikation mit Hilfe der Elliptic Curves Cryptography. Der Graph umfasst 100 Initiierungsvorgänge und schafft einen Eindruck über die Abweichung vom Mittelwert.

Aktivitäten „Bearbeite Authentifizierungsanfrage“ (10,47 % / 8,78 ms) bzw. „Warte auf Authentifizierungsantwort“ (11,07 % / 9,28 ms) im Server nehmen ebenfalls relativ viel Zeit ein. Dies begründet sich durch die Erstellung der ECDSA-Schlüssel, worauf der jeweils andere warten muss. An dieser Stelle könnte die Performance optimiert werden, indem die Schlüssel vorberechnet werden.

Erwähnenswert ist noch, dass das Warten auf eine Authentifizierungsantwort etwas länger dauert (Client: 10,32 %, 6,79 ms / Server: 11,07 %, 9,29 ms) , als das Bearbeiten (inkl. Senden) dieser (Client: 7,13 %, 4,69 ms / Server: 10,47 %, 8,78 ms) , da hier sowohl auf die Verschlüsselung des Partners als auch auf die Übertragung der Nachricht gewartet werden muss.

Das Prüfen der Authentifizierungsantwort verläuft sowohl im Client als auch im Server sehr schnell (Client: 6,39 %, 4,21 ms / Server: 5,08 %, 4,27 ms), da hier u.a. das symmetrische Verfahren zum Einsatz kommt und keine CAN-Frames übertragen werden.

Schließlich ist auffällig, dass die Wartezeit auf den MAC-Schlüssel im Gegensatz zum AES-Schlüssel sehr kurz ausfällt(AES: 8,47 %, 5,57 ms / MAC: 1,23 %, 0,81 ms). Dies liegt daran, dass der Server sowohl AES als auch MAC-Schlüssel direkt nacheinander verschlüsselt und versendet. Daher liegt der MAC-Schlüssel bereits in der Queue des Clients, der ihn nur noch weiterverarbeiten muss.

Die Sende- und Bearbeitungszeit des AES-Schlüssels dauert länger als des MAC-Schlüssels, weil der MAC-IV hartkodiert ist. (Bearbeitungszeit im Client: AES: 14,84 %, 9,76 ms / MAC: 6,27 %, 4,13 ms)

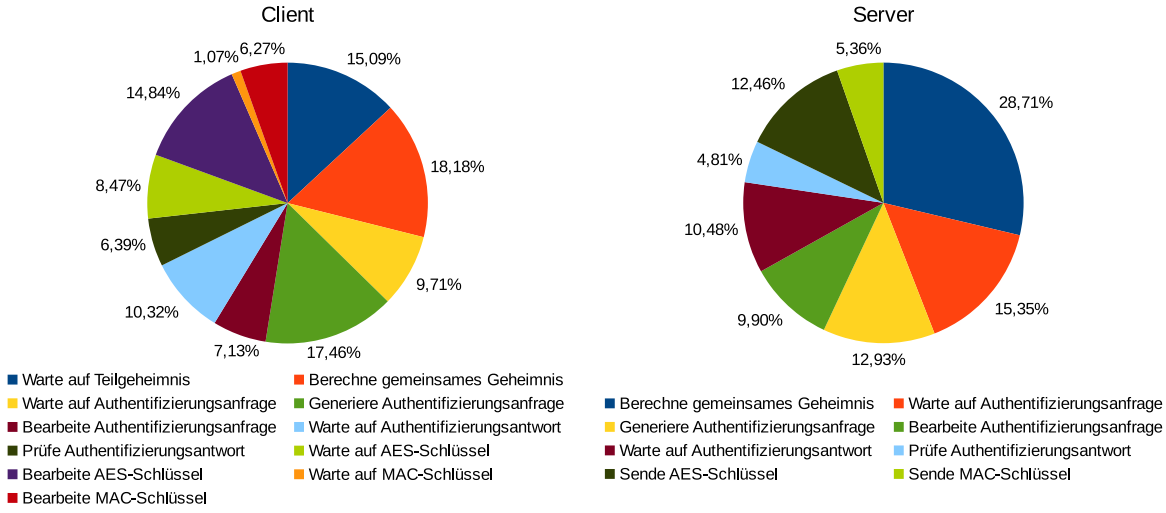


Abbildung 5.7: Detaildarstellung des Zeitverhaltens bei der Initiierung über das EC-Verfahrens im Client (links) und im Server (rechts)

RSA1024

Beim RSA-Verfahren zeigt sich gegenüber dem EC ein komplett verändertes Bild. Zunächst ist an der Skalierung der y-Achse direkt ersichtlich, dass der RSA-Algorithmus mit einer mittleren Initiierungszeit von $\approx 2,59\,s$ wesentlich länger braucht, als der EC ($\approx 75,05\,ms$). Darüber hinaus zeigen die starken Ausschläge im Kurvenverlauf, dass eine starke Abweichung von $\approx 792,08ms$ (30,62 %) vom Mittelwert (2,59s) auftritt. So Schwankt in den Messewerten die Initiierungszeit zwischen minimal $\approx 1,24\,s$ und maximal $\approx 5,56\,s$. Dies liegt, wie auch die insgesamt sehr lange Initiierungszeit, an der Auswahl der Primzahlen mit Hilfe eines Primzahltests.

Auf den ersten Blick erscheint es so, als ob die Zeiten im Client und Server identisch sind. Da die y-Achse im Diagramm jedoch einen wesentlich größeren Bereich anzeigen muss, damit alle Messwerte dargestellt werden, wird eine Abweichung von $\approx 100\,ms$ als Überlagerung der Linien dargestellt. Die Abweichung der Messwerte von Client und Server ist vergleichbar mit der des RSA mit vorberechneten Schlüsseln aus Abbildung 5.10. Auch hier ist der Server jeweils etwa 15 ms schneller als der Client. Dies hat die gleichen Gründe, wie bereits beim ECC beschrieben.

Bei einem Blick auf die Detailmessungen, in Abbildung 5.9 dargestellt, fällt direkt auf, dass 99 % der Initiierungszeit für die Schlüsselerzeugung (bspw. Singnaturschlüssel im Client: 22,66 % / 632,68ms) bzw. das Warten auf die Schlüsselerzeugung des Partners verwendet wird (Client: 50,04 % / 1397 ms, Server: 50,41 % / 1400 ms). Damit die restlichen Aktivitäten dargestellt werden können, wurden sie in der Aktivität „Bearbeite sym. Schlüssel“ / „Bearbeite Anfragen / Sende sym. Key“ zusammengefasst (Client: 1,97 % / 37,10 ms, Server: 1,47 % / 40,92 ms). Für eine genauere Aufschlüsselung der

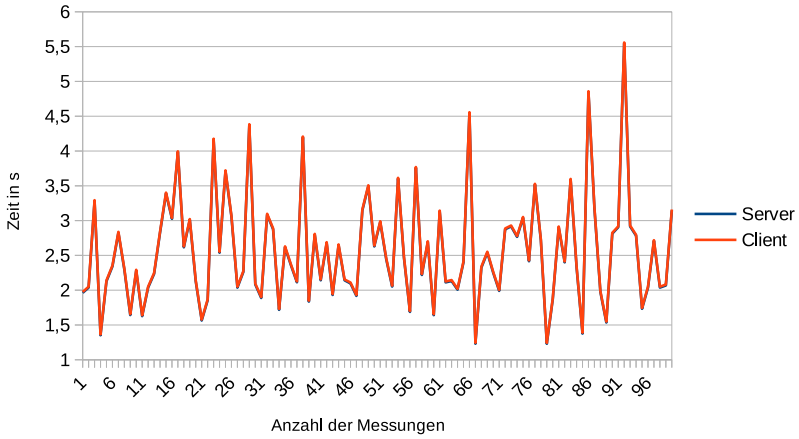


Abbildung 5.8: Darstellung der benötigten Zeit zur Initiierung der sicheren Kommunikation mit Hilfe des RSAs. Der Graph umfasst 100 Initiierungsvorgänge und schafft einen Eindruck über die Abweichung vom Mittelwert. Anmerkung: Die Messwerte liegen sehr dicht zusammen, sodass es aus den ersten Blick so erscheint, als ob es sich um eine Messreihe handelt.

einzelnen Laufzeiten kann die Detailmessung des RSA mit vorberechneten Schlüsseln verwendet werden.

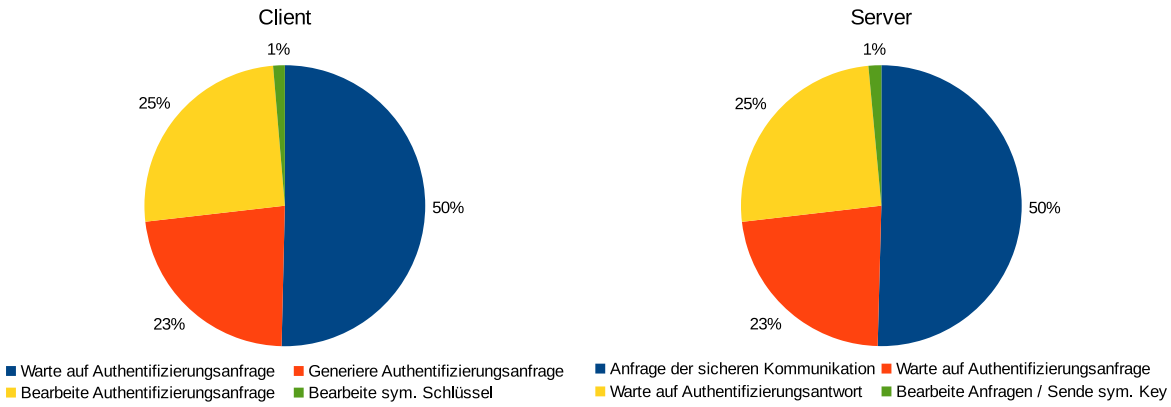


Abbildung 5.9: Detaildarstellung des Zeitverhaltens bei der Initiierung über das RSA-Verfahrens im Client (links) und im Server (rechts)

RSA1024 mit vorberechneten Schlüsseln

Nutzt man den RSA mit zuvor berechneten Schlüsseln, erhält man eine wesentlich bessere Performance gegenüber dem RSA ohne Schlüsselvorberechnung, wie in Abbildung 5.10 zu sehen ist. Hier beträgt die gesamte Initiierungszeit lediglich $\approx 99,52\,ms$ und ist damit in der Größenordnung des ECs. Auch die Standardabweichung fällt mit $\approx 1,80\,ms\,(1,80\,\%)$

deutlich geringer aus als ohne Vorberechnung der Schlüssel. Dies bestätigt die Erkenntnis, dass der wesentliche Anteil der Initiierungszeit des RSA auf die Berechnung des Schlüssels entfällt.

Auch hier ist bemerkenswert, dass der Client etwas mehr Zeit benötigt, was sich wieder damit erklären lässt, dass der Client zunächst die Kommunikation beginnt und schließlich Nachrichten bekommt die entschlüsselt werden müssen. Die restliche Zeit müssen beide Partner jedoch aufeinander warten.

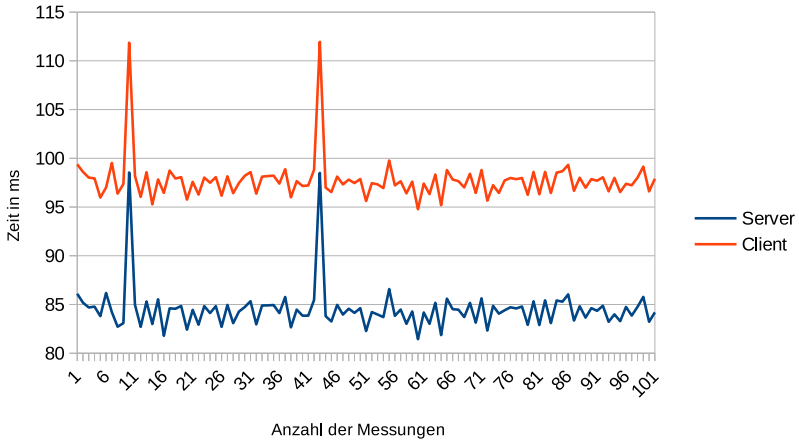


Abbildung 5.10: Darstellung der benötigten Zeit zur Initiierung der sicheren Kommunikation mit Hilfe des RSAs. Der Graph umfasst 100 Initiierungsvorgänge und schafft einen Eindruck über die Abweichung vom Mittelwert. Hier wurden die RSA-Schlüssel vor der Messung generiert.

Im Detail (Abbildung 5.11) ist hier auffällig, dass die Aktivitäten, die eine verschlüsselte oder signierte Nachricht übertragen, wesentlich mehr Zeit brauchen, als im EC. Bereits das Generieren einer Authentifizierungsanfrage (Client: 12,89 % / 5,82 ms, Server: 7,07 % / 5,94 ms) und vor allem das Warten auf diese benötigt einen signifikanten Anteil an der Initiierung (Client: 12,89 % / 17,16 ms, Server: 7,07% / 17,15 ms). Hier werden öffentlichen Schlüssel und die Authentifizierungsanfrage selbst auf dem Bus übertragen werden, was aufgrund der Schlüssellänge die Wartezeit verursacht.

Außerdem ist auffällig, dass der Server wesentlich länger zum Bearbeiten einer Authentifizierungsanfrage (15,08 %, 12,67 ms) benötigt, jedoch kürzer auf die Antwort des Clients wartet (12,55 %, 10,54 ms). Im Client sind diese Werte umgekehrt (Bearbeitung: 8,44 % / 11,23 ms, Warten: 20,88 %, 27,79 ms). Dies wird durch die grünen und gelben Tortenstücke in 5.11 dargestellt. Ein möglicher Erklärungsansatz dafür ist, dass der Client immer etwas schneller auf den Bus schreibt, als der Server dadurch die Aktivität „Bearbeite Authenntifizierungsanfrage“ schneller abschließt. Der Server hingegen möchte auf den Bus senden, empfängt seinerseits jedoch zu diesem Zeitpunkt die Authentifizierungsantwort des Clients, wodurch er im CANSend bzw. Treiberaufruf verharrt. Erst

wenn er die Nachricht vom Client empfangen hat, ist der Bus frei und er kann senden.

Das Prüfen der empfangenen Nachricht verläuft jedoch recht schnell (Client: 5,94 % / 7,91 ms, Server: 9,63 % / 8,08 ms), da hier lediglich je einmal verifiziert und entschlüsselt wird. Besonders drastisch fällt die Zeit für das Senden des AES-Schlüssels und AES-IVs ins Gewicht (24,00 % /20,16 ms), da dies je eine Verschlüsselung und Signatur beinhaltet, was fast ein Viertel der Zeit ausmacht. Der Client muss diese Zeit warten, benötigt danach nochmals Zeit, um die Schlüssel zu entschlüsseln.

Auch hier kommt die lange Sendezeit des MAC-Schlüssels (11,26 % / 9,46 ms) gegenüber der kurzen Empfangszeit des MAC-Schlüssels (2,03 % / 2,70 ms) vermutlich daher, dass während der Entschlüsselung des AES-Schlüssels bereits der MAC-Schlüssel empfangen wird und so fast direkt vorliegt. Die kürzere Bearbeitungszeit von MAC (4,83 % / 6,42 ms) gegenüber AES (13,02 % / 17,33 ms) ist, wie bereits beschrieben, darauf zurückzuführen, dass der MAC-IV im aktuellen Prototypen hartkodiert ist, der AES-IV jedoch nicht.

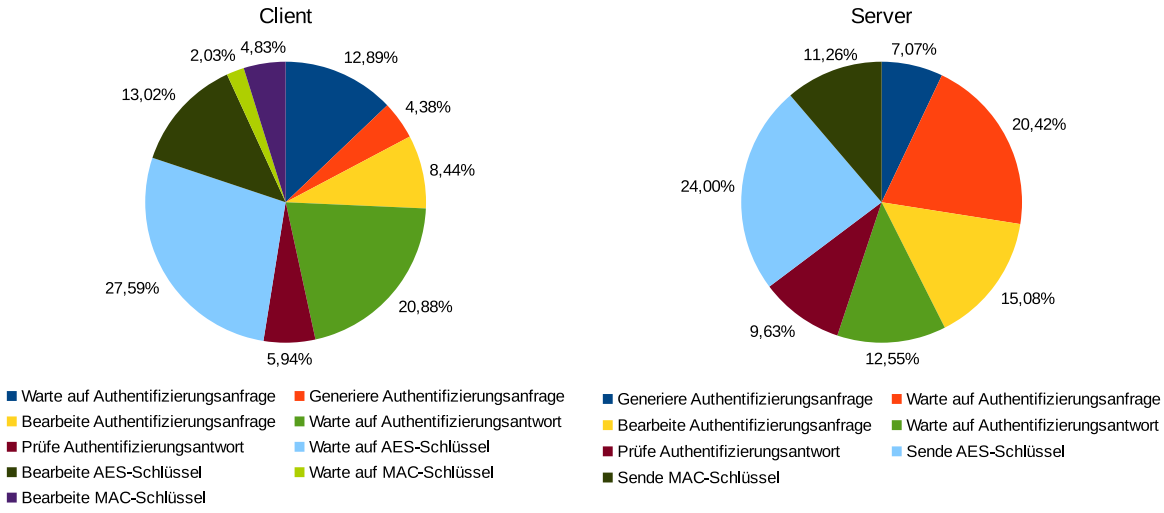


Abbildung 5.11: Detaildarstellung des Zeitverhaltens bei der Initiierung über das RSA-Verfahrens mit vorberechneten Schlüsseln im Client (links) und im Server (rechts)

Anmerkung: Wie bereits beschrieben, wird die Schlüsselberechnung vor der Kommunikationsanfrage durchgeführt. Es ist jedoch wichtig zu sehen, dass diese Zeit nicht „verschwunden“ ist, sondern lediglich vorgezogen wurde und daher im Diagramm nicht dargestellt wird.

RSA2048

Wie bereits erwähnt, kommt in den vorgestellten Messungen die RSA-Variante mit lediglich 1024 Bit Schlüsseln zum Einsatz. Testmessungen mit der 2048 Bit-Version zeigen, dass nicht nur die Schlüsselgenerierung bis zu 8 Sekunden (Schnitt ca. 5,5 s) dauert und

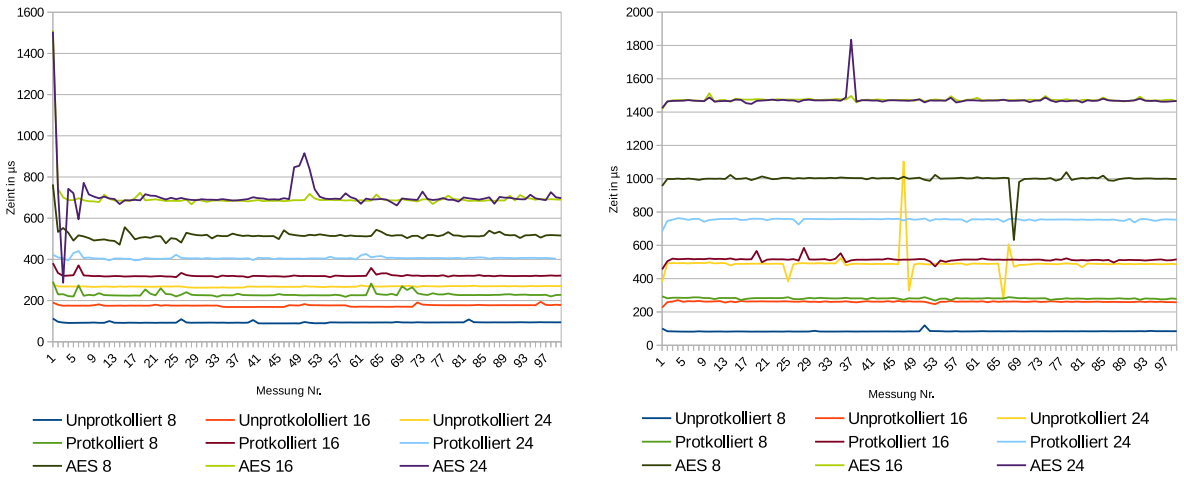


Abbildung 5.12: Gegenüberstellung der unprotokollierten, protokollierten und verschlüsselten Übertragung je eines Datenvolumens von 8, 16 und 24 Byte. Links das Senden, rechts das Empfangen.

damit fast zehnmal so lange wie bei der Variante mit 1024 Bit. Auch die Verschlüsselung mit 1024 Bit benötigt nur ein Drittel der Zeit, die Entschlüsselung sogar nur ein Fünftel. Hinzu kommt noch, die durch die längeren Schlüssel gestiegene Anzahl von CAN-Frames zum Schlüsseltausch[2].

5.2.3 Nachrichtenübertragung

Für die sichere Kommunikation wurde die Nachrichtenübertragung mit dem AES zwischen zwei Teilnehmern gemessen. Da es hier keine Unteraktivitäten gibt, wurde auf Kreisdiagramme für die Detailmessungen verzichtet.

AES

[h] In diesem Abschnitt soll die komplette Übertragung einer AES-Nachricht behandelt werden. Hierbei wurden Nachrichten abwechselnd verschlüsselt, versendet und vom Kommunikationspartner entschlüsselt. Dieser verschlüsselt einer weitere, versendet sie, usw. Gemessen wird dabei jeweils die Dauer der Ver- und Entschlüsselns sowie die Dauer, bis eine Antwort eintrifft. Dabei wird deutlich, dass der AES mit 44,81 μs für das Entschlüsseln bzw. 49,61 μs sehr schnell verläuft. Die Dauer, bis eine Antwort eintrifft, beträgt mit 3,84 ms und ist damit deutlich länger. Somit dauert es rein rechnerisch 1,83 ms bis ein Frame verschlüsselt übertragen wird. Dies erscheint vor allem in dem Kontext, dass ein Frame nur 300 μs benötigt, sehr lang. Jedoch ist hier zu beachten, dass aufgrund des

[2]Um den Umfang des Kapitels nicht unnötig zu vergrößern, sollen die Messwerte an dieser Stelle nur kurz zusammengefasst und nicht durch Diagramme dargestellt werden.

Präfixprotokolls, was zwei Byte extra benötigt, und des AES Paddings, was die Nachricht auf 16 Byte auffüllt, insgesamt 3 Frames übertragen werden müssen. Außerdem wird die Protokollierung vier Mal durchlaufen, was ebenfalls zu einer Verzögerung führt.

Die Gegenüberstellung in Abbildung 5.12 zeigt die Zeit, die eine Nachricht benötigt, um versendet (links) bzw. empfangen zu werden (rechts). Hierbei wurden jeweils Nachrichten mit 8, 16 und 24 Byte Datenvolumen übertragen und die benötigte Zeit für die reine Datenübertragung, protokollierte- und die verschlüsselte Übertragung gemessen. Die Messung endet in dem Moment, wenn der Treiberaufruf zurück kommt. Diese Messungen entsprechen dem realen Anwendungsfall bei CAN, da aus Sicht des Programms keine Informationen des Busses zur Verfügung stehen.

Das Diagramm stellt ein zu erwartendes Bild dar. Zunächst ist die unprotokollierte Datenübertragung von 8 und 16 Byte die schnellste. Bemerkenswert ist jedoch, dass die Übertragung von 8 Byte protokollierten Datenvolumens (2 Frames) bereits schneller ist, als die von drei unprotokollierten Frames. Schließlich folgen die drei AES-Varianten. Interessant ist hierbei, dass die 16 und 24 Byte Varianten nahezu identische Laufzeiten haben. Dies lässt sich durch das Padding erklären, dass Nachrichten auf 32 Byte padded. Dadurch entstehen gleiche Verschlüsselungs- und Übertragungszeiten.

Beim Entschlüsseln der Nachricht zeigt sich im Großen und Ganzen ein ähnliches Bild. Allerdings sind alle Operationen teils deutlich langsamer. Dies trifft vor allem auf die verschlüsselten und protokollierten Nachrichten zu. Hieraus kann geschlossen werden, dass der Prototyp bei der Protokollierung noch verbessert werden kann.

Schlüsselupdate

Die benötigte Zeit für ein Schlüsselupdate wurde nicht explizit gemessen. Je nach gewähltem Verfahren können hierfür jedoch die Messwerte für die Initiierung oder Kommunikation auf das Schlüsselupdate eines Teilnehmers übertragen werden. So lässt sich zum Beispiel das Verteilen eines neuen Schlüssels an einen Client mit Perfect Foreward Secrecy mit der Zeit des Initiierens der sicheren Kommunikation gleichsetzen. Wird eine Verschlüsselung der neuen Sitzungsschlüssel mit den alten Sitzungsschlüsseln gewählt, dürfte die benötigte Zeit ungefähr das Zwölffache der Zeit eines AES-Frames (3,84 ms) benötigen, da zwölf mal so viele Frames übertragen werden müssen, der Zeitbedarf für die Verschlüsselung jedoch nicht stark ins Gewicht fällt.

Hierbei muss jedoch beachtet werden, dass es sich lediglich um Werte für einen Client handelt. Für genauere Zeiten und mehrere Clients sollten weitere Messungen angestellt werden. Insbesondere die Parallelisierbarkeit der genannten Abläufe im Server spielt hierbei eine Rolle.

5.2.4 Zusätzliche Busbelastung

Um die zusätzlich Busbelastung zu ermitteln, wurde die Anzahl der CAN-Frames während der Messungen gezählt. Tabelle 5.1 stellt die ermittelten Werte dar.

	Client	Server
EC	31	64
RSA	103	196
AES	3	3

Tabelle 5.1: Tabellarische Darstellung der für eine Initiierung und die Übertragung einer sicheren Nachricht (<= 8 Byte) benötigten CAN-Frames

Es ist direkt klar, dass die Anzahl der Frames, die bei der Initiierung der sicheren Kommunikation übertragen werden, eine zusätzliche Busbelastung darstellen, die sonst nicht anfallen würde. Auch hier schneidet das Elliptic Curves-Verfahren deutlich besser ab als der RSA. Dies ist durch die deutlich kürzeren Schlüssellängen, die übertragen werden müssen, zu begründen. Außerdem erzeugt die symmetrische Verschlüsselung einen deutlich geringeren Verschlüsselungsoverhead.

Dies führt voraussichtlich auch bei Systemen mit mehreren Teilnehmern zu einer besseren Performance gegenüber dem RSA.

Im Falle der AES-verschlüsselten Kommunikation lässt sich der Kommunikationsoverhead leicht durch die Treppenfunktion $y = (x + (16 - (x \mod 16) + 2))/8$ beschreiben. Hier steht die 16 für das Paddingoverhead und die 2 für die vom Präfixprotokoll benötigten Verwaltungsdaten. Das y liefert die Anzahl der benötigten Frames bei einer Datenmenge von x. Dagegen kann die normale Nachrichtenübertragung mit $y = (x + 2 + (8 - (x + 2 \mod 8)))/8$ beschrieben werden. Betrachtet man die beiden Formeln, so wird klar, dass die verschlüsselte Kommunikation mit steigender Nachrichtenlänge effizienter wird, was an den Konstanten in den Formeln liegt.

5.3 Zusammenfassung der Ergebnisse

Zusammenfassend kann man für die Initiierung einer sicheren Kommunikation eine klare Empfehlung für das Verfahren mit elliptischen Kurven geben. Es ist mit einem hinreichenden Sicherheitsniveau immer noch schneller als das RSA-Verfahren mit einem als zu kurz anzusehenden Schlüssel. Hinzu kommt, dass bei dem betrachteten EC-Verfahren

die Schlüssel während der Initiierung berechnet werden, beim RSA jedoch vorberechnet werden müssen.

Im Detail ist zu sehen, dass die Verwendung eines symmetrischen Verschlüsselungsverfahrens auf der Basis des gemeinsamen Geheimnisses deutliche Vorteile liefert. Auch wenn zunächst fast ein Drittel auf das Erstellen des gemeinsamen Geheimnisses entfällt, kann durch die schnellere Verschlüsselung Zeit eingespart werden. Dies ist auch bei der Anzahl der übertragenen CAN-Frames deutlich. Hier benötigt der EC ebenfalls lediglich ein Drittel der CAN-Frames, die für das RSA-Verfahren aufgewendet werden müssen.

Mit einem hinreichenden Sicherheitsniveau ist das RSA-Verfahren aufgrund von horrenden Authentifizierungszeiten schon in einer 1:1-Kommunikation kaum praktikabel. Zudem werden wesentlich mehr Nachrichten benötigt, was die Buslast bei mehreren Teilnehmern stark erhöht. Hieraus sind weitere Verzögerungen anzunehmen.

Im Falle der sicheren Übertragung fällt vor allem die Übertragung von deutlich mehr Frames bei kurzer Nachrichtenlänge ins Gewicht. Es müssen bis zu dreimal mehr Nachrichten übertragen werden, als ohne Verschlüsselung und Authentifizierung. Der AES-Algorithmus wird auf den Testgeräten sehr schnell ausgeführt, sodass kaum eine Verzögerung bei der symmetrischen Ver- und Entschlüsselung, sowie der Bilden und Verifizieren das MACs feststellbar war.

Kapitel 6

Fazit

In diesem Kapitel werden zunächst die in der Arbeit erreichten Ergebnisse zusammengefasst. Des Weiteren soll eine Bewertung erfolgen, die auf Basis der Ergebnisse eine grundsätzliche Aussage über die sichere Kommunikation über das Controller Area Network liefert.

Schließlich soll ein Ausblick auf wichtige Fragen und offen gebliebene Punkte gegeben werden, die während der Bearbeitung entstanden sind, jedoch nicht Bestandteil dieser Arbeit wurden.

6.1 Zusammenfassung

Die Aufgabe des Erstellens eines Konzepts zur sicheren Kommunikation über CAN wurde zusammenfassend in drei Teilprobleme unterteilt. So wurde zunächst eine theoretische Beschreibung des geforderten Sicherheitsniveaus angefertigt, um eine Übersicht der durchzuführenden Aktivitäten zu erhalten. Hierfür wurden mit der Elliptic Curves Cryptography und dem RSA-Kryptosystem zwei aktuelle Verfahren zur Initiierung gewählt. Die sichere Nachrichtenübertragung wurde mit dem Advanced Encryption Standard erreicht.

Um einen Anhaltspunkt über die grundsätzliche Realisierbarkeit zu erhalten, wurde eine prototypische Implementierung auf Basis eines Kommunikationsmodells erstellt. In der Implementierung konnte ein Sicherheitsniveau erreicht werden, das nahezu dem in Kapitel 3 beschriebenen entspricht.

Dieser Prototyp diente schließlich als Basis für Messungen, die eine konkrete Aussage über den Zeitfaktor und die zusätzliche Busbelastung lieferten. Hierbei konnte ein Überblick über die grundsätzliche Perfomance der gewählten Verfahren gegeben werden. Außerdem war ein detaillierterer Einblick in den Zeitbedarf der einzelnen Aktivitäten möglich.

Abgerundet wird dies durch den Bezug auf die Eigenschaften von Embedded Systems, beispielsweise durch eine statische Speicherverwaltung. Themen aus dem Bereich

der IT-Sicherheit wurden ebenfalls berücksichtigt. So wurde beispielsweise die Perfect Foreward Secrecy und dem Problem, der Verteilung des Master Secrets behandelt. Zusammenfassend konnte so eine fundierte Entscheidungsbasis für eine Realisierung einer sicheren Kommunikation über CAN gegeben werden.

6.2 Bewertung

Die Bewertung der Erkenntnisse, die diese Arbeit liefert, hängen vom jeweiligen Anwendungsbereich ab. Die Entscheidung über die Realisierung eines Sicherheitsniveaus sollte im Einzelfall getroffen werden. Es lassen sich jedoch einige generelle Schlussfolgerungen hinsichtlich der Sicherheit und Realisierbarkeit daraus ziehen, die hier vorgestellt werden sollen.

So kann grundsätzlich angenommen werden, dass eine hinreichend sichere Kommunikation nach dem Stand der Technik auf dem CAN-Bus durchführbar ist. Hierbei bietet das vorgestellte Kommunikationsmodell eine Möglichkeit, wie eine M:N-Kommunikation realisiert werden kann. Das vorgestellte Initiierungsverfahren mit Hilfe von elliptischen Kurven stellt dabei ein deutlich effizienteres kryptografisches System dar als das RSA-Verfahren.

Es wird auch klar, dass vor allem ein hohes Sicherheitsniveau, wie z.B. die Forderung der Perfect Foreward Secrecy mit regelmäßigem Schlüsseltausch, kaum realisierbar ist. Dies folgt aus der Tatsache, dass viele Systemressourcen für die Berechnung der Schlüssel und Verschlüsselung während der Initiierung aufgewendet werden müssen. Außerdem steigt die Buslast während der Initiierung stark an. Dieses Problem ist bereits in dem Testaufbau mit einer Eins-zu-eins-Kommunikation sichtbar und wird sich bei einer M:N-Kommunikation oder langsameren Busteilnehmern weiter verschärfen.

Eine Verschlüsselung mit Hilfe des AES erfordert zwar ebenfalls einen gewissen Mehraufwand für die Berechnung und Kommunikation, der jedoch in einem Rahmen bleibt, welcher für viele Systeme tolerierbar sein dürfte. Hier sollte eine teilweise realisierung des Sicherheitsniveaus in Betracht gezogen werden.

6.3 Ausblick

Auch wenn einige Aspekte des Themas bearbeitet werden konnten und aussagekräftige Ergebnisse liefern, bleiben offene Punkte, die Gegenstand weiterer Forschungen sein können. Hier sollen einige dieser Punkte, die im Rahmen der Bearbeitung auffielen, kurz beschrieben werden.

6.3.1 Performance und Zeitverhalten

Diese Arbeit bietet eine Gegenüberstellung eines Verfahrens basierend auf Elliptic Curves und dem RSA-Kryptosystem mit einer Empfehlung für die Elliptic Curves Cryptography. Es konnte jedoch keine genauere Analyse hinsichtlich Kurvenparametern mehr durchgeführt werden. Außerdem bleibt, wie bereits beschrieben, die Untersuchung der Performance von mehreren Kommunikationsteilnehmern und Domains. Auch der Einfluss von unterschiedlichen CAN-IDs auf die Initiierungszeiten sollte einer genaueren Betrachtung unterzogen werden. Dies ist vor allem interessant, wenn andere Teilnehmer gleichzeitig unverschlüsselt kommunizieren.

Außerdem sollte die Performance auf Systemen mit unterschiedlichen Prozessoren, Speichergrößen und Betriebssystemen untersucht werden, da in einem realen Aufbau ebenfalls verschiedene Systeme zum Einsatz kommen. Auch die Kommunikation zwischen langsamen und schnellen Teilnehmern kann hier von Bedeutung sein. Hierbei ist zu erwarten, dass die für kryptografische Operationen benötigte Zeit auf langsameren Prozessoren zunimmt, während die Zeit für die Nachrichtenübertragung ungefähr gleich bleibt.

6.3.2 CAN FD

Des Weiteren wurde im Jahr 2012 mit CAN FD ein Standard mit einer erhöhten Bandbreite von CAN entwickelt. Dieser umfasst ein Datenvolumen von bis zu 64 Byte und sollte die sichere Kommunikation deutlich beschleunigen. In Verbindung mit EC könnte es möglich sein, jede Nachricht in einem Frame zu versenden, was die Protokollierung überflüssig macht.

6.3.3 Angriffszenarien über andere CAN-Frames

Ein potentielles Sicherheitsrisiko stellen Angriffe über die Remote-, Error- und Overloadframes dar. Insbesondere über Errorframes lässt sich sehr leicht eine Denial-of-Service-Attacke durchführen, indem laufend Errorframes mit einer hohen Priorität gesendet werden. Diese „gewinnen“ die Arbitierung und überschreiben somit alle anderen Frames auf dem Bus. Dieses Problem des sogenannten „Babling Idiot“ kann nur durch eine zusätzliche Hardwarekomponente, den Buswächter, gelöst werden. Auch wenn hier keine vertraulichen Daten preisgegeben werden führt dies zu Fehlfunktionen.

6.3.4 Embedded Systems

Für einen Einsatz in Embedded Systems ist eine Aussage über Größer der Größe der Binärdatei wichtig, da im Allgemeinen wenig Speicher für Programmcode zur Verfügung

steht. Hierfür kann der Prototyp zur Analyse verwendet werden. Dabei sollte jedoch auf ein statisches Linken der OpenSSL Bibliotheken geachtet werden.

Anhang A

Schematische Darstellung zum Erreichen des Sicherheitsniveaus

In diesem Anhang ist eine schematische Darstellung zum erreichen eines möglichst hohen Sicherheitsniveaus aufgeführt. Dazu wird in Tabellenform auf die durchzuführenden Schritte eingegangen, die erforderlichen Operationen und Schlüssel aufgeführt, und deren Zweck knapp erläutert. Eine genau Beschreibung der einzelnen Schritte ist im Kapitel 3 zu finden.

Glossar zu diesem Abschnitt

Grundsätzliches:

Abkürzung:	Bedeutung:
[Nachricht]	Nachrichten bestehend aus mehreren Teilnachrichten
X = Y ?	Prüfung auf Gleichheit
NONCE	Number used only once

Instanzen:

Abkürzung:	Bedeutung:
c1	Client 1
c2	Client 2
s	Server

Parameter:

Abkürzung:	Bedeutung:
P_{c2}	Passwort des Clients c2
h_{pc2}	Hashwert des Passwortes von Client c2
M	Nachricht
pk_{c2a}	Asymmetrischer, öffentlicher Schlüssel von c2 zur Authentifizierung
sk_{c2a}	Asymmetrischer, privater Schlüssel von c2 zur Authentifizierung
pk_{c2e}	Asymmetrischer, öffentlicher Schlüssel von c2 zur Verschlüsselung
sk_{c2e}	Asymmetrischer, privater Schlüssel von c2 zur Verschlüsselung
$k_{cryptd1}$	Symmetrischer Schlüssel zur Verschlüsselung der Domain D1
k_{macd1}	Symmetrischer Schlüssel zur Berechnung des MACs der Domain D1
shsec, $shsec_s$, $shsec_c$	gemeinsames Geheimnis, Serverteil / Clientteil des gemeinsamen Geheimnisses

(Die symmetrischen Schlüssel k stehen sowohl für Schlüssel als auch für den Initialisierungsvektor)

Algorithmen:

Abkürzung:	Bedeutung:
H(P)	Bildet Hashwert des Parameters P
$ASYM_{ENC}(pk_{sa}, M)$	Asymmetrisches Verschlüsseln der Nachricht M mit Schlüssel pk_{sa}
$ASYM_{DEC}(sk_{sa}, M)$	Asymmetrisches Entschlüsseln der signierten Nachricht M mit Schlüssel sk_{sa}
$ASYM_{SIGN}(sk_{sa}, M)$	Asymmetrisches Signieren der Nachricht M mit Schlüssel sk_{sa}
$ASYM_{VER}(pk_{sa}, M)$	Asymmetrisches Verifizieren der signierten Nachricht M mit Schlüssel pk_{sa}
$SYM_{ENC}(k_{cryptd1}, M)$	Symmetrisches Verschlüsseln von M mit Schlüssel kcryptd1
$MAC(k_{macd1}, M)$	Bilden eines MACs von M mit dem Schlüssel K_{macd1}
$SHSEC_{GEN}()$	Generieren eines Shared Secret (optional mit Teilgeheimnis $shsec_s$ oder $shsec_c$ als Parameter)

Anmerkung: c2 und d1 dienen hier lediglich als Beispiele

Authentifizierung eines Clients am Server mit RSA

	Server				Client			
Schritt:	Beschreibung	Bekannte Daten	Formel	Übertragung	Formel	Bekannte Daten	Beschreibung	Bemerkungen:
Auth RSA1	Server generiert: Symmetrischen Schlüssel zur Verschlüsselung: $k_{cryptd1}$ Symetrischen Schlüssel zur Authentifizierung: k_{macd1}	P_{c1} $k_{cryptd1}$, k_{macd1}				P		Das Passwort P und der Hashwert hpc1 sind als Shared Secrets vorab ausgetauscht worden.
AuthRSA 2	Server generiert Asymmetrischen Schlüssel zur Authentifizierung: pk_{sa}, sk_{sa} und zur Verschlüsselung: pk_{se}, sk_{se}	P_{c1} pk_{sa}, sk_{sa} pk_{se}, sk_{se} $k_{cryptd1}$, k_{macd1}				P pk_{c1a}, sk_{c1a} pk_{c1e}, sk_{c1e}	Client generiert: Asymmetrischen Schlüssel zur Authentifizierung: pk_{c1a}, sk_{c1a} und zur Verschlüsselung: pk_{c1e}, sk_{c1e}	
AuthRSA 3		P_{c1} pk_{sa}, sk_{sa} pk_{se}, sk_{se} pk_{c1a} pk_{c1e} $k_{cryptd1}$, k_{macd1}		Client sendet: [sec_com_req || [sk_{c1a}, sec_com_req]] und pk_{c1a}, pk_{c1e}	$ASYM_{SIGN}(sk_{c1a}$, sec_com_req)	P pk_{c1a}, sk_{c1a} pk_{c1e}, sk_{c1e}	Client sendet: Anfrage zur sicheren Komm. Signiert mit sk_{c1a}, öffentlichen Schlüssel zur Authentifizierung und öffentlichen Schlüssel zur Verschlüsselung	Die Signatur dient dazu, dass ein Angreifer die Kommunikationsdomain nicht verändern kann.
AuthRSA 4	Server verifiziert: Anfrage zur sicheren Kommunikation	P_{c1} pk_{sa}, sk_{sa} pk_{se}, sk_{se} pk_{c1a} pk_{c1e} $k_{cryptd1}$, k_{macd1}	$ASYM_{VER}(pk_{c1a}$, sec_com_req) sec_com_req			P pk_{c1a}, sk_{c1a} pk_{c1e}, sk_{c1e} pk_{sa} pk_{se}		Challange-Response-Verfahren
AuthRSA 5	Server sendet Anfrage zur Authentifizierung zusammen mit einen Nonce signiert mit sk_{sa} und verschlüsselt mit pk_{c1e} und sendet pk_{sa}, pk_{se}	P_{c1} pk_{sa}, sk_{sa} pk_{se}, sk_{se} pk_{c1a} pk_{c1e} $k_{cryptd1}$, k_{macd1}	$ASYM_{ENC}(pk_{c1e}$, auth_req || NONCE) $ASYM_{SIGN}(sk_{sa}$, [pk_{c1e}, auth_req || NONCE])	Server sendet: [[pk_{c1e}, [auth_req || NONCE]] || [sk_{sa}, [pk_{c1e}, auth_req || NONCE]]]] und pk_{sa}, pk_{se}		P pk_{c1a}, sk_{c1a} pk_{c1e}, sk_{c1e} pk_{sa} pk_{se}		Challange-Response-Verfahren

Glossar: siehe Seite 1 dieses Anhangs

Fortsetzung nächste Seite

Authentifizierung eines Clients am Server mit RSA

	Server				Client			
Schritt:	Beschreibung	Bekannte Daten	Formel	Übertragung	Formel	Bekannte Daten	Beschreibung	Bemerkungen:
AuthRSA 5		P_{c1} pk_{sa}, sk_{sa} pk_{se}, sk_{se} pk_{c1a} pk_{c1e} $k_{cryptd1}$, k_{macd1}			$ASYM_{VER}(pk_{sa}, [sk_{sa}[auth\_req \| \| NONCE])$ $ASYM_{DEC}(sk_{c1e}, [sk_{sa}[auth\_req \| \| NONCE])$ auth_req || NONCE	P pk_{c1a}, sk_{c1a} pk_{c1e}, sk_{c1e} pk_{sa} pk_{se}	Client verifiziert: Nachricht mit öffentlichem Schlüssel pk_{sa} Client entschlüsselt: Nachricht mit privatem Schlüssel sk_{c1e} und bekommen auth_req und Nonce.	Wir gehen von einer erfolgreichen Prüfung der Signatur aus.
AuthRSA 6		P_{c1} pk_{sa}, sk_{sa} pk_{se}, sk_{se} pk_{c1a} pk_{c1e} $k_{cryptd1}$, k_{macd1}		Client sendet: $[h_{(P \| \| NONCE)} \| \| [sk_{c1a}[h_{(P \| \| NONCE)}]]]$	$H(P \| \| NONCE)$ $ASYM_{SIG}(sk_{c1a}, [H(P \| \| NONCE)])$	P pk_{c1a}, sk_{c1a} pk_{c1e}, sk_{c1e} pk_{sa} pk_{se}	Client sendet: Hashwert von Passwort P verbunden mit der Nonce Verschlüsselt mit pk_s Signiert mit sk_{c1a}	
AuthRSA 7	Server prüft: Passwort P_{c1}, indem er den gesendeten Hashwert verifiziert und entschlüsselt und das gespeicherte Passwort P mit der Nonce hasht, und die Erg. Vergleicht.	P_{c1} pk_{sa}, sk_{sa} pk_{se}, sk_{se} pk_{c1a} pk_{c1e} $k_{cryptd1}$, k_{macd1}	$ASYM_{VER}([sk_{c1a}[pk_{se}\ h_{(P \| \| NONCE)}]])$ $h_{(P \| \| NONCE)} = H(P_{c1} \| \| NONCE)$?			P pk_{c1a}, sk_{c1a} pk_{c1e}, sk_{c1e} pk_{sa} pk_{se}		Wir gehen von einer erfolgreichen Prüfung des Passwortes aus.
AuthRSA 8	Server sendet: $K_{cryptd1}$ und k_{macd1} jeweils verschlüsselt mit pk_{c1e} und authentisiert mit sk_{sa}	P_{c1} pk_{sa}, sk_{sa} pk_{se}, sk_{se} pk_{c1a} pk_{c1e} $k_{cryptd1}$, k_{macd1}	$k_{cryptd1}$ $ASYM_{ENC}(pk_{c1e}, [k_{cryptd1}])$ $ASYM_{SIG}(sk_{sa}, [pk_{c1e}[k_{cryptd1}]])$ k_{macd1} $ASYM_{ENC}(pk_{c1e}, [k_{macd1}])$ $ASYM_{SIG}(sk_{sa}, [pk_{c1e}[k_{macd1}]])$	Server sendet: $[[pk_{c1e}[k_{cryptd1}]] \| \| [sk_{sa}[pk_{c1e}[k_{cryptd1}]]]]$ und $[[pk_{c1e}[k_{macd1}]] \| \| [sk_{sa}[pk_{c1e}[k_{macd1}]]]]$		P pk_{c1a}, sk_{c1a} pk_{c1e}, sk_{c1e} pk_{sa} pk_{se}		
AuthRSA 9		P_{c1} pk_{sa}, sk_{sa} pk_{se}, sk_{se} pk_{c1a} pk_{c1e} $k_{cryptd1}$, k_{macd1}			$ASYM_{VER}(pk_{sa}, [sk_{sa}[pk_{c1e}[k_{cryptd1}]]])$ $ASYM_{DEC}(sk_{c1e}, [pk_{c1e}[k_{cryptd1}]])$ $k_{cryptd1}$ und $ASYM_{VER}(pk_{sa}, [sk_{sa}[pk_{c1e}[k_{macd1}]]])$ $ASYM_{DEC}(sk_{c1e}, [pk_{c1e}[k_{macd1}]])$ k_{macd1}	P pk_{c1a}, sk_{c1a} pk_{c1e}, sk_{c1e} pk_{sa} pk_{se} $k_{cryptd1}$, k_{macd1}	Client prüft: Authentizität des Servers mit pk_{sa} Entschlüsselt Nachricht mit sk_{c1e}	Ab diesem Punkt kann der Client an der sicheren Kommunikation teilnehmen

Glossar: siehe Seite 1 dieses Anhangs

Authentifizierung eines Clients am Server mit Hilfe von elliptischen Kurven

	Server				Client			
Schritt:	Beschreibung	Bekannte Daten	Formel	Übertragung	Formel	Bekannte Daten	Beschreibung	Bemerkungen:
Auth EC1	Server generiert: Symmetrischen Schlüssel zur Verschlüsselung: $k_{cryptd1}$ Symetrischen Schlüssel zur Authentifizierung: k_{macd1}	P_{c1} $k_{cryptd1}$, k_{macd1}				P		Das Passwort P und der Hashwert hpc1 sind als Shared Secrets vorab ausgetauscht worden.
AuthEC2	Server generiert Asymmetrischen Schlüssel zur Authentifizierung: pk_{sa}, sk_{sa}	P_{c1} pk_{sa}, sk_{sa} $k_{cryptd1}$, k_{macd1}				P pk_{c1a}, sk_{c1a}	Client generiert: Asymmetrischen Schlüssel zur Authentifizierung: pk_{c1a}, sk_{c1a}	
AuthEC3		P_{c1} pk_{sa}, sk_{sa} pk_{c1a} $k_{cryptd1}$, k_{macd1}		Client sendet: [sec_com_req || [sk_{c1a}, sec_com_req]] und Pk_{c1a}	$ASYM_{SIGN}(sk_{c1a}, sec\_com\_req)$	P pk_{c1a}, sk_{c1a}	Client sendet: Anfrage zur sicheren Komm. Signiert mit sk_{c1a}, öffentlichen Schlüssel zur Authentifizierung	Die Signatur dient dazu, dass ein Angreifer die Kommunikationsdomain nicht verändern kann.
AuthEC4	Server verifiziert: Anfrage zur sicheren Kommunikation	P_{c1} pk_{sa}, sk_{sa} pk_{c1a} $k_{cryptd1}$, k_{macd1}	$ASYM_{VER}(pk_{c1a}, sec\_com\_req)$ sec_com_req			P pk_{c1a}, sk_{c1a}		
AuthEC5	Server generiert: Shared Secret für diese Anfrage	P_{c1} pk_{sa}, sk_{sa} pk_{c1a} $k_{cryptd1}$, k_{macd1}	$SHSEC_{GEN}()$ $shsec_s$	Server sendet: $shsec_s$ und pk_{sa}		P pk_{c1a}, sk_{c1a}		

Glossar: siehe Seite 1 dieses Anhangs
Fortsetzung nächste Seite

Authentifizierung eines Clients am Server mit Hilfe von elliptischen Kurven

	Server				Client			
Schritt:	Beschreibung	Bekannte Daten	Formel	Übertragung	Formel	Bekannte Daten	Beschreibung	Bemerkungen:
AuthEC5		P_{c1} pk_{sa}, sk_{sa} pk_{c1a} $k_{cryptd1}$, k_{macd1}		Client sendet: $Shsec_c$	$SHSEC_{GEN}(shsec_s)$ shsec, $shsec_c$	P pk_{c1a}, sk_{c1a} shsec	Client generiert: Mit Teilgeheimnis vom Server das Gesamtgeheimnis und sein Teilgeheimnis Client sendet: Sein Teilgeheimnis an den Server	
AuthEC6	Server generiert: Mit Teilgeheimnis vom Client das gemeinsame Geheimnis	P_{c1} pk_{sa}, sk_{sa} pk_{c1a} $k_{cryptd1}$, k_{macd1} shsec	$SHSEC_{GEN}(shsec_c)$ shsec			P pk_{c1a}, sk_{c1a} shsec		
AuthEC7	Server sendet Anfrage zur Authentifizierung zusammen mit einen Nonce signiert mit sk_{sa} und verschlüsselt mit pk_{c1e} und sendet pk_{sa}, pk_{se}	P_{c1} pk_{sa}, sk_{sa} pk_{c1a} $k_{cryptd1}$, k_{macd1} shsec	SYM_{ENC}(shsec, auth_req || NONCE) $ASYM_{SIGN}(sk_{sa}$, [shsec, auth_req || NONCE])	Server sendet: [[pk_{c1e}, [auth_req || NONCE]] || [sk_{sa}, [pk_{c1e}, auth_req || NONCE]]]] und pk_{sa}, pk_{se}		P pk_{c1a}, sk_{c1a} shsec		Challange-Response-Verfahren
AuthEC8		P_{c1} pk_{sa}, sk_{sa} pk_{c1a} $k_{cryptd1}$, k_{macd1} shsec			$ASYM_{VER}(pk_{sa}$, [sk_{sa}[auth_req || NONCE]) SYM_{DEC}(shsec, [sk_{sa}[auth_req || NONCE]) auth_req || NONCE	P pk_{c1a}, sk_{c1a} shsec	Client verifiziert: Nachricht mit öffentlichem Schlüssel pk_{sa} Client entschlüsselt: Nachricht mit privatem Schlüssel sk_{c1e} und bekommen auth_req und Nonce.	Wir gehen von einer erfolgreichen Prüfung der Signatur aus.
AuthEC9		P_{c1} pk_{sa}, sk_{sa} pk_{c1a} $k_{cryptd1}$, k_{macd1} shsec		Client sendet: [$h_{(P \,||\, NONCE)}$ || [sk_{c1a} [$h_{(P \,||\, NONCE)}$]]]	H(P || NONCE) $ASYM_{SIG}(sk_{c1a}$, [H(P || NONCE)])	P pk_{c1a}, sk_{c1a} shsec	Client sendet: Hashwert von Passwort P verbunden mit der Nonce Verschlüsselt mit pk_s Signiert mit sk_{c1a}	

Glossar: siehe Seite 1 dieses Anhangs
Fortsetzung nächste Seite

Authentifizierung eines Clients am Server mit Hilfe von elliptischen Kurven

	Server				Client			
Schritt:	Beschreibung	Bekannte Daten	Formel	Übertragung	Formel	Bekannte Daten	Beschreibung	Bemerkungen:
AuthEC10	Server prüft: Passwort P_{c1}, indem er den gesendeten Hashwert verifiziert und entschlüsselt und das gespeicherte Passwort P mit der Nonce hasht, und die Erg. Vergleicht.	P_{c1} pk_{sa}, sk_{sa} pk_{c1a} $k_{cryptd1}$, k_{macd1} shsec	$ASYM_{VER}([sk_{c1a}[pk_{se}\ h_{(P \mid\mid NONCE)}]])$ $h_{(P \mid\mid NONCE)} = H(P_{c1} \mid\mid NONCE)$?			P pk_{c1a}, sk_{c1a} shsec		Wir gehen von einer erfolgreichen Prüfung des Passwortes aus.
AuthEC11	Server sendet: $K_{cryptd1}$ und k_{macd1} jeweils verschlüsselt mit pk_{c1e} und authentisiert mit sk_{sa}	P_{c1} pk_{sa}, sk_{sa} pk_{c1a} $k_{cryptd1}$, k_{macd1} shsec	$k_{cryptd1}$ $SYM_{ENC}(shsec, [k_{cryptd1}])$ $ASYM_{SIG}(sk_{sa}, [pk_{c1e}[k_{cryptd1}]])$ k_{macd1} $SYM_{ENC}(shsec, [k_{macd1}])$ $ASYM_{SIG}(sk_{sa}, [pk_{c1e}[k_{macd1}]])$	Server sendet: $[[shsec\ [k_{cryptd1}]] \mid\mid [sk_{sa}\ [shsec\ [k_{cryptd1}]]]]$ und $[[shsec\ [k_{macd1}]] \mid\mid [sk_{sa}\ [shsec\ [k_{macd1}]]]]$		P pk_{c1a}, sk_{c1a} shsec		
AuthEC12		P_{c1} pk_{sa}, sk_{sa} pk_{c1a} $k_{cryptd1}$, k_{macd1} shsec			$ASYM_{VER}(pk_{sa}, [sk_{sa}[shsec\ [k_{cryptd1}]]])$ $SYM_{DEC}(shsec, [shsec\ [k_{cryptd1}]])$ $k_{cryptd1}$ und $ASYM_{VER}(pk_{sa}, [sk_{sa}[shsec\ [k_{macd1}]]])$ $SYM_{DEC}(shsec, [shsec\ [k_{macd1}]])$ k_{macd1}	P pk_{c1a}, sk_{c1a} shsec, $k_{cryptd1}$, k_{macd1}	Client prüft: Authentizität des Servers mit pk_{sa} Entschlüsselt Nachricht mit shsec	Ab diesem Punkt kann der Client an der sicheren Kommunikation teilnehmen

Glossar: siehe Seite 1 dieses Anhangs

Keyupdate aller Clients einer Domain

	Server				Alle Clients der Domain D1			
Schritt:	Beschreibung	Bekannte Daten	Formel	Übertragung	Formel	Bekannte Daten	Beschreibung	Bemerkungen:
KeyUpdate1	Server sendet: stop_com_req signiert mit sk_{sa}	P_{c1} pk_{sa}, sk_{sa} pk_{c1X} pk_{c1X} $k_{cryptd1}$, $k_{cryptd1neu}$ k_{macd1}, $k_{macd1neu}$	$ASYM_{SIG}(pk_{sa}, stop\_com\_req)$	Server sendet: [stop_com_req || [sk_{sa}[stop_com_req]]]		P_{cX} pk_{cXa}, sk_{cXa} pk_{cXe}, sk_{cXe} pk_{sa}, pk_{se} $k_{cryptd1}$ k_{macd1}		Dieser Schritt wird bei Initialisierung eines neuen Clients parallel zu Schritt Auth8 und zyklisch durchgeführt.
KeyUpdate2		P_{c1} pk_{sa}, sk_{sa} pk_{c1X} pk_{c1X} $k_{cryptd1}$, $k_{cryptd1neu}$ k_{macd1}, $k_{macd1neu}$				P_{cX} pk_{cXa}, sk_{cXa} pk_{cXe}, sk_{cXe} pk_{sa}, pk_{se} $k_{cryptd1}$ k_{macd1}	Clients stoppen: interne Kommunikation	Die symmetrischen Schlüssel sind bereits in Schritt Auth7 generiert worden.
KeyUpdate3	Server sendet: Neue symmetrische Schlüssel jeweils verschlüsselt mit pk_{c1e} und authentisiert mit sk_{sa}	P_{c1} pk_{sa}, sk_{sa} pk_{c1X} pk_{c1X} $k_{cryptd1}$, $k_{cryptd1neu}$ k_{macd1}, $k_{macd1neu}$	$k_{cryptd1neu}$ $ASYM_{ENC}(pk_{c1e}, k_{cryptd1neu})$ $ASYM_{SIG}(sk_{sa}, [pk_{c1e}[k_{cryptd1neu}]])$ und $k_{mac1neu}$ $ASYM_{ENC}(pk_{c1e}, k_{mac1neu})$ $ASYM_{SIG}(sk_{sa}, [pk_{c1e}[k_{mac1neu}]])$	Server sendet: [[pk_{c1e} [$k_{cryptd1neu}$]] || [sk_{sa} [pk_{c1e} [$k_{cryptd1neu}$]]]] und [[pk_{c1e}[$k_{macd1neu}$]] || [sk_{sa} [pk_{c1e}[$k_{macd1neu}$]]]]		P_{cX} pk_{cXa}, sk_{cXa} pk_{cXe}, sk_{cXe} pk_{sa}, pk_{se} $k_{cryptd1}$, $k_{cryptd1neu}$ k_{macd1}, $k_{macd1neu}$		
KeyUpdate4		P_{c1} pk_{sa}, sk_{sa} pk_{c1X} pk_{c1X} $k_{cryptd1}$, $k_{cryptd1neu}$ k_{macd1}, $k_{macd1neu}$			$ASYM_{VER}(pk_{sa}, [sk_{sa}[pk_{cXe}[k_{cryptd1neu}]]])$ $ASYM_{DEC}(sk_{c1e}, [pk_{cXe}[k_{cryptd1neu}]])$ $k_{cryptd1neu}$ und $ASYM_{VER}(pk_{sa}, [sk_{sa}[pk_{cXe}[k_{MAC1}]]])$ $ASYM_{DEC}(sk_{cXe}, [pk_{cXe}[k_{macd1}]])$ $k_{macd1neu}$	P_{cX} pk_{cXa}, sk_{cXa} pk_{cXe}, sk_{cXe} pk_{sa}, pk_{se} $k_{cryptd1}$, $k_{cryptd1neu}$ k_{macd1}, $k_{macd1neu}$	Clients prüfen: Authentizität des Servers mit pk_{sa} Entschlüsselt Nachricht mit sk_{cXe}	
KeyUpdate5		P_{c1} pk_{sa}, sk_{sa} pk_{c1X} pk_{c1X} $k_{cryptd1}$, $k_{cryptd1neu}$ k_{macd1}, $k_{macd1neu}$				P_{cX} pk_{cXa}, sk_{cXa} pk_{cXe}, sk_{cXe} pk_{sa}, pk_{se} $k_{cryptd1}$, $k_{cryptd1neu}$ k_{macd1}, $k_{macd1neu}$	Clients ersetzen: $k_{cryptd1}$ durch $k_{cryptd1neu}$ k_{macd1} durch k_{macd1}	Clients nehmen danach die Sichere Kommunikation wieder auf

Anmerkung: Das 'X' symbolisiert, dass für jeden Client ein Schlüssel mit einer ID im Server vorliegt
Dieses Beispiel bezieht sich auf RSA als Kryptosystem. EC ist aquivalnt abgesehen von symmetrischer Verschlüsselung

Glossar: siehe Seite 1 des AnhangsGlossar: siehe Seite 1 des Anhangs

Sichere Kommunikation

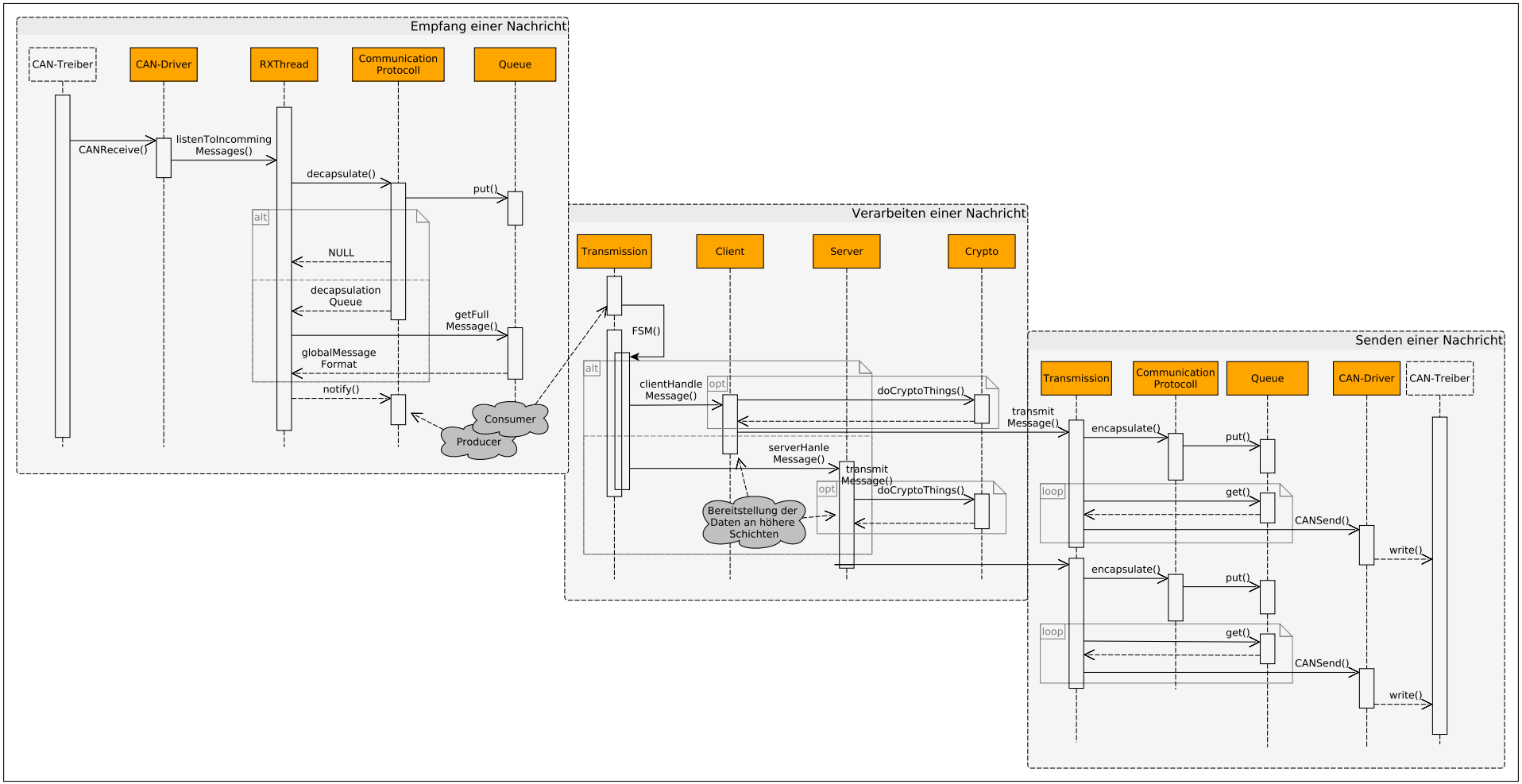
	Client 1			Übertragung	Client 2			
Schritt:	Beschreibung	Bekannte Daten	Formel		Formel	Bekannte Daten	Beschreibung	Bemerkungen:
Sec-Com1	Client 1 berechnet: Verschlüsselt M symmetrisch mit $k_{cryptd1}$ Authentifiziert: verschlüsselte Nachricht mit MAC-Verfahren und Schlüssel k_{macd1}	P_{c1} pk_{c1a}, sk_{c1a} pk_{c1e}, sk_{c1e} pk_{sa}, pk_{se} $k_{cryptd1}$ k_{macd1}	M $SYM_{ENC}(k_{cryptd1}, M)$ $MAC(k_{macd1}, [k_{cryptd1}, M])$			P_{c2} pk_{c2a}, sk_{c2a} pk_{c2e}, sk_{c2e} pk_{sa}, pk_{se} $k_{cryptd1}$ k_{macd1}		Annahme: beide Clients sind bereits authentifiziert.
Sec-Com2	Client 1 sendet: Verschlüsselte Daten verbunden mit dem Message Authentication Code der verschlüsselten Daten	P_{c1} pk_{c1a}, sk_{c1a} pk_{c1e}, sk_{c1e} pk_{sa}, pk_{se} $k_{cryptd1}$ k_{macd1}		Client 1 sendet: $[SYM_{ENC}(k_{cryptd1}, M)$ $\|\|$ $MAC(k_{macd1}, [k_{cryptd1}, M])]$		P_{c2} pk_{c2a}, sk_{c2a} pk_{c2e}, sk_{c2e} pk_{sa}, pk_{se} $k_{cryptd1}$ k_{macd1}		|| = einfaches Aneinanderhängen der Nachrichten
Sec-Com3		P_{c1} pk_{c1a}, sk_{c1a} pk_{c1e}, sk_{c1e} pk_{sa}, pk_{se} $k_{cryptd1}$ k_{macd1}			$MAC_{received} = MAC(k_{macd1}, [k_{cryptd1}, M])$? Wenn gleich: $SYM_{DEC}(k_{cryptd1}, M)$	P_{c2} pk_{c2a}, sk_{c2a} pk_{c2e}, sk_{c2e} pk_{sa}, pk_{se} $k_{cryptd1}$ k_{macd1}	Client2 überprüft: Selbst erstellte MAC mit empfangener. Entschlüsselt: Nachricht M mit Verschlüsselungsschlüssel	$MAC_{received}$ ist die empfangene MAC.

Glossar: siehe Seite 1 dieses Anhangs

Anhang B

Sequenzdiagramm des Prototypen

In diesem Anhang ist das Sequenzdiagramm des Prototypen nochmals komplett aufgeführt, sodass die Zusammenhänge besser dargestellt werden. Dieses Diagramm wird in Unterabschnitt 4.3.3 genauer beschrieben und in Teilen dargestellt.


Empfang einer Nachricht
CAN-Treiber
CAN-Driver
RXThread
Communication Protocoll
Queue
CANReceive()
listenToIncomming Messages()
decapsulate()
put()
alt
NULL
decapsulation Queue
getFull Message()
globalMessage Format
notify()
Consumer
Producer
Verarbeiten einer Nachricht
Transmission
Client
Server
Crypto
FSM()
alt
clientHandle Message()
opt
doCryptoThings()
serverHanle Message()
transmit Message()
opt
doCryptoThings()
Bereitstellung der Daten an höhere Schichten
Senden einer Nachricht
Transmission
Communication Protocoll
Queue
CAN-Driver
CAN-Treiber
transmit Message()
encapsulate()
put()
loop
get()
CANSend()
write()
encapsulate()
put()
loop
get()
CANSend()
write()

Abkürzungsverzeichnis

AES Advanced Encryption Standard

CAN Controller Area Network

CAN FD Controller Area Network Fexible Data Rate

CBC Cipher Block Chaining

CMAC Cipher-Based Message Authentication Code

CSMA/CR Carrier Sense Multiple Access / Collision Resolution

DHKE Diffie Hellman Key Establishment

DSA Digital Signature Algorithm

EC Elliptic Curves

ECC Elliptic Curves Cryptography

ECDH Elliptic Curves Diffie Hellman

ECDSA Elliptic Curves Digital Signature Algorithm

FSM Finite State Machine

HDLC High-Level Data Link Control

HMAC Keyed-Hash Message Authentication Code

IND-CPA Ciphertext Indistinguishability (Ununterscheidbarkeit der Geheimtexte)

IV Initialisierungsvektor

KDF Key Derivation Function

MAC Message Authentication Code

NIST National Institute of Standards and Technology

OAEP Optimal Asymmetric Encryption Padding

PFS Perfect Foreward Secrecy

PKCS Public Key Cryptography Standards

RSA Rivest, Shamir, Adelman Kryptoverfahren

SHA Secure Hash Algorithm

Literaturverzeichnis

[AD14] AUTOMATION&DRIVES (Hrsg.): *A&D Vorsprung Automation*. www.aud24.net/media/automation-drives/aud-2014.../aud-2014-03.pdf, 2014. – (Online abgerufen: 02.06.2015)

[ARM14] ARM: *ARM Cortex-A53 MPCore Processor Technical Reference Manual*, 2014

[Bos91] Bosch: *CAN Specification Version 2.0*, 1991

[BSI05] BSI: *IT-Sicherheitsmanagement und IT-Grundschutz BSI-Standards zur IT-Sicherheit*. Bundesanzeiger, 2005

[BSI14] BSI: *BSI - Technische Richtlinie Kryptografische Verfahren: Empfehlung und Schlüssellängen*. Bundesanzeiger, 2014

[Buc10] Buchmann, Johannes: *Einführung in die Kryptographie*. Springer-Verlag, 2010

[Dai15] Daimler, AG: *Smart App Center*. https://www.smart.com/de/de/index/app-center/app-center.html, 2015. – (Online abgerufen: 02.06.2015)

[Eck13] Eckert, Claudia: *IT-Sicherheit: Konzepte-Verfahren-Protokolle*. Oldenbourg Verlag, 2013

[Ets01] Etschberger, Konrad: *Controller area network: basics, protocols, chips and applications*. Ixxat press, 2001

[G+05] Group, OSEK u. a.: *OSEK/VDX Operating System Specification 2.2. 3*, 2005

[Hot13] Hottelet, Ulrich: *Industrie ist gegen Hacker schlecht gerüstet*. http://www.welt.de/wirtschaft/webwelt/article120916015/Industrie-ist-gegen-Hacker-schlecht-geruestet.html, 2013. – (Online abgerufen: 02.06.2015)

[Ker13] Kerrisk, Michael: *Linux Programmer's Manual*. http://man7.org/linux/man-pages/man2/clock_gettime.2.html, 2013. – [Online; Stand 11. Juli 2015]

[Nie94] Nielsen, Jakob: *Usability engineering*. Elsevier, 1994

[PP09] Paar, Christof ; Pelzl, Jan: *Understanding cryptography: A Textbook for Students and Practitioners*. Springer Science & Business Media, 2009

[Rei13] Reissmann, Ole: *Angriff auf Pkw-Software: Hacker steuern Autos fern*. http://www.spiegel.de/auto/aktuell/computerexperten-hacken-auto-software-a-914783.html, 2013. – (Online abgerufen: 02.06.2015)

[Tan09] Tanenbaum, Andrew S.: *Moderne Betriebssysteme*. Pearson Deutschland GmbH, 2009

[Tea15] Team, Linux Kernel D.: *Readme file for the Controller Area Network Protocol Family (aka SocketCAN)*. https://www.kernel.org/doc/Documentation/networking/can.txt, 2015. – [Online; Stand 9. Juni 2015]

[VMC02] Viega, John ; Messier, Matt ; Chandra, Pravir: *Network Security with OpenSSL: Cryptography for Secure Communications*. „ O'Reilly Media, Inc.“, 2002

[Wie12] Wietzke, Joachim: *Embedded Technologies: Vom Treiber bis zur Grafik-Anbindung*. Springer-Verlag, 2012

[Wik14a] Wikipedia: *Feldbus — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Feldbus&oldid=136425321, 2014. – [Online; Stand 3. Juni 2015]

[Wik14b] Wikipedia: *Topologie (Rechnernetz) — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Topologie_(Rechnernetz)&oldid=133613844, 2014. – [Online; Stand 3. Juni 2015]

[Wik15a] Wikipedia: *Controller Area Network — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Controller_Area_Network&oldid=141905852, 2015. – [Online; Stand 3. Juni 2015]

[Wik15b] Wikipedia: *Eingebettetes System — Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Eingebettetes_System&oldid=142492164. Version: 2015. – [Online; Stand 13. Juli 2015]

[Wik15c] Wikipedia: *Nonce — Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Nonce&oldid=138646701, 2015. – [Online; Stand 1. Juli 2015]

[WT06] Wietzke, Joachim ; Tran, Manh T.: *Automotive Embedded Systeme: Effizfientes Framework-Vom Design zur Implementierung*. Springer-Verlag, 2006