Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Design und Implementierung eines Konzepts zur eindeutigen Identifizierung von Softwarekomponenten auf Geräten im IIoT-Umfeld

Jan Güth

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer/in:	Dr. rer. nat. Frank Dürr, Dr.-Ing. Jan Schlechtendahl
Beginn am:	15. April 2019
Beendet am:	15. Oktober 2019

Kurzfassung

Unter dem Begriff Internet der Dinge (Internet of Things (IoT)) werden Konzepte und Technologien zusammengefasst, die physischen Geräten die Vernetzung mit dem Internet ermöglichen. Beim Begriff des klassischen IoT stehen in erster Linie die Vernetzung von Geräten im privaten Umfeld, wie z.B. Haushaltsgeräte, im Mittelpunkt. In den letzten Jahren haben sich jedoch einige Teilgebiete entwickelt, wie beispielsweise der Bereich des Industrial Internet of Things (IIoT), oft auch als Industrie 4.0 bezeichnet. Beim IIoT steht nicht der Verbraucher, sondern industrielle Prozesse und Abläufe im Mittelpunkt, die Probleme in den verschiedenen Teilgebieten sind jedoch meist dieselben. Eines dieser Probleme stellt die Identifizierung von Geräten und Software dar. Die Identifikation von Geräten mit eindeutigen und unveränderlichen Hardwaremerkmalen ist relativ leicht zu lösen, die eindeutige Identifikation von Software nicht.

Die vorliegende Arbeit befasst sich mit der Ausarbeitung eines Konzepts zur eindeutigen Identifizierung von Software auf verschiedenen Geräten. Dabei soll der Identifizierungsmechanismus zur Einhaltung von Softwarelizenzbedingungen verwendet werden können. Auch verschiedene Ausführungsumgebungen, wie beispielsweise virtuelle Maschinen, sollen im Konzept berücksichtigt werden. Des Weiteren wird untersucht, ob auch eine hardwarebasierte Sicherheitstechnologie wie Intel SGX zur Lösung des Identifikationsproblems eingesetzt werden kann. Durch eine prototypische Implementierung und eine abschließende Evaluierung wird gezeigt, dass der entwickelte Identifizierungsmechanismus eine Unterscheidung von verschiedenen Geräten und der darauf ausgeführten Software zulässt und somit auch zur Einhaltung von Softwarelizenzbedingungen verwendet werden kann.

Abstract

The term Internet of Things (IoT) covers concepts and technologies that enable physical devices to be connected to the Internet. The term IoT primarily refers to the networking of devices in private environments, such as household appliances. In recent years, however, some subareas have evolved, such as the Industrial Internet of Things (IIoT), often also referred to as Industry 4.0. IIoT does not focus on the consumer, rather on industrial processes and procedures, but the problems in the different subareas are usually the same. One of these problems is the identification of devices and software. The identification of devices with unique and unchangeable hardware characteristics is relatively easy to solve, the unique identification of software is not.

This thesis is concerned with the development of a concept for the unique identification of software on different devices. The identification mechanism should be able to be used to comply with software license conditions. Various execution environments, such as virtual machines, should also be considered in the concept. Furthermore, it will be investigated whether a hardware-based security technology such as Intel SGX can also be used to solve the identification problem. A prototypical implementation and a final evaluation show that the developed identification mechanism allows a distinction between different devices and the software executed on them and can therefore also be used to comply with software license conditions.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

AES	Advanced Encryption Standard
AIOTI	Alliance for Internet of Things Innovation
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BIOS	Basic Input/Output System
CA	Certification Authority
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DHS	Diffie-Hellman-Schlüsselaustausch
DLL	Dynamic Link Library
EDL	Enclave Definition Language
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IC	Integrated Circuit (dt. Integrierte Schaltkreise)
ID	Identifikator
Intel SGX	Intel Software Guard Extensions
IP	Internetprotokoll
IoT	Internet of Things (dt. Internet der Dinge)
IIoT	Industrial Internet of Things (dt. Industrie 4.0)
JNI	Java Native Interface
JVM	Java Virtual Machine

M2M	Machine-to-Machine
MAC-Adresse	Media-Access-Control-Adresse
MISK	Mechanismus zur eindeutigen Identifizierung von Softwarekomponenten
MQTT	Message Queue Telemetry Transport
Nonce	Number used once
OSHI	Operating System & Hardware Information
PC	Personal Computer
PSW	Intel SGX Platform Software Package
PUF	Physical Unclonable Function
RFID	Radio Frequency Identification
ROM	Read-only Memory
RPC	Remote Procedure Calls
RSA	Ron Rivest, Adi Shamir und Len Adleman
SaaS	Software as a Service
SDK	Software Development Kit
Server-Nonce	Kontrollserver-Nonce
SHA	Secure Hash Algorithm
SSD	Solid State Drive
SVN	Sicherheitsversionsnummer
SW-ID	Per Software definierter Identifikator
SWKP	Softwarekomponente
TCG	Trusted Computing Group
TCP	Transmission Control Protocol
TEE	Trusted Execution Environments
TLS	Transport Layer Security
TPM	Trusted Platform Module

URI	Unified Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
VMM	Virtual Machine Monitor
VPN	Virtual Private Network
WG	Working Group

1 Einleitung

Software ist heutzutage allgegenwärtig, sowohl im privaten als auch im industriellen Umfeld. Die verwendeten Geräte zur Ausführung der Software beschränken sich dabei schon lange nicht mehr auf den klassische Personal Computer (PC). Tablets und Smartphones gehören mittlerweile genauso zum Alltag, wie die Verwendung von cloudbasierten Softwareprodukten. Durch die Zunahme der Digitalisierung und Vernetzung kommen immer mehr verschiedene Geräte hinzu, auch solche, dessen eigentliche Aufgabe nicht die Ausführung von Software ist, wie beispielsweise Fernseher, Kühlschränke und Waschmaschinen. Zudem verfügen die meisten Geräte über Schnittstellen zur Vernetzung und zur Anbindung an das Internet. Allgemein wird diese Vernetzung von physischen Geräten und die Anbindung an das Internet unter dem Begriff Internet der Dinge (Internet of Things (IoT)) zusammengefasst. Im industriellen Umfeld, wo die Steuerung von Maschinen und Fertigungsprozessen im Vordergrund steht, wird der Begriff Industrie 4.0 (Industrial Internet of Things (IIoT)) verwendet [1]. Diese Vielzahl an verschiedenen Geräten und Ausführungsumgebungen stellt für die Softwareanbieter bei der Bereitstellung und Verwaltung von Software eine große Herausforderung dar.

Eine aktuelle Studie der *BSA The Software Alliance* [2] zeigt, dass 2018 weltweit 37% der Software unlizenziert in Verwendung ist. In Mittel- und Osteuropa liegt die Quote sogar bei 57%. Weltweit bedeutet das einen Umsatzverlust von 46,3 Milliarden US-Dollar. Ein zentrales Problem bei der Kontrolle und Verwaltung von vergebenen Lizenzen stellt die Identifizierung von ausgeführter Software dar. Eine häufig verwendete Methode ist, die ausgeführte Software anhand des verwendeten Gerätes zu identifizieren. Die Softwarelizenz wird durch einen Registrierungsvorgang fest einem Gerät zugeordnet. Ein solcher Mechanismus ist heute jedoch kaum mehr anwendbar. Zum einen nimmt die Verwendungsdauer eines Gerätes immer weiter ab. Zum anderen kommen immer häufiger Technologien, wie beispielsweise Virtualisierung- und Container-Technologien, zum Einsatz, die den physischen Zugriff auf ein Gerät verhindern. Somit ist auch die Identifizierung eines Gerätes nicht mehr ohne weiteres möglich.

Ein weiterer Trend geht dahin, Software immer häufiger als Service (Software as a Service (SaaS)) anzubieten, wo zur Verwendung der Software keine zusätzliche Installation auf dem eigenen Gerät nötig ist. Die Software wird über einen normalen Web-Browser verwendet, wie es bei E-Mail-Clients heute schon gängige Praxis ist. Der SaaS-Gedanke hat den großen Vorteil, dass die verwendete Ausführungsumgebung, die Installation und Bereitstellung eines Dienstes ganz in der Hand des Softwareanbieters liegt. Da zur Verwendung immer eine Internetverbindung bestehen muss, kann der Softwareanbieter die rechtmäßige Verwendung seines

angebotenen Dienstes über einen Authentifizierungsmechanismus jedes Mal erneut prüfen. Im industriellen Umfeld, wo durch die Software Geräte bzw. Maschine teilweise in Echtzeit gesteuert und überwacht werden müssen, ist der SaaS-Ansatz allerdings nur selten anwendbar.

1.1 Problemstellung und Zielsetzung

Ein allgemeines Problem im Bereich Internet der Dinge (Internet of Things (IoT)) und den Teilgebieten, wie Industrie 4.0 (Industrial IoT (IIoT)), stellt die eindeutige Identifizierung von Geräten und Softwarekomponenten dar [3]. Eine Softwarekomponente kann ein eigenständiges Softwareprodukt oder auch nur ein Teil eines verteilten Softwaresystems sein, wie z.B. ein einzelner Service eines Systems mit Microservice-Architektur. Während die Identifikation von Geräten mit eindeutigen und unveränderlichen Hardwaremerkmalen relativ einfach ist, stellt die eindeutige Identifikation von Softwarekomponenten, insbesondere in virtualisierten Umgebungen, eine große Herausforderung dar. Softwarekomponenten und virtuelle Umgebungen können leicht kopiert und vervielfältigt werden und somit auch die per Software definierten Identifikatoren (Software-defined identifier (SW-ID)), wie z.B. ein Lizenzschlüssel.

Der Wandel im IIoT-Umfeld geht dahin, dass Funktionalität in Form von Softwarekomponenten immer seltener mit dazugehöriger Hardware ausgeliefert wird. Vielmehr wollen Kunden ihre eigene, bereits vorhandene und in ihre Infrastruktur integrierte, Hardware für die Software verwenden. Dies führt dazu, dass die Hersteller der Softwarekomponenten keinen Einfluss und oftmals auch keinen Zugriff auf die darunterliegende Hardware haben. Des Weiteren entscheiden sich die Kunden immer häufiger für den Betrieb in einer Cloud-Umgebung, in welcher ein Zugriff auf die Hardware ebenfalls nicht möglich ist. Aus diesem Grund ist eine generelle Lösung, die daraufsetzt, eine Softwarekomponente über die darunterliegende Hardware zu identifizieren oder Mechanismen des Betriebssystems zur Erkennung mehrerer Instanzen einer Softwarekomponente nutzt, wie in Abbildung 1.1 dargestellt, nur bedingt geeignet.

Diese Heterogenität hat zur Folge, dass die Softwarekomponenten einen eigenen Identifikator benötigen. Über diesen können die Anfragen und Daten einer Softwarekomponente eindeutig zugeordnet werden und zugleich kann geprüft werden, ob die SW-ID gültig ist. Da eine SW-ID leicht kopiert und vervielfältigt werden kann, wird ein zusätzlicher Mechanismus zur Erkennung von mehrfach instanziierter Software mit gleicher SW-ID benötigt.

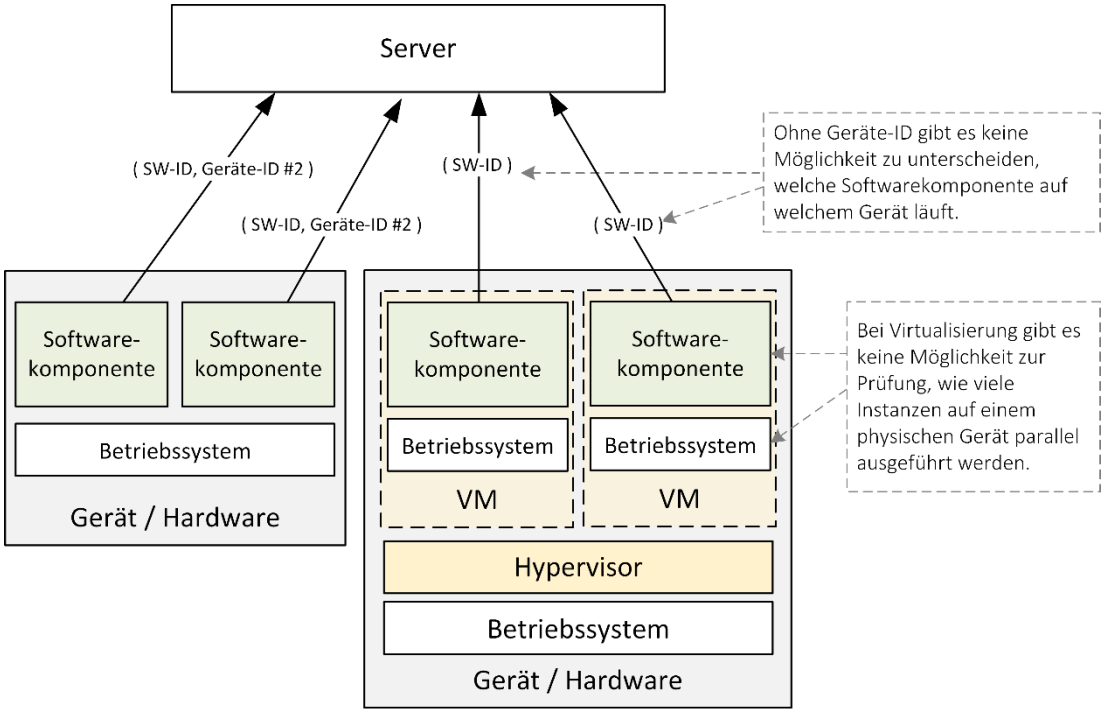

Abbildung 1.1: Problem bei der Identifizierung von Softwarekomponenten in virtualisierten Umgebungen ohne Geräte-Identifikator (Geräte-ID)

Ziel dieser Arbeit ist die Ausarbeitung eines Konzepts, welches es ermöglicht zu unterscheiden, ob zwei unterschiedlichen Anfragen mit der gleichen SW-ID von derselben Instanz, oder von zwei unterschiedlichen Instanzen einer Softwarekomponente kommen. Vorausgesetzt wird dafür, dass die verwendete SW-ID eindeutig ist. Hierfür soll die folgende Forschungsfrage beantwortet werden:

- Wie muss ein Mechanismus zur Erkennung mehrfach instanziierte Software mit gleicher SW-ID konzeptioniert sein?

Dabei sollen die folgenden Aspekte berücksichtigt werden:

- Wie kann die Zuordnung von Softwarekomponente und SW-ID eindeutig validiert werden?
- Wie ist die Erkennung mehrerer Instanzen mit gleicher SW-ID auf demselben physischen Gerät möglich?
- Wie kann das Konzept auch in virtualisierten Umgebungen angewendet werden?
- Können moderne hardwarebasierte Sicherheitstechnologien zur Lösung des Problems eingesetzt werden?

1.2 Aufbau der Arbeit

Die vorliegende Arbeit ist in sieben Kapitel gegliedert. Nach der **Einleitung** in diesem Kapitel, welches die Motivation, Problemstellung und Zielsetzung und die Gliederung dieser Arbeit beschreibt, werden im zweiten Kapitel wichtige **Grundlagen** für das weitere Verständnis der Arbeit vorgestellt. In Kapitel drei **Stand der Technik und verwandte Arbeiten** wird zunächst ein Überblick über den aktuellen Stand von Identifikatoren im IoT-Umfeld gegeben. Anschließend werden drei hardwarebasierte Sicherheitstechnologien vorgestellt und zwei Arbeiten, die eine weitere Möglichkeiten zur sicheren Ausführung bieten. Kapitel vier **Konzept** beinhaltet eine detaillierte Problembeschreibung, die Anforderungen an das Konzept, die Auswahl der hardwarebasierten Sicherheitstechnologie und die Beschreibung der konzeptionellen Lösung. In Kapitel fünf wird eine prototypische **Implementierung** des Konzepts vorgestellt. In Kapitel sechs **Evaluierung** werden die Anforderungen und das Konzept gegenübergestellt. Im siebten und letzten Kapitel **Zusammenfassung und Ausblick** werden die Ergebnisse dieser Arbeit nochmals betrachtet und ein Ausblick für zukünftige Arbeiten wird gegeben.

2 Grundlagen

In diesem Kapitel werden einige relevante Grundlagen vorgestellt, die zum besseren Verständnis späterer Teile dieser Arbeit wichtig sind. Im ersten Abschnitt wird das Konzept einer Trusted Execution Environment (TEE) vorgestellt. Anschließend werden der Diffie-Hellman-Schlüsselaustausch und Physical Uncloneable Functions (PUFs) näher beschrieben.

2.1 Trusted Execution Environments (TEE)

Eine Trusted Execution Environment (TEE) ist eine sichere, isolierte und vertrauenswürdige Ausführungsumgebung [4–7], die meistens als Teil des Hauptprozessor umgesetzt wird, kann jedoch auch ein eigenständiger Prozessor oder Chip (System-On-Chip) sein [6, 7]. Eine TEE wird dabei durch Kombination von Hardware- und Software-Funktionen auf einem System realisiert [5]. Von Seiten der Hardware muss für die sichere Ausführungsumgebung Speicher- und Rechenkapazität zur isolierten Ausführung vorhanden sein [7, 8]. Aufgrund dessen, dass die isolierte Ausführungsumgebung einer TEE nur einen begrenzten Speicherplatz hat, ist es empfehlenswert, nur bestimmte und sicherheitskritische Teile einer Anwendung in der TEE auszuführen [5]. Eine einheitliche Definition was eine TEE ist, existiert nicht [8], jedoch gibt es einen von der Industrie anerkannten Standard, der von der GlobalPlatform [9] spezifiziert wird [6, 10]. Nach Sabt, Achemlal & Bouabdallah muss eine TEE jedoch folgende Eigenschaften erfüllen: „*Die Bereitstellung einer manipulationssicheren Verarbeitungsumgebung, die auf einem separaten Kernel läuft*“ [8]. Des Weiteren wird von den Autoren definiert, dass die Authentizität des ausgeführten Codes, die Integrität der Laufzeitumgebung und die Vertraulichkeit von Code und Daten in einem persistenten Speicher gewährleistet sein muss, sowie die Möglichkeit, Code der in der TEE ausgeführt werden soll, durch eine dritte Partei überprüft werden kann (Remote Attestation) [8]. Zudem darf der Code nicht statisch sein und muss sicher aktualisiert werden können [8]. Ein beispielhafter Aufbau einer TEE mit den wichtigsten Komponenten ist in Abbildung 2.1 zu sehen.

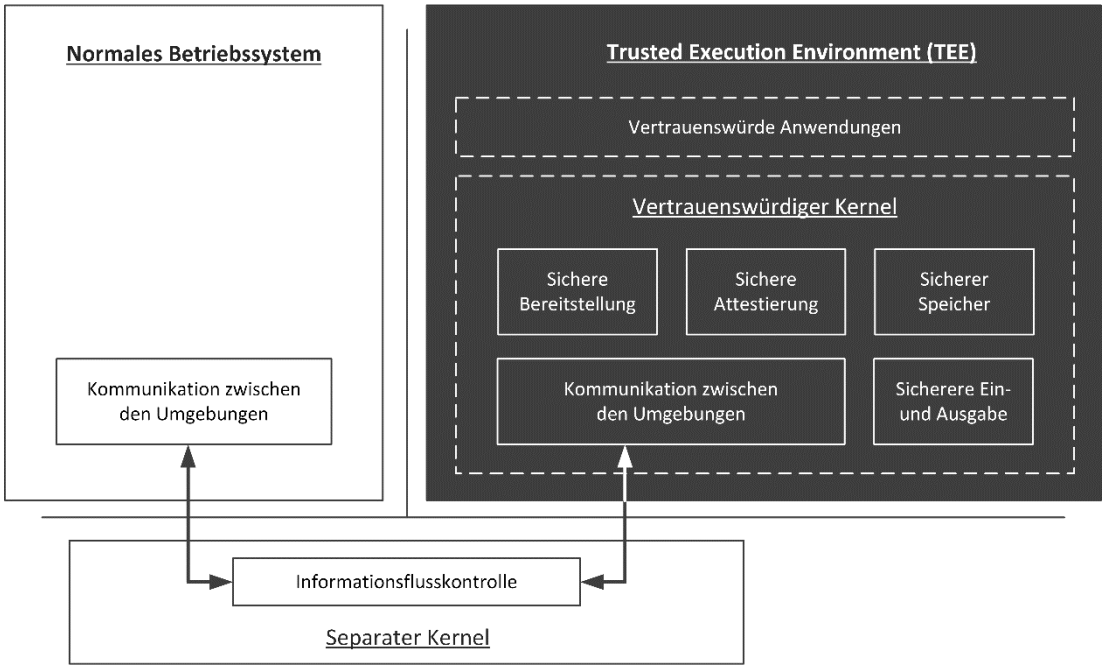

Abbildung 2.1: Aufbau und Bestandteile einer TEE Welt (Abbildung in Anlehnung nach [8])

Ein Anwendungsszenario für TEEs ist beispielsweise der Schutz von digitalen Inhalten. So kann beispielsweis beim Video-Streaming eine TEE dazu verwendet werden, dass Inhalte nur im gesicherten Bereich der TEE zwischengespeichert und verarbeitet werden, um so den Zugriff auf die Daten von außen zu verhindern und eine Vervielfältigung zu vermeiden [6]. Zwei der verbreitetsten Technologien die eine TEE realisieren, sind ARM TrustZone und Intel Software Guard Extensions (SGX) [11]. Diese beiden Technologien werden im Abschnitt 3.2 genauer behandelt.

2.2 Diffie-Hellman-Schlüsselaustausch (DHS)

Das von Whitfield Diffie und Martin Hellman entwickelte Verfahren zum geheimen Schlüsselaustausch [12, 13], wurde bereits 1976 in ihrer Veröffentlichung zur Public-Key-Kryptographie beschrieben [14].

Eine naheliegende Lösung zur verschlüsselten Kommunikation über einen unsicheren Kanal, wäre die Verwendung eines asymmetrischen Kryptographie-Verfahrens, wie dem RSA-Verfahren (nach seinen Erfindern Ron Rivest, Adi Shamir und Len Adleman benannt) [15]. Bei dieser Variante hat jeder Teilnehmer einen privaten und einen öffentlichen Schlüssel. Eine Nachricht kann nun vor dem Senden mit dem öffentlichen Schlüssel des Empfängers verschlüsselt werden und nur der Empfänger kann die Nachricht mit seinem geheimen Schlüssel wieder entschlüsseln [15]. Der Vorteil ist, dass vorher kein geheimer Schlüssel ausgetauscht werden muss. Der Nachteil von asymmetrischen Kryptographie-Verfahren im Vergleich zu symmetrischen Kryptographie-Verfahren ist jedoch, dass diese deutlich langsamer sind [13].

Aus diesem Grund wird der Diffie-Hellman-Schlüsselaustausch (DHS)[1] verwendet, um einen symmetrischen Sitzungsschlüssel über einen unsicheren Kanal auszutauschen, damit anschließend die Kommunikation mit einem symmetrischen Verfahren, wie beispielsweise dem Advanced Encryption Standard (AES), verschlüsselt werden kann [13].

Zur Durchführung des DHS-Protokolls müssen sich beide Teilnehmer, im Folgenden werden diese zur besseren Unterscheidung Alice und Bob genannt, auf eine möglichst große Primzahl p und eine natürlich Zahl g einigen. Beide Zahlen dürfen öffentlich bekannt sein und können daher über einen unsicheren Kanal ausgetauscht werden [12, 13]. Anschließend brauchen beide Teilnehmer je eine weitere geheime Zahl. Für Alice wird diese geheime Zahl a benannt und für Bob b. Für a und b werden dabei zufällig aus der Menge $\{0, 1, \dots, p-2\}$ gewählt [15].

Alice bildet nun eine weitere Zahl A durch Potenzieren der Basis g und durch die Anwendung der Modulo-Funktion mit p:

$$A = g^a \ mod \ p$$

Bob wendet die gleiche Funktion mit seiner Zahl b an, um B zu erhalten:

$$B = g^b \ mod \ p$$

Nun müssen die beiden Zahlen A und B zwischen Alice und Bob ausgetauscht werden. Auch dieser Austausch kann wieder über einen unsicheren Kanal vollzogen werden [12]. Anschließend potenzieren wieder beide ihre geheime Zahl a bzw. b mit der ausgetauschten Zahl A bzw. B als Basis und wenden wieder die Modulo-Funktion mit p an:

$$\text{Alice: } K = B^a \ mod \ p$$

$$\text{Bob: } K = A^a \ mod \ p.$$

Beide konnten nun durch die Kommutativität der Operatoren denselben Wert K berechnen und haben nun einen gemeinsamen geheimen Schlüssel, der zur symmetrischen Verschlüsselung verwendet werden kann. Der Austausch der Werte ist nochmals in Abbildung 2.2 verdeutlicht. Sollte Angreifer alle übertragenen Werte g, p, A und B durch Abhören des Kanals erlangt haben, müsste dieser nun den diskreten Logarithmus von A bzw. B zur Basis g berechnen, wofür es bis heute keine effiziente Methode gibt [12].

[1] Auch als Diffie-Hellman-Schlüsselvereinbarung bezeichnet [12, 16].

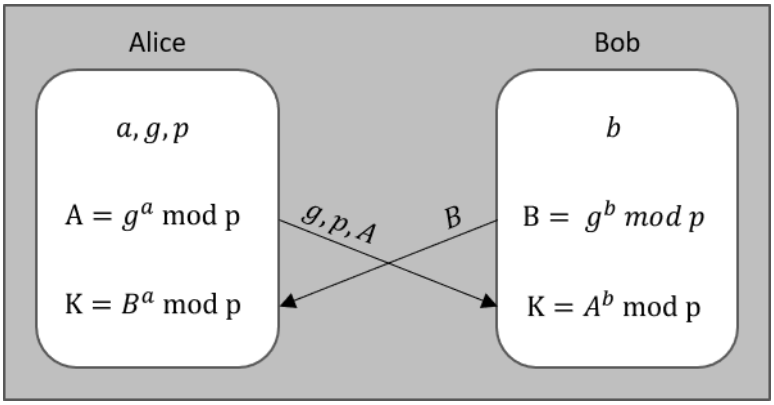

Abbildung 2.2: Prinzip des Diffie-Hellman-Schlüsselaustauschs

Eine weitere Möglichkeit für einen Angriff auf das DHS-Protokoll ist, neben der zuvor erwähnten Berechnung des Schlüssels K, der sogenannte Man-In-The-Middle-Angriff [15]. Bei diesem Angriff versucht ein Teilnehmer, welcher in diesem Szenario Marvin genannt wird, aktiv in das Protokoll einzugreifen [13]. Für den Angriff auf das DHS-Protokoll sitzt Marvin zwischen Alice und Bob. Will Alice nun die Werte g, p und A an Bob senden, fängt Marvin diese Werte ab und merkt sich diese. An Bob sendet Marvin nun seine selbst generierten Werte für g, p und A. Bob empfängt die Werte von Marvin im Glauben, dass diese von Alice sind, und berechnet B und den Schlüssel K. B sendet Bob zurück an Alice. B wird jedoch wieder von Marvin abgefangen, der sich nun denselben Wert K wie Bob berechnen kann. An Alice sendet Marvin nun einen eigenen erstellten Wert B, basierend auf den zuvor gesendeten Werten von Alice zurück, womit sich Alice nun auch einen Schlüssel K berechnen kann. Alice und Bob haben nun beide einen Schlüssel K, der jedoch unterschiedlich ist. Tauschen nun Alice und Bob im weiteren Verlauf verschlüsselte Nachrichten aus, kann Marvin, der nun beide Schlüssel kennt, diese abfangen und mit dem jeweilig passenden Schlüssel entschlüsseln. Die entschlüsselte Nachricht kann durch Marvin nun beliebig manipuliert oder ausgetauscht werden. Die geänderte Nachricht kann dann mit dem passenden Schlüssel für den Empfänger verschlüsselt und weitergesendet werden, ohne das Alice oder Bob etwas davon mitbekommen [13, 15].

2.3 Physical Unclonable Function (PUF)

Physikalisch nicht klonbare Funktionen (Physical Unclonable Functions (PUFs)) sind hardwarebasierte Bausteine zum Schutz von Informationen [17]. Das Ziel dieser Bausteine ist es, eine erhöhte Sicherheit beim Speichern von Schlüsseln und zur Erstellung bzw. Generierung von Schlüsseln für Verschlüsselungsverfahren bereitzustellen [17, 18].

Die Funktionsweise der PUFs basiert auf zufälligen Schwankungen in der Herstellung von integrierten Schaltkreisen (Integrated Circuits (ICs)) [17, 19]. Dabei ist der Produktionsprozess für die Bausteine immer gleich, jedoch wird durch die Eigenschaften der Materialien nie exakt dasselbe Ergebnis erreicht [19]. Dies ist Vergleichbar mit einem Fingerabdruck eines

Menschen. Dadurch kann jede PUF eindeutig identifiziert werden und die Einzigartigkeit der Merkmale, die durch den Produktionsprozess im Material entstanden sind, kann dazu verwendet werden, dass genau dasselbe Ergebnis nur mit der gleichen PUF reproduzierbar ist [18].

Zu den wichtigsten Eigenschaften einer PUF zählen, dass diese nicht klonbar ist, ein Ergebnis, das berechnet wird, nicht vorhersehbar ist und einen Manipulationsbeweis (Tamper Evidence) bietet [18]. Der erste Punkt, dass PUFs nicht klonbar sein dürfen, wird durch den bereits erwähnten Produktionsprozess erreicht, da durch diesen nicht exakt der gleiche Baustein reproduzierbar ist. Die nicht Vorhersehbarkeit eines Ergebnisses, dass mittels einer PUF erzeugt wird, wie beispielsweise einem Schlüssel, meint, dass ein Ergebnis für eine bestimme Eingabe nicht vorauszusagen ist, auch wenn mehrere Eingaben beobachtet wurden. Der Manipulationsbeweis bedeutet, dass eine PUF auf eine invasive (äußere) Einwirkung reagiert und entweder andere Ergebnisse liefert oder die gespeicherten Information ganz gelöscht werden [18].

Weitere Einsatzzwecke von PUFs sind die Verwendung zur Authentifizierung und zum Schutz von geistigem Eigentum (Intellectual Property) [17, 19]. Aber auch Dokumenten, wie beispielsweise einem Reisepass, und anderen Produkten, kann durch die Verwendung der Eigenschaft von PUFs ein fälschungssicheres Merkmal in Form eines Chips gegeben werden [17]

3 Stand der Technik und verwandte Arbeiten

Dieses Kapitel gibt zunächst einen Überblick über den aktuellen Stand zu verwendeten Identifikatoren im IoT-Bereich. Im zweiten Abschnitt werden verschiedene hardwarebasierte Sicherheitstechnologien vorgestellt, die später in Kapitel 4 auf die Verwendbarkeit für die gesuchte Lösung hin untersucht werden. Im dritten Abschnitt werden zwei weitere Arbeiten im Zusammenhang mit Trusted Execution Environments (TEE) vorgestellt.

3.1 Identifikatoren im IoT-Umfeld

Ein noch immer offener Punkt im IoT-Bereich ist, neben dem Fehlen von Standardisierungen, Skalierbarkeit, Interoperabilität, Zuverlässigkeit, Sicherheit und Datenschutz [20], die eindeutige Identifizierung von Geräten und Objekten, wie beispielsweise Daten [3, 20]. Bisher gibt es für die Identifikation keine einheitlichen Identifikationsmechanismen [21, 22]. Da der IIoT-Bereich bei der Identifizierung im Zusammenhang mit der Vernetzung von einem Gerät zu einem anderen (Machine-to-Machine (M2M)) die gleichen Probleme aufweist [23], wird zwischen IoT und IIoT in diesem Zusammenhang nicht weiter unterschieden.

Eine weit verbreitete Technologie zur eindeutigen Identifizierung und Verfolgung von physischen Objekten im IoT-Umfeld, ist die RFID-Technologie (Radio Frequency Identification Technologie) [3, 24]. Zur Identifizierung werden die auf einem kleinen passiven Chip (RFID-Chip) gespeicherten Daten über Radiowellen ausgelesen [24]. Durch die geringe Reichweite von nur einigen Metern eignet sich sie RFID-Technologie jedoch nur für Objekte in unmittelbarer Nähe und nicht beispielsweise zur Identifizierung in Netzwerken [24].

Zur allgemeinen Identifizierung von Geräten und Objekten, bzw. allgemeiner als Entität (Entity) bezeichnet [25], gibt es eine Vielzahl von Arbeiten die sich mit diesem Problem beschäftigen. Beispielsweise werden eigene Identifizierungsmechanismen durch Verwendung von IoT-Protokollen, wie MQTT (Message Queue Telemetry Transport) entwickelt [26], oder es werden adressbasierte Ontologien von URLs zur Identifikation verwendet [27]. Auch PUFs [28], Software-defined Networking (SDN) Ansätze [29], die Analyse von Kommunikationsdaten [30], Verwendung von Kontext-Bewusstsein (Context-Awareness) der Geräte [31] oder Ansätze, welche Blockchain-Technologien benutzen [32–34], werden verwendet. Die verschiedenen Lösungen behandeln dabei jedoch spezielle Probleme in bestimmten Bereichen.

Die Alliance for Internet of Things Innovation (AIOTI) ist ein Zusammenschluss von verschiedenen Firmen, mit dem Ziel Standardisierungen und Interoperabilität im Bereich IoT zu schaffen [35]. Die AIOTI organisiert sich verschiedenen Arbeitsgruppen zu Themen wie Smart Cities, Smart Mobility und Smart Manufacturing [35]. Mit der Arbeitsgruppe 3 (Working Group 3 (WG03)), werden allgemein Standardisierungen im Bereich IoT behandelt, wie beispielsweise von Identifikatoren [36]. Die Veröffentlichung der WG03, mit dem Titel „Identifiers in Internet of Things (IoT)“ (Identifikatoren im Internet der Dinge), fasst die verschiedenen Identifikatoren in verschieden Kategorien zusammen und gibt jeweils Standards an, die für eine Kategorie eingesetzt werden kann. AIOTI gibt dabei auch eine Definition an, was ein Identifikator ist: „*Ein Identifikator ist ein Muster (Pattern) zur eindeutigen Identifizierung einer einzelnen Entität (Instanz Identifikator) oder einer Klasse von Entitäten (z.B. Typ Identifikator) in einem bestimmten Kontext*“ [36]. Die Kategorien für Identifikatoren nach AIOTI sind wie folgt definiert:

- Thing-Identifikator (Thing Identifier):
  Ein Thing-Identifikator identifiziert eine Entität (Entity). Entitäten können physische Objekte, wie Sensoren, Maschinen, Fahrzeuge oder Gebäude sein, aber auch reine Datenobjekte können als Entität betrachtet werden [36].

- Anwendungs- und Service-Identifikator (Application and Service Identifier):
  Anwendungs- und Service-Identifikatoren identifizieren Anwendungen und Services selbst, aber auch Methoden, APIs (Application Programming Interfaces) und RPC (Remote Procedure Calls) fallen darunter [36].

- Kommunikations-Identifikator (Communication Identifier):
  Kommunikations-Identifikatoren sind beispielweise IP-Adressen (Internet-Protokoll-Adressen), Telefonnummern, HTTP Session Tokens oder Ethernet MAC (Media Access Control) Adressen. Diese Identifikatoren identifizieren einen Kommunikations-Endpunkt [36].

- Nutzer-Identifikator (User Identifier):
  Nutzer-Identifikatoren identifizieren eine nutzende Instanz einer Anwendung oder eines Service. Diese Instanz kann ein menschlicher Nutzer sein, aber auch eine andere Anwendung oder Service. Für diese Art der Nutzung wird in der Regel ein Authentifizierungsverfahren benötigt [36].

- Daten-Identifikator (Data Identifier):
  Daten-Identifikatoren identifizieren sowohl Instanzen von Daten als auch Datentypen. Beispiele hierfür sind digitale Zwillinge, Zeitreihen-Daten (Time-Series-Data) und Datentypen mit bestimmten Eigenschaften, die für einen bestimmten Nutzungskontext definiert sein müssen [36].

- Positions-Identifikator (Location Identifier):
  Ein Positions-Identifikator spezifiziert eine bestimmte geografische Position. Eine solche Position kann beispielsweise durch GPS (Global Positioning System) Koordinaten angegeben werden, aber auch Adressen, Postleitzahlen und Raumnummer sind Positionsbestimmende Identifikatoren [36].

- Protokoll-Identifikator (Protocol Identifier):
  Protokoll-Identifikatoren werden benötigt, damit Daten in der richtigen Reihenfolge und Format ausgetauscht werden können. Ein Protokoll-Identifikator gibt an, wie referenzierte Informationen interpretiert werden müssen. Ein Beispiel hierfür ist eine URI (Unified Resource Identifier), welche im IETF RFC 3968 [37] spezifiziert ist. Um die Ressource abzurufen wird ein bestimmtes Protokoll benötigt, wie z.B. HTTP (Hypertext Transfer Protocol) oder FTP (File Transfer Protocol) [36].

Neben der AIOTI hat sich auch das European Research Cluster On The Internet Of Things (IERC) das Zusammenfassen von verschiedenen Themen im IoT-Bereich zur Aufgabe gemacht [38]. So stellen sie in einer Veröffentlichung die verbreitetsten Identifikationsmechanismen in Europa und China zusammen [39]. Ähnlich wie die AIOTI werden auch hier allgemeine Kategorien (Taxonomien) von Identifikatoren eingeführt. Anders als bei AIOTI werden hier nur drei Kategorien verwendet: Objekt-Identifikator, Kommunikations-Identifikator und Anwendungs-Identifikator [39]. Diese drei Kategorien sind auch bei AIOTI wiederzufinden, wie bereits oben beschrieben wurde.

Von beiden, der AIOTI und dem IERC, werden Identifikatoren in bestimmte Kategorien eingeteilt und Standardmechanismen bzw. Protokolle zur Anwendung für die jeweilige Kategorie empfohlen, um so die Interoperabilität zwischen verschiedenen Teilnehmern zu verbessern. Dabei werden keine neuen Verfahren, Protokolle oder konkrete Lösungsansätze für bestimmte Probleme im Bereich Identifikation vorgegeben.

3.2 Hardwarebasierte Sicherheitstechnologien

Es gibt viele verschiedene Anwendungen, die Daten sicher verwalten müssen. Solche Anwendungen sind beispielsweise im Bereich Finanzen und Medizin finden, aber auch andere Daten wie Passwörter und personenbezogene Daten, die in den meisten Anwendungen vorkommen, sollten geschützt gespeichert werden [40]. Daher sollten diese Daten getrennt voneinander gespeichert werden und der Zugriff durch andere Anwendungen auf die geheimen Daten sollte verhindern werden [41]. Dies zu ermöglichen, ist ein Ziel der Trusted Execution Environments (TEEs).

Wie in Abschnitt 1.1 beschrieben, ist ein Ziel der Arbeit zu untersuchen, ob eine moderne hardwarebasierte Sicherheitstechnologie eingesetzt werden kann, um eine eindeutige Identifizierung von Softwarekomponenten zu ermöglichen. Da ein Teil der Identifizierung von per Software definierten Identifikatoren (SW-IDs) abhängig ist und diese auch unter die Kategorie

geheime Daten fallen, die vor dem Zugriff von anderen Anwendungen geschützt werden sollen, werden nachfolgend drei hardwarebasierte Sicherheitstechnologien vorgestellt, die eine TEE realisieren, bzw. im Fall von TPM nur gewisse Mechanismen, wie einen sicheren Speicher [11]. Die im nachfolgenden beschriebenen hardwarebasierten Sicherheitstechnologien wurden Aufgrund ihrer weiten Verbreitung gewählt [11].

3.2.1 Intel Software Guard Extension (Intel SGX)

Intel Software Guard Extension (Intel SGX) sind Erweiterungen der Intel Architektur durch Einführung von neuen Prozessor-Befehlen und Änderungen beim Speicherzugriff [41]. Diese Erweiterungen wurden in einigen Intel Prozessoren ab der Skylake-Serie eingeführt [42] und ermöglichen die Erstellung von geschützten Container innerhalb einer Anwendung [41]. Diese geschützten Container werden als Enklaven bezeichnet und stellen die Umsetzung einer hardwarebasierten TEE dar [43]. Enklaven sind spezielle gesicherte Bereiche im Adressraum der Anwendung, der sowohl die Vertraulichkeit (Confidentiality) als auch die Integrität (Integrity) der Daten gewährleistet, auch wenn das System bereits von Malware befallen ist [41, 44]. Zugriffe auf den Speicher der Enklave sind nur durch die Enklave selbst möglich, andere Zugriffe werden blockiert, auch wenn die Anfragen von privilegierter Software wie einem Hypervisor (Virtual Machine Monitor (VMM)), dem BIOS (Basic Input/Output System) oder dem Betriebssystem selbst kommen [41]. Die Daten der Enklave sind im Speicher für andere Prozesse nur verschlüsselt sichtbar und können nur von der Enklave selbst abgerufen und entschlüsselt werden [45]. Wird der Versuch den Inhalt einer Enklave zu ändern erkannt, wird die Ausführung der Enklave abgebrochen [41]. Die Erkennung basiert auf der Signatur der Enklave, die beim Erstellen einer Enklave generiert wird. Wird der Inhalt nachträglich geändert, ändert sich auch die kryptographische Signatur der Enklave und die Ausführung wird durch den Prozessor unterbunden [43, 45].

Der spezielle und verschlüsselte Speicherbereich der Enklaven, der durch den Intel Prozesor kontrolliert wird, wird als Enclave Page Cache (EPC) bezeichnet [44]. Die Kontrolle durch den Prozessor beinhaltet auch, dass eine Enklave nur die eigenen Daten aus dem EPC lesen kann, was über die kryptographische Signatur einer Enklave realisiert wird [45]. Die Größe des EPCs ist im BIOS fest vorgegeben und zwischen 64 bis 128 MB groß [46]. Typischerweise reicht die Größe des EPC für ca. 5-20 Enklaven gleichzeitig, was auch beim Design und der Implementierung von Anwendungen berücksichtigt werden sollte [46, 47].

Generell bestehen Anwendungen, die Intel SGX verwenden, aus mindestens zwei Teilen: einem sicheren Bereich, der Enklave, und einem unsicheren Bereich, im folgenden Enklaven Anwendung genannt, über den die Zugriffe auf die Enklave geschehen [10, 43]. Intel SGX bietet gegenüber anderen hardwarebasierten TEE Realisierungen, wie beispielsweise ARM TrustZone, den Vorteil, dass die Enklaven im Prozessor unter Ring 3 ausgeführt werden, wie normale Anwendungen ohne Intel SGX auch [11, 43, 45]. Die bereits beschriebene Trennung, die durch den EPC realisiert wird, erlaubt jedoch keine direkten Zugriffe von außen, auch nicht

durch privilegierte Prozesse, wie dem Betriebssystem. Dadurch können aus einer Enklave weder direkte Eingaben noch Ausgaben stattfinden [45]. Solche Ein- bzw. Ausgaben sind beispielsweise das Einlesen von Dateien, Eingaben von der Tastatur oder die Ausgabe von Daten auf eine Konsole bzw. den Bildschirm [45].

Zur Realisierung der Kommunikation zwischen Enklaven und der Enklaven Anwendung im unsicheren Bereich werden spezielle Aufrufe verwendet, über die Daten ausgetauscht werden können. Diese Aufrufe werden ECALLs und OCALLs genannt [46, 48]. ECALLs sind Funktionen der Enklave, die durch die Enklaven Anwendung aufgerufen werden können. OCALLs sind Funktionen die von der Enklave verwendet werden, um Ausgaben zur Enklaven Anwendung zu machen [48]. Alle ECALLs und OCALLs müssen in der Enclave Definition Language (EDL) in einer EDL-Datei definiert werden. Die Funktionen in der EDL-Datei bilden die Enklaven Schnittstellen (Interface) [46]. Dies ist schematisch in Abbildung 3.1 zu sehen. Verwendet eine Enklaven Anwendung mehrere Enklaven, gibt es für jede Enklave eine eigene EDL-Datei. Die ECALLs aus der EDL-Datei werden in der Enklave implementiert, die OCALLs in der Enklaven Anwendung. Beim Entwurf der Enklaven Schnittstellen und Aufrufe sollte berücksichtigt werden, dass jeder Aufruf einer ECALL- bzw. OCALL-Funktion einen Kontextwechsel (Context Switch) des Prozessors bedeutet (ein Wechsel zum bzw. vom EPC) und damit immer einen erhöhten Zeitaufwand mit sich bringt [45, 46].

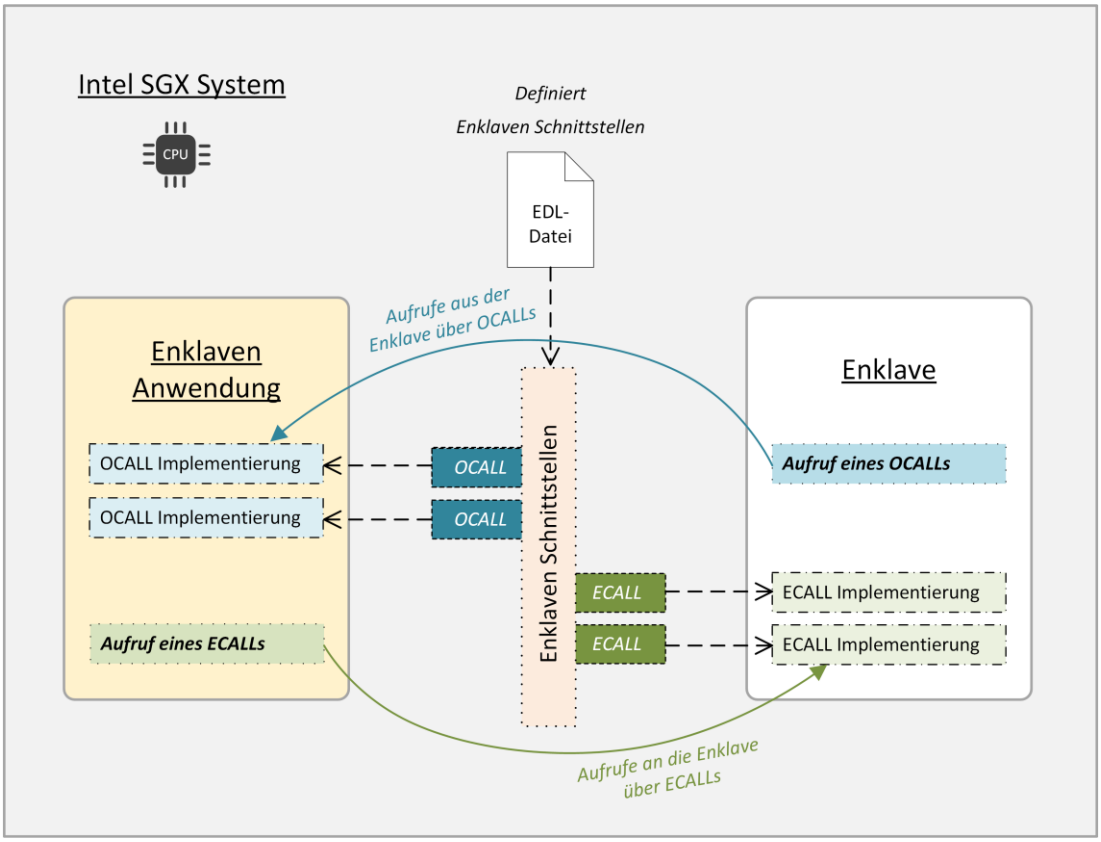

Abbildung 3.1: Definition und Aufrufe der Intel SGX Enklaven Schnittstellen

Einen weiteren wichtigen Mechanismus, den Intel SGX mitbringt, ist ein spezielles Enklaven abhängiges Verschlüsselungsverfahren, welches bei Intel SGX Sealing (dt. Versiegelung) genannt wird [48, 49]. Wird eine Enklave beendet, werden auch alle Daten, die beispielsweise über ECALLs an die Enklave übergeben wurden, gelöscht. Daher wird das Sealing benötigt, um Daten sicher aus einer Enklave dauerhaft speichern zu können [49]. Die Verschlüsselung beim Sealing findet durch den Advanced Encryption Standard (AES) mit einem 128 Bit Schlüssel statt [48]. Der kryptographische Schlüssel für das AES-Verfahren wird vom Prozessor abgerufen. Zum Abrufen des Schlüssels stehen zwei Varianten bzw. Richtlinien (Policies) zur Verfügung: die MRSIGNER und die MRENCLAVE [48, 49].

Bei der MRSIGNER Variante wird der Schlüssel für das Sealing vom Prozessor in Abhängigkeit von einem Signierungsschlüssel des Softwareanbieters angefordert [48]. Die Enklaven können mit dem erlangten Schlüssel die Daten über das Sealing speichern. Das bedeutet, dass alle Enklaven auf demselben System (der Prozessor muss derselbe sein), die denselben Signierungsschlüssel hinterlegt haben, diese Daten wieder entschlüsseln können [48, 49]. Zusätzlich zum Signierungsschlüssel des Softwareanbieters kann eine sogenannte Sicherheitsversionsnummer (SVN) zum Anfordern eines Schlüssels verwendet werden. Die SVN entspricht dabei einer aufsteigenden Versionsnummer, z.B. eines Software-Produkts. Die SVN wird dazu verwendet, dass die Daten älterer Versionen von Enklaven entschlüsselt werden können, beispielsweise nach einem Update. Ist jedoch die SVN kleiner als die, die für das Sealing der Daten verwendet wurde, funktioniert das Entschlüsseln nicht mehr [48, 49].

Die zweite MRENCLAVE Variante ist deutlich restriktiver. Bei dieser wird ein Schlüssel des Prozessors verwendet, der in Abhängigkeit der kryptographischen Signatur einer Enklave angefordert wird [48]. Dadurch können nur Enklaven auf demselben System, mit genau der gleichen Version die Daten wieder entschlüsseln. Das bedeutet, sobald sich der Quellcode einer Enklave ändert, sei es wegen eines Updates oder durch einen Manipulationsversuch, können die Daten nicht mehr über das Sealing entschlüsselt werden [48, 49]. Jedoch können auch bei dieser Variante alle Instanzen einer Enklave der gleichen Version, die somit auch dieselbe kryptographischen Signatur bilden, die Daten der anderen Instanzen entschlüsseln [48].

Ein weiterer durch Intel SGX angebotener Mechanismus zur Erhöhung des Vertrauens in andere Komponenten ist die Funktionalität der Attestierung (Attestation). Dies ist ein Validierungsmechanismus zur Überprüfung, dass der Quellcode und die Daten einer Enklave nach der Auslieferung nicht nachträglich geändert wurden [49]. Zusätzlich wird durch die Attestierung bestätigt, dass das System, auf dem die Enklave ausgeführt wird, Intel SGX unterstützt und somit eine sichere Ausführungsumgebung zur Verfügung steht [48]. Die Attestierung ist sowohl lokal (Local Attestation), als auch entfernt (Remote Attestation) möglich [49].

Die lokale Attestierung gibt zwei Enklaven die Möglichkeit, sich gegenseitig zu bestätigen, dass beide auf demselben System ausgeführt werden [48]. Dazu wird ein sogenannter Report einer Enklave erstellt. Der Report enthält die kryptographische Prüfsumme der Enklave, die vom Prozessor signiert wird. Der Report wird an die zweite Enklave übermittelt. Diese kann nun über den Prozessor prüfen lassen, ob die Signierung von demselben Prozessor stammt [48,

49]. Zur Signierung des Reports wird ein symmetrischer Schlüssel des Prozessors verwendet, wodurch auch nur derselbe Prozessor den Report entschlüsseln und bestätigen kann [48].

Bei der entfernten Attestierung soll bestätigt werden, dass eine Enklave auf einem anderen Intel SGX System mit unveränderten Quellcode ausgeführt wird [48]. Dazu wird ähnlich, wie bei der bereits beschriebenen lokalen Attestierung, eine kryptographische Prüfsumme der Enklave gebildet. Diese wird anders als bei der lokalen Attestierung mit einem privaten Schlüssel des Intel Prozessors signiert. Das Ergebnis wird als Quote bezeichnet [48]. Die Quote wird an die Komponente gesendet, die den Beweis angefordert hat. Diese kann nun die Quote über einen von Intel angebotenen Service überprüfen lassen und so feststellen, ob die Signierung mit einem Schlüssel von einem Intel Prozessor durchgeführt wurde. Dies ist möglich, da Intel die öffentlichen Schlüssel der Prozessoren kennt [48]. Ist die Signierung gültig, kann die Komponente, die den Beweis angefordert hat, die kryptographische Prüfsumme der Enklave, die im Quote enthalten ist, überprüfen. Ist die Prüfsumme identisch mit der Enklaven-Version die erwartet wird, ist die Attestierung abgeschlossen und somit sichergestellt, dass die erwartete Version der Enklave in einer sicheren Ausführungsumgebung läuft [48, 49].

Intel SGX wird bereits in einigen Bereichen eingesetzt, wie beispielsweise zur Absicherung von Docker Containern [47]. Aber auch im Bereich der Cloud, welche aus Sicht des Softwareanbieters als unsichere Ausführungsumgebung angesehen wird, wird Intel SGX verwendet, um dort eine sichere und vertrauenswürdige Ausführungsumgebung bereitstellen zu können [42]. Fremantle, Aziz & Kirkham verwenden Intel SGX, um ein sicheres API (Application Programming Interface) für IoT-Geräte bereitzustellen, die anschließend durch Verwendung der Blockchain-Technologie verwaltet werden sollen [34].

Intel SGX bietet durch die Verschlüsselung des Speichers und durch Verwendung von geheimen im Prozessor hinterlegter Schlüsseln, Schutz vor einigen Hardware- und Software-Attacken [46]. Reverse Engineering (Rekonstruktion von Quellcode) und Seitenkanal-Attacken (Side-Channel Attacks) sind jedoch dennoch möglich und müssen durch Berücksichtigung beim Entwurf und Umsetzung der Software vermieden werden [48]. Des Weiteren wurde 2018 eine Sicherheitslücke mit dem Namen „Foreshadow“ in Intel Prozessoren entdeckt [50]. Die Sicherheitslücke betrifft die virtuelle Speicherverwaltung des Arbeitsspeichers. Durch Ausführung von bestimmten Befehlen, in nicht vorhergesehener Reihenfolge (Out-of-Order), konnten bestimmte Schlüssel aus dem Prozessor ausgelesen werden, die von Intel SGX zur Signierung und Attestierung verwendet werden [50].

3.2.2 ARM TrustZone

ARM TrustZone ist eine Erweiterung einiger Systeme mit ARM-Architektur, die 2004 eingeführt wurde [11]. Die Erweiterungen bestehen aus einigen Hardwaremodulen, die zur Trennung eines Systems in eine *sichere Welt* und eine *normale Welt* verwendet werden [6, 45]. Die sichere Welt ist bei ARM TrustZone die Umsetzung der Trusted Execution Environment (TEE) [11]. Zur Realisierung der beiden Welten unterscheidet ein Prozessor mit ARM TrustZone Erweiterung die Befehle aus der sicheren Welt und der normalen Welt [45]. Hierfür wird

vom Prozessor ein zeitversetztes (Time-Sliced) Verfahren (Scheduling) verwendet, wodurch beide Welten den Eindruck haben, einen eigenen Prozessor zu verwenden [6, 51].

Aus softwareseitiger Sicht kann die Trennung der beiden Welten als zwei verschiedene Betriebssysteme betrachtet werden [6]. Bei der Verwendung von allgemeinen Betriebssystemen, wie Linux oder Android, läuft das allgemeine Betriebssystem in der normalen Welt. In der sicheren Welt wird ein spezielles Betriebssystem, wie beispielsweise ein angepasstes Linux Betriebssystem, ausgeführt [6]. Zur Absicherung der Betriebssysteme beim Starten, wird ein spezieller Bootloader (Startprogramm) verwendet, dass durch verschiedene Schritte überprüft, dass das sichere Betriebssystem nicht durch ein infiziertes oder geändertes ausgetauscht wurde. Zur Prüfung werden unter anderem auch öffentliche Schlüssel des Betriebssystem-Anbieters verwendet [6]. Nach den Prüfschritten ist zunächst nur die sichere Welt initialisiert und erst danach werden die Teile der normalen Welt geladen, um so Konflikte zu vermeiden [45]. Sind beide Betriebssysteme hochgefahren, können diese über einen speziellen Monitor kommunizieren, welcher auch Teil der sicheren Welt ist [45]. Diese Trennung ist in Abbildung 3.2 dargestellt.

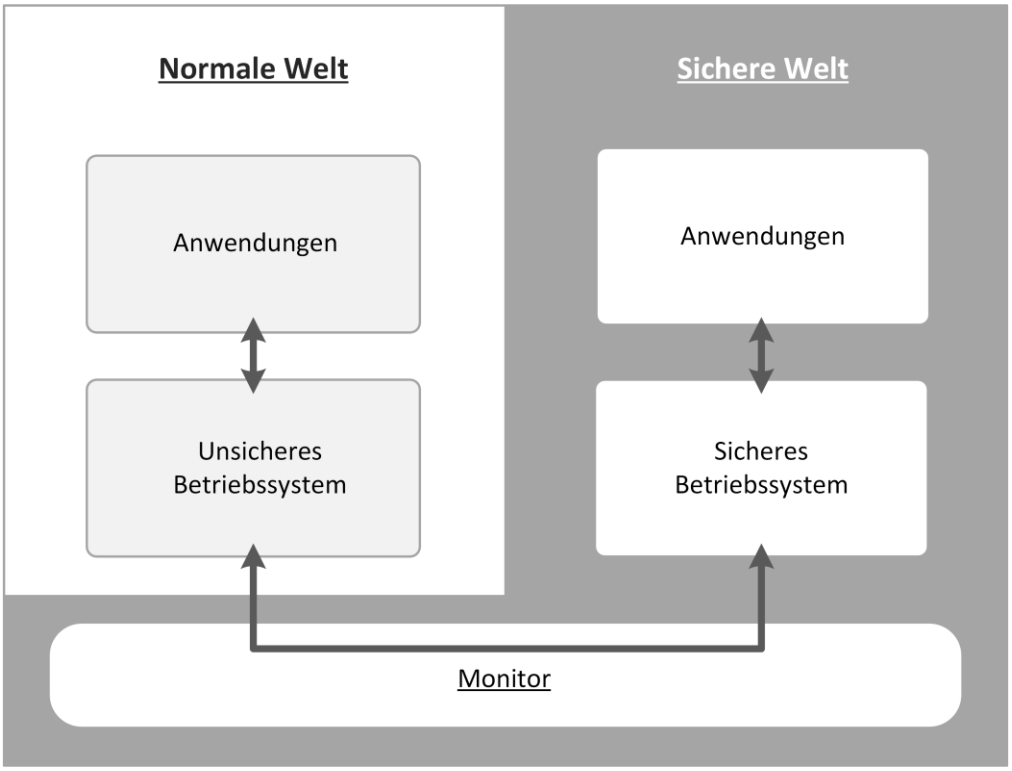

Abbildung 3.2: ARM TrustZone - Unterteilung in normale und sichere Welt (Abbildung in Anlehnung nach [6])

Aus der sicheren Welt kann auf alle Speicherbereiche zugegriffen werden, auch auf die der normalen Welt. Aus der normalen Welt ist nur der Speicherbereich der normalen Welt

zugreifbar und zusätzlich ein definierten Speicherbereich der sicheren Welt [6, 45]. Der definierte Speicherbereich der sicheren Welt dient zum Datenaustausch von der normalen Welt in die sichere Welt [45]. Die Zugriffe werden dabei immer vom Monitor kontrolliert. Dieser implementiert hierfür auch die Kontextwechsel (Context Switch) des Prozessors, die für die sichere Kommunikation der beiden Welten notwendig sind [6, 45]. Für beide Welten gibt es eine eigene Seitentabelle, welche die Speicherbereich verwaltet [45].

Ein Beispiel für den Einsatz der ARM TrustZone ist das von Samsung entwickelte KNOX [11]. Dies ist ein von Samsung entwickeltes System, das ARM TrustZone verwendet, um private von geschäftlichen Daten auf einem Gerät (Smartphone) zu trennen. In der normalen Welt werden ganz normal Apps ausgeführt und Daten gespeichert. Wird über eine spezielle Anwendung in den sicheren Bereich gewechselt, ist eine gesonderte Authentifizierung notwendig, die den Zugriff auf die geschäftlichen Daten ermöglicht [11].

Ein Nachteil von ARM TrustZone ist, dass es keine Mechanismen zur Überprüfung von außen (Attestierung) bereitstellt, dass die in der sicheren Welt ausgeführte Software auch die ist, welche erwartet wird [45]. Eine weitere Einschränkung, die es bei der Auswahl Technologie zu beachten gilt, ist, dass nur eine TEE (sichere Welt) ausgeführt werden kann und diese Zugriff auf alles hat [11]. Nach Pinto und Santos existiert zudem eine Vielzahl von bekannten Schwachstellen, die sowohl die Softwareseitige Umsetzung einer TEE, als auch Teile der Hardware betreffen, wie beispielsweise den Zwischenspeicher (Cache) oder die Energieverwaltung (Power Management) [11]. Diese Schwachstellen müssen beim Entwickeln eines Systems mit ARM TrustZone berücksichtigt werden [11].

3.2.3 Trusted Plattform Modul (TPM)

Das Trusted Platform Module (TPM) ist ein fälschungssicherer Chip, der typischerweise von einem Standard Computer (Personal Computer (PC)) verwendet wird [45, 52]. Anders als die beiden zuvor vorgestellten hardwarebasierten Sicherheitstechnologien Intel SGX und ARM TrustZone, sind TPMs kein Teil des Prozessors, sondern ein eigenständiges Hardware-Modul (Chip) [45]. TPM wird von der Trusted Computing Group (TCG), aktuell in der Version 2.0, spezifiziert [52, 53].

Die Aufgabe eines TPMs ist das sichere Speichern von kryptographischen Schlüsseln, z.B. zur Attestierung von Software, und ist mittlerweile im Segment der Standard Computer weit verbreitet [45]. Ein TPM hat zur Erfüllung seiner Funktion mehrere Schlüssel, wie den Endorsement-Schlüssel (Bestätigungs-Schlüssel), den Storage-Root-Schlüssel (Speicher-Wurzel-Schlüssel) und die Attestation-Identity-Schlüssel (Attestierungs-Identitäts-Schlüssel) [53].

Der Endorsement-Schlüssel ist ein eindeutiger Schlüssel und wird für RSA-Verfahren[1] verwendet. Über den Endorsement-Schlüssel kann ein TPM zudem eindeutig identifiziert werden [53]. Der private Teil des Endorsement-Schlüssels darf nie das TPM verlassen, weswegen auch eine Sicherung (Backup) des Schlüssels nicht möglich ist. Der Endorsement-Schlüssel kann jedoch gelöscht und ein neuer kann erstellt werden. Durch diese Schritte werden jedoch auch alle zuvor abgeleiteten Schlüssel und Signierungen ungültig [53]. Der zweite Schlüssel, der Storage-Root-Schlüssel, wird dafür verwendet, um alle im TPM gespeicherten Schlüssel zu verschlüsseln, damit diese sicher abgelegt werden können. Die Daten bzw. Schlüssel können danach nur von demselben TPM wieder entschlüsselt werden [53]. Die Attestation-Identity-Schlüssel sind ebenfalls Schlüssel für RSA-Verfahren, mit einer festen Länge von 2048 Bit. Diese Schlüssel werden zur Erstellung von Signaturen von Hardware- und Software-Konfigurationen erstellt. Durch die Signaturen können die Konfigurationen bzw. Zustände überprüft und somit Attestiert werden [53]. Damit sich auch externe Parteien von dem Zustand bzw. Version einer Software überzeugen können, können die Signaturen abgerufen werden und die entfernte Partei kann sich davon überzeugen, dass die Software, bestimmte Firmware-Versionen oder auch Hardware-Konfigurationen auf einem entfernten System dem entspricht, was erwartet bzw. gewünscht ist (Remote Attestation) [53]. Damit nicht anhand der Signaturen ein TPM und damit ein PC verfolgt werden kann, werden mehrere Attestation-Schlüssel erstellt, die alle durch Ableiten von einem Wurzel-Schlüssel, durch sogenannte Key-Derivation-Funktionen, erlangt werden [53].

Zusätzlich zu den Schlüsseln, die in einem eigenen nicht flüchtigen Speicher des TPMs abgelegt werden, bietet ein TPM einen Zufallszahlen-Generator zur Erstellung von Nonce (**n**umber used **once**) Werten, eine spezielle RSA-Engine zur Durchführung RSA-basierter Ver- und Entschlüsselungen und spezielle Hash-Algorithmen, die für Signierungen und Integritätsprüfungen verwendet werden [54].

Eines der weitverbreitetsten Programme, das TPM verwendet, ist das Programm BitLocker von Microsoft, das die TPM-Funktionen zum Verschlüsseln von Festplatten verwendet [53]. Des Weiteren wird TPM von E-Mail-Programmen, wie Thunderbird und Outlook für die Verschlüsselung von Nachrichten verwendet, von Web Browsern, wie beispielsweise Internet Explorer, Firefox oder Chrome, und auch Virtual Private Network (VPN) Programme, wie der Cisco VPN Client, verwenden TPM [53].

Angriffe auf TPMs sind möglich, wenn physischer Zugriff auf ein System besteht. Beispielsweise werden die Prüfsumme für die Signaturen vom Prozessor berechnet, da ein TPM keine eigene isolierte Ausführungsumgebung hat. Dadurch kann die Kommunikation zwischen Prozessor und TPM gestört bzw. manipuliert werden [45]. Zudem sind einige Angriffe bekannt,

[1] Das RSA-Verfahren, benannt nach seinen Erfindern Ron Rivest, Adi Shamir und Len Adleman, ist ein asymmetrisches Verschlüsselungsverfahren [15, 53].

die durch falsche Verwendung oder durch unzureichende Beschreibungen oder logischen Widersprüchen aus der Spezifikation hervorgehen [45, 55]. Auch sind Fehler bekannt, die auf den Hersteller von TPMs zurückzuführen sind [55].

3.3 Ausführungsumgebungen für TEEs

In diesem Abschnitt werden zwei Arbeiten vorgestellt, welche die Bereitstellung einer Trusted Execution Environments (TEEs) als Ziel haben. Einmal durch Verwendung einer bereits etablierten hardwarebasierten TEE-Technologie und einmal durch eine rein softwarebasierte Lösung.

Die Arbeit von Arnautov et al. stellt eine Container-basierte Ausführungsumgebung mit Docker und Intel SGX vor [47]. Hierfür werden die Vorteile von Containern und Intel SGX kombiniert und eine sichere Ausführungsumgebung, eine TEE, ermöglicht. Nach Arnautov et al. haben Containern gegenüber virtuellen Maschinen (VMs) den Vorteil, dass weniger System Ressourcen benötigt werden [47]. Für VMs kann Intel SGX bereits verwendet werden, für Container noch nicht. SCONE (Secure CONtainer Environment) ermöglicht die Ausführung von Linux Anwendungen in Docker Containern. Die Container werden dann als Enklave mit Intel SGX ausgeführt werden [47]. An der Docker Engine selbst oder den Programmierschnittstellen (Application Programming Interface (API)) von Docker wurde zur Realisierung nichts geändert, jedoch wurde eine Art Wrapper (Hülle) um Docker erstellt. Diese wird dazu verwendet um Docker wie gewöhnlich verwenden zu können, jedoch werden zusätzlich die Bibliotheken von SCONE mit eingebunden. Die Bibliotheken ermöglichen beispielsweise, dass eine Anwendung aus einem sicheren Container mit SCONE auch Ein- und Ausgaben machen können, was aus Enklaven normalerweise nicht möglich ist [47]. Eine Einschränkung von SCONE ist, dass es nur auf einem System mit Intel SGX ausgeführt werden kann und nur Linux als Betriebssystem zu Verfügung steht.

McGillion, Dettenborn & Nyman stellen unter dem Namen Open-TEE eine softwarebasierte Lösung für eine TEE vor [5]. Ziel der virtualisierten TEE ist die Bereitstellung einer Ausführungsumgebung zum Testen und Debuggen von Software, die Funktionen einer TEE verwenden soll. Durch Einhaltung der Spezifikationen der GlobalPlatform [9] für TEEs, soll es mit Open-TEE möglich sein, eine Anwendung auf Systemen ohne eine hardwarebasierte TEE zu entwickeln und zu testen und anschließend ohne Änderungen auf einem System mit hardwarebasierter TEE auszuführen [5]. Open-TEE eignet sich jedoch nur zur Bereitstellung einer Ausführungsumgebung zu Test- und Entwicklungszwecken, da ohne die hardwarebasierten Schutzmechanismen einer TEE, wie beispielsweise der isolierten Ausführungsumgebung, nicht mehr Sicherheit gewährleistet werden kann.

4 Konzept

Wie in Abschnitt 1.1 beschrieben, ist das Ziel dieser Arbeit eine Möglichkeit zur eindeutigen Identifizierung von Softwarekomponenten auf verschiedenen Geräten zu finden. Ein Aspekt der dabei betrachtet wird, ist die Überprüfung, ob eine hardwarebasierte Sicherheitstechnologie zu diesem Zweck eingesetzt werden kann. In diesem Kapitel werden zunächst das Systemmodell und die Problembeschreibung vorgestellt. Anschließend werden die Anforderungen definiert und die Auswahl der Sicherheitstechnologie wird getroffen. Danach wird das Konzept zur Lösung der Problemstellung beschrieben.

4.1 Systemmodell und Problembeschreibung

Gesucht wird eine Lösung, die eine eindeutige Identifizierung von Softwarekomponenten ermöglicht. Eine Softwarekomponente kann ein eigenständiges Softwareprodukt oder auch nur ein Teil eines verteilten Softwaresystems sein, wie z.B. ein einzelner Service eines Systems mit Microservice-Architektur. Dieser Mechanismus zur eindeutigen Identifizierung von Softwarekomponenten, kurz MISK, soll im Industrie 4.0 (Industrial IoT (IIoT)) Umfeld eingesetzt werden.

Eine Voraussetzung für MISK ist, dass jede Instanz einer Softwarekomponente über einen per Software definierten Identifikator (SW-ID) verfügen muss. Hierzu wird die Annahme getroffen, dass jede SW-ID eindeutig ist und jede SW-ID nur das Ausführen einer Instanz einer Softwarekomponente legitimiert. Als Instanz wird eine laufende Version einer Softwarekomponente bezeichnet. Ziel von MISK ist es nun, dass zwei Instanzen einer Softwarekomponente mit derselben SW-ID zuverlässig als zwei verschiedene Instanzen dieser Softwarekomponente erkannt werden können, egal ob diese auf demselben physischen Gerät, auf unterschiedlichen physischen Geräten oder in virtualisierten Umgebungen ausgeführt werden.

Zur Verdeutlichung wird im Folgenden ein kurzes Szenario beschrieben. Das Szenario beinhaltet verschiedene Rollen. Eine bereits erwähnte Rolle ist die Softwarekomponente, welche durch ihr Verhalten auch als Client betrachtet werden kann. Die zweite Rolle ist der Kontrollserver. Zusätzlich wird ein Kommunikationskanal zwischen der Softwarekomponente und dem Kontrollserver benötigt, wie z.B. eine Netzwerkverbindung. Über diesen wird angenommen, dass es sich um einen sicheren und zuverlässigen Kommunikationskanal handelt und somit auch keine Nachrichten verloren gehen. Dies könnte z.B. durch Verwendung von TCP (Transmission Control Protocol) mit TLS (Transport Layer Security) zur Übertragung erreicht

werden. Des Weiteren wird über den Kontrollserver die Annahme getroffen, dass dieser nicht manipuliert werden kann und die Authentizität, z.B. durch ein Public-Key-Verfahren, sichergestellt ist. Das Szenario beinhaltet folgende Interaktion: Die Instanz einer Softwarekomponente identifiziert sich über den Kommunikationskanal beim Kontrollserver mittels der SW-ID. Damit der Kontrollserver sicherstellen kann, dass die Softwarekomponente die ist, der die SW-ID zugeordnet wurde, wird ein Beweis bzw. Kontrollmechanismus benötigt.

Im Allgemeinen wird der Vorgang zur Prüfung auf Authentizität (Echtheit) eines Teilnehmers bzw. Datenquelle Authentifizierung genannt [13]. Einer der verbreitetsten Authentifizierungsmechanismen ist das Passwort-Verfahren [56]. Dabei wird zusätzlich zum Identifikator ein Passwort mitgegeben. Über die Kombination von Identifikator und Passwort kann der Kontrollserver prüfen, ob das Passwort zum Identifikator passt. Dieses Verfahren setzt voraus, dass ein Geheimnis (hier Passwort) existiert, welches nur der Client und der Kontrollserver kennt. Der Identifikator, wie beispielsweise eine E-Mail-Adresse, kann hingegen öffentlich bekannt sein. Anders als beim beschriebenen Passwort-Verfahren sollte die SW-ID, welche in diesem Szenario der Identifikator ist, nicht öffentlich bekannt sein, da eine SW-ID das Ausführen genau einer Instanz der Softwarekomponente legitimiert. Aus diesem Grund muss betrachtet werden, welche Möglichkeiten zur Umgehung der legitimierten Ausführung einer Softwarekomponente bestehen. Generell kann ein Angreifer[1] versuchen mit einer einzelnen SW-ID mehrere Instanzen einer Softwarekomponente parallel auszuführen. Dazu sind die folgenden Möglichkeiten in Betracht zu ziehen:

Möglichkeit 1

Der Angreifer hat eine zur Ausführung notwendige SW-ID im Klartext, die bereits durch eine andere laufende Instanz der Softwarekomponente verwendet wird. Diese wird nun zur Ausführung einer weiteren Instanz verwendet.

Möglichkeit 2

Der Angreifer hat eine Kopie einer bereits instanziierten Softwarekomponente mit hinterlegter SW-ID. Eine solche Kopie2 kann beispielsweise das Abbild (Image) einer virtuellen Maschine sein.

Beide Möglichkeiten sollen durch das zu entwerfende Verfahren unterbunden werden.

[1] Zur besseren Verständlichkeit wird hier der Begriff des Angreifers eingeführt. Mit Angreifer sind alle Personen gemeint, welche versuchen die Bedingungen zur rechtmäßigen Inbetriebnahme einer Softwarekomponente zu umgehen.

[2] Abgrenzung: Die Möglichkeit zur Erstellung eines physischen Klons der Hardware wird hier nicht näher betrachtet.

4.2 Anforderungen

Die Aufgabe von MISK ist es, die gleichzeitige Nutzung einer SW-ID zu erkennen und in Verbindung mit dem Kontrollserver, entsprechend zu reagieren. Eine beispielhafte Integration von MISK ist in Abbildung 4.1 zu sehen. Aus dem oben beschriebenen Szenario lassen sich nun folgende Anforderungen[3] ableiten:

Anforderung 1

Gleichzeitig ausgeführte Instanzen von Softwarekomponenten mit derselben SW-ID sollen während des Betriebs erkannt werden.

Anforderung 2

MISK soll in verschiedene Softwareprodukte integrierbar sein, die bereits über eine SW-ID, wie z.B. einem Lizenzschlüssel, verfügen.

Anforderung 3

Das Format der SW-ID für MISK soll variabel sein und nicht von einem strikten Schema, wie z.B. einem 25 Zeichen langem Lizenzschlüssel, abhängig sein.

Anforderung 4

MISK soll auch in virtualisierten Umgebungen funktionieren.

Anforderung 5

Der Anwender einer Softwarekomponente mit MISK soll keine spezielle Hardware benötigen, wie beispielsweise einem USB-Dongle (Universal-Serial-Bus-Dongle). Handelsübliche Hardware (Commodity Hardware) im Bereich Workstation und Server, sollen für den Betrieb ausreichend sein.

Anforderung 6

Die Funktion von MISK soll nicht abhängig von einer speziellen Sicherheitstechnologie sein.

a) Unterstützt das Gerät eine bestimmte Sicherheitstechnologie, kann diese von MISK eingesetzt werden.

b) Unterstützt das Gerät diese Sicherheitstechnologie nicht, bzw. ist diese deaktiviert, soll MISK dennoch funktionieren.

Anforderung 7

Als Betriebssysteme sollen sowohl Windows- als auch Linux-Betriebssysteme unterstützt werden.

[3] Auf eine Unterteilung in funktionale- und nichtfunktionale-Anforderungen wurde hier verzichtet, da nach Glinz eine klare Unterscheidung oft nicht möglich ist und je nach Formulierung eine Anforderung als funktionale- oder nichtfunktionale-Anforderung betrachtet werden kann [57].

Anforderung 8

MISK soll auch ohne Verbindung zum Kontrollserver funktionieren, jedoch unter bestimmten Einschränkungen (z.B. nur für einen gewissen Zeitraum oder mit eingeschränkter Funktionalität).

Anforderung 9

Die Verarbeitungsgeschwindigkeit (Performanz) der Softwarekomponente soll durch MISK nicht signifikant verschlechtert werden.

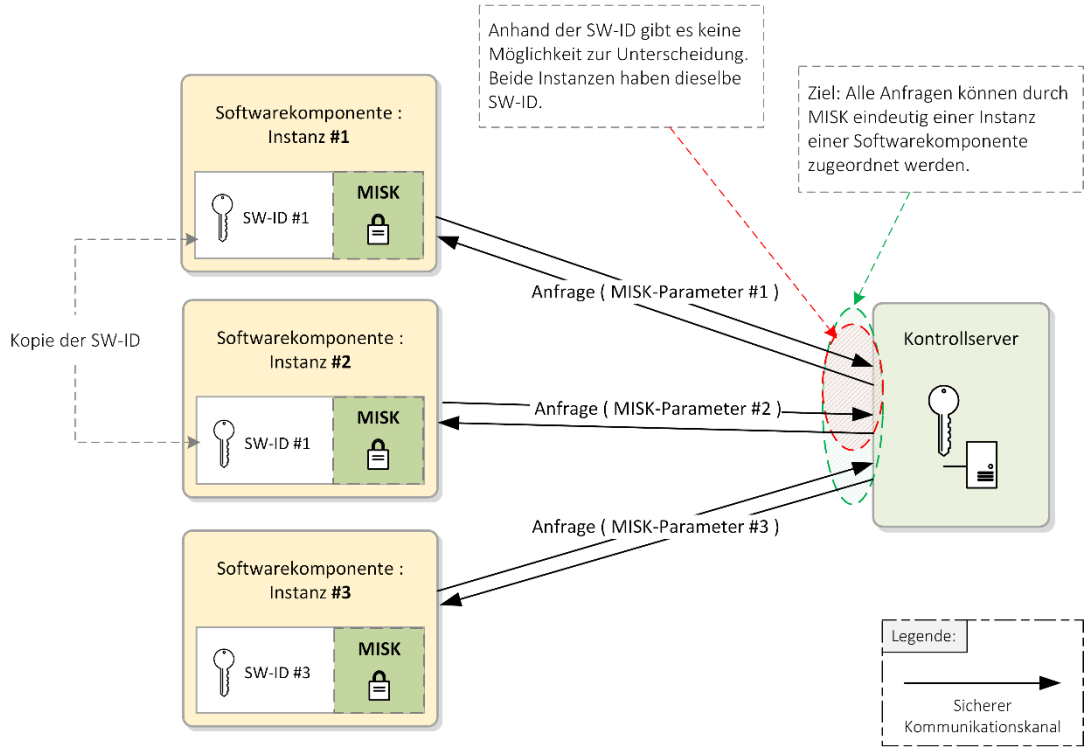

Abbildung 4.1: Identifizierung der Softwarekomponenten-Instanzen durch MISK

4.3 Auswahl der Sicherheitstechnologie

Im Abschnitt 3.2 wurden bereits die hardwarebasierten Sicherheitstechnologien vorgestellt. Wie in **Anforderung 6** formuliert, soll für das Konzept überprüft werden, ob es eine moderne hardwarebasierte Sicherheitstechnologien gibt, die sich zum Einsatz für MISK eignet. Eine der wichtigsten Anforderungen dafür, die sich aus **Anforderung 5** ergibt, dass handelsübliche Hardware im Bereich Workstation und Server ausreichend sein soll, ist, dass die Sicherheitstechnologie eine weite Verbreitung in dieser Kategorie von Hardware bzw. Geräten hat. Nach Pinto und Santos sind dies die Sicherheitstechnologien ARM TrustZone, Intel SGX und TPM [11]. Als weitere wichtige Kriterien wurden die Verfügbarkeit einer isolierten Ausführungsumgebung von Code, einer speziellen hardwareabhängigen Verschlüsselung, im folgenden

Datenversiegelung (Sealing) genannt und die Möglichkeit zur entfernten Attestierung (Remote Attestation) von Daten bzw. Quellcode definiert. Des Weiteren wurde überprüft, ob sich die Funktion der Sicherheitstechnologie an einem Gerät abschalten lässt. Das Abschalten der Sicherheitstechnologie könnte gezielt dazu verwendet werden, um die Sicherheitsmechanismen, die MISK verwenden soll, zu umgehen. Die Ergebnisse sind in der Tabelle 4.1 zusammengefasst.

	Isolierte Ausführung	Attestierung	Daten-versiegelung	Nicht abschaltbar
ARM TrustZone	Ja	Nein	Nein	Ja
Intel SGX	Ja	Ja	Ja	Nein
TPM	Nein	Ja	Ja	Nein

Tabelle 4.1: Vergleich der Sicherheitstechnologien

Eine weitere wichtige Kategorie von hardwarebasierten Sicherheitsmechanismen sind sogenannte physikalisch nicht klonbare Funktionen (Physical Unclonable Functions (PUFs)), die im Abschnitt 2.3 bereits vorgestellt wurden. PUFs können zwar eine sehr hohe Fälschungssicherheit ihrer eindeutigen Merkmale vorweisen [18], haben aber den Nachteil, dass immer ein spezieller Hardwarebaustein vorhanden sein muss, der anders als die weitverbreiteten TPMs [53], nicht auf handelsüblichen Geräten verfügbar ist.

Aufgrund dessen, dass Intel SGX bis auf eine Anforderung alle erfüllt und durch die weite Verbreitung von Intel Prozessoren [58], wird für MISK Intel SGX als Sicherheitstechnologie gewählt. Ein weiterer Vorteil von Intel SGX ist, dass es im Prozessor unter Ring 3 ausgeführt wird [11], welche für Anwendungsprogramme vorgesehen ist und daher leicht von anderen Anwendungsprogrammen verwendet werden kann.

Die ARM TrustZone bietet zwar auch eine isolierte Ausführung, jedoch keine Möglichkeit zur entfernten Attestierung von Code und keinen Mechanismus zur Datenversiegelung [11]. Des Weiteren ermöglicht ARM TrustZone nur die Ausführung eines einzelnen sicheren Containers in der sicheren Welt [11], wohingegen bei Intel SGX mehrere Enklaven parallel ausgeführt werden können. Zudem ist die eine Umsetzung nicht durch reine Software auf Anwendungsebene möglich [11].

TPMs können durch die sicher verwalteten Schlüssel [53] gut zur Identifizierung eines Geräts, in welches das TPM verbaut ist, verwendet werden. Dies bringt jedoch gegenüber einer Geräte-Seriennummer für den Verwendungszweck im Konzept keinen großen Mehrwert, da die sicher gespeicherten Schlüssel im TPM von jeder Instanz abgerufen werden können, die Zugriff auf das Gerät haben. Des Weiteren wird die isolierte Ausführung als eines der wichtigsten Kriterien gesehen, welches TPMs nicht erfüllen.

4.4 Struktur des Identifikators

Wie im vorigen Abschnitt beschrieben, wurde Intel SGX als Sicherheitstechnologie für MISK gewählt. Da diese spezielle hardwareabhängige Sicherheitstechnologie nicht auf jedem Gerät verfügbar ist, MISK jedoch, wie in **Anforderung 6** beschrieben, nicht nur basierend auf der gewählten Sicherheitstechnologie funktionieren soll, wird ein spezielles Prüfverfahren eingeführt. Das Prüfverfahren testet das System auf dem Gerät auf verschiedene Eigenschaften, wie z.B. ob Intel SGX verfügbar ist oder, ob es sich um eine virtualisierte Umgebung handelt. Das Ergebnis der Prüfung ist die Entscheidung, wie der Identifikator für die jeweilige Softwarekomponenten-Instanz zu erstellen ist. Das Prüfverfahren muss für die gleichen Ausgangsbedingungen (Hardwarekomponenten, Betriebssystem, Systemeinstellungen, Laufzeitumgebung, usw.), das gleiche Ergebnis liefern und somit deterministisch sein. Die einzelnen Stufen des Prüfverfahrens sind in Abbildung 4.2 dargestellt. Insgesamt sind vier verschiedene Stufen zur Erstellung des Identifikators im Konzept vorgesehen. Die Nummerierung der Stufen geben die Prioritäten, mit welcher diese gewählt werden sollen, in absteigender Form an: 1 hat höchste Priorität und 4 die niedrigste. Diese Stufen – im Folgenden wird jede Stufe als Variante bezeichnet – sind wie folgt benannt:

Variante 1 (Sicherheitstechnologie)

Variante 2 (Hersteller Geräte-Identifikator)

Variante 3 (Generierter Geräte-Identifikator)

Variante 4 (Virtualisierte Umgebung)

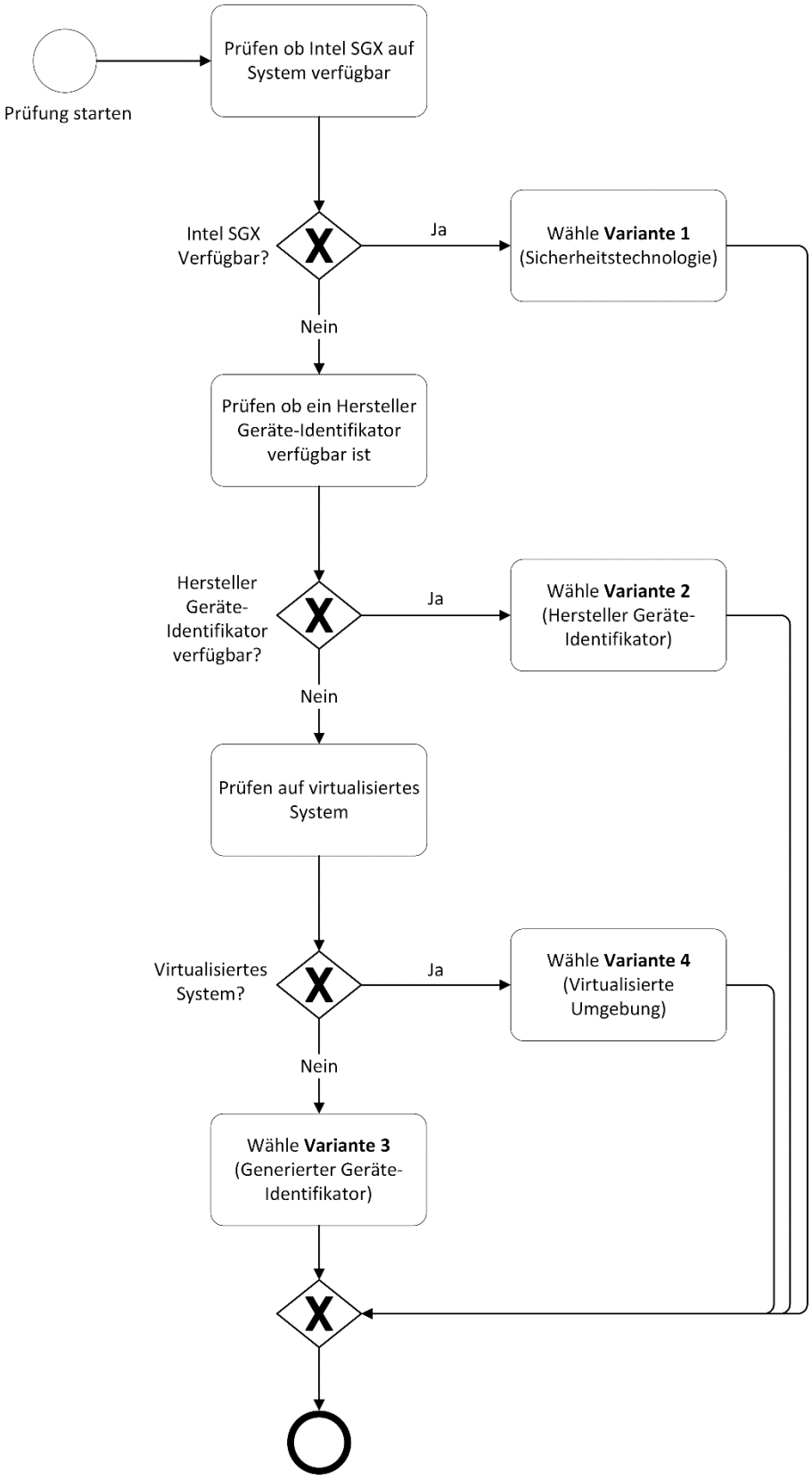

Abbildung 4.2: Schritte des Prüfverfahrens mit Variantenauswahl

Die Bedingungen für die Prüfung und zur Erstellung des eindeutigen Identifikators werden im folgenden Abschnitt 4.5 genauer erläutert. Das Ergebnis jeder Variante ist ein Identifikator im einheitlichen Format, welcher im Folgenden als MISK-Identifikator bezeichnet wird. Der MISK-Identifikator wird aus mehreren Parametern zusammengesetzt, wie beispielsweise des Geräte-Identifikators (Geräte-ID). Die Identifizierung eines Geräts anhand einer Geräte-ID, wie es bei den Varianten 2 und 3 im Vordergrund steht, sind jedoch nur eine Eingrenzung, da hierdurch noch nicht mehrere Softwarekomponenten-Instanzen, die mit der gleichen SW-ID auf demselben Gerät laufen, unterschieden werden können. Die Struktur des MISK-Identifikators ist dabei wie folgt definiert:

```
MISK-Identifikator:
 1) SW-ID
 2) Variante
 3) Geräte-ID
 4) Softwarekomponenten-Nonce
 5) Kontrollserver-Nonce
```

Die einzelnen Parameter des MISK-Identifikators und ihre Funktionen werden im Folgenden genauer erläutert.

4.4.1 SW-ID

Der erste Parameter des MISK-Identifikators ist der bereits erwähnte per Software definierte Identifikator (SW-ID), z.B. ein Lizenzschlüssel. Die Generierung und Bereitstellung einer gültigen SW-ID ist nicht Teil des Konzeptes, sondern Teil eines externen Lizenzmodells, welches MISK als Lösung zur Einhaltung bzw. Überprüfung der Lizenzvorschriften verwenden kann.

4.4.2 Variante

Der zweite Parameter gibt an, durch welche Variante der MISK-Identifikator erstellt wurde. Hierdurch kann unterschieden werden, welche Parameter verfügbar sind (für Variante 4 ist beispielsweise keine Geräte-ID vorhanden (siehe Abschnitt 4.5.4)) und es kann ein Rückschluss auf das Vertrauenslevel des MISK-Identifikators gezogen werden. Die Nummerierung der Varianten gibt das Vertrauenslevel in absteigender Form an: Variante 1 (Sicherheitstechnologie) hat aufgrund der höheren Fälschungssicherheit, durch die verwendeten Sicherheitsmechanismen von Intel SGX, das höchste Vertrauenslevel und Variante 4 (Virtualisierte Umgebung) das niedrigste, da hier der MISK-Identifikator nicht zusätzlich wie bei den anderen Varianten an ein Gerät gebunden werden kann.

4.4.3 Geräte-ID

Der Geräte-Identifikator (Geräte-ID) dient zur eindeutigen Identifizierung eines Geräts. Die jeweilige Erstellung und Verwendung für die einzelnen Varianten werden in den folgenden Abschnitten 4.5.1 bis 4.5.4 erläutert.

4.4.4 Softwarekomponenten-Nonce

Die Softwarekomponenten-Nonce (SWKP-Nonce) wird bei jedem Start einer Softwarekomponenten-Instanz neu generiert. Eine Nonce (**n**umber used **once**) bezeichnet dabei einen Wert, der im verwendeten Kontext bzw. einer Sitzung (Session) nur einmal verwendet werden darf [59]. Die Nonce muss dabei nicht zwingend eine Nummer sein, sondern kann auch eine Zeichenkette oder ein zufälliger, aber eindeutiger, Hash-Wert sein. Das allgemeine Ziel bei der Verwendung von Nonces ist die Vermeidung von Replay-Attacken[4] [45, 62]. Für die SWKP-Nonce wird die Annahme getroffen, dass diese im Kontext der Verwendung eindeutig ist und der nächste Wert nicht vorhersehbar ist. Dies kann beispielsweise durch Verwendung von Zufallswerten in Kombination mit einem Zeitstempel (Timestamp) erreicht werden. Der Zeitstemple sorgt dafür, dass es immer eine konstant aufsteigende Zahl gibt und der Zufallswert verhindert die Vorhersehbarkeit. Die Generierung der SWKP-Nonce findet auf Seite der Softwarekomponente statt.

Ziel der SWKP-Nonce ist es, zwei parallellaufende Softwarekomponenten-Instanzen auf demselben Gerät eindeutig voneinander unterscheiden zu können. Dies ist möglich, da zwei Softwarekomponenten-Instanzen immer unterschiedliche Nonce-Werte haben müssen, wie aus der zuvor genannten Annahme der Eindeutigkeit der Nonce-Werte hervorgeht. Der Grund dafür, dass die SWKP-Nonce nur flüchtig ist und nach jedem Start einer Softwarekomponenten-Instanz neu generiert wird, ist, dass so bei der Erstellung einer Kopie, wie in Abschnitt 4.1 bei **Möglichkeit 2** beschrieben, die Nonce nicht in der Kopie enthalten ist. Dadurch können die Anfragen zweier Softwarekomponenten-Instanzen immer unterschieden werden, auch wenn ansonsten alle anderen Parameter des MISK-Identifikators identisch sind. Dadurch ist es möglich, zwischen einer wiederholten Anfrage derselben Softwarekomponenten-Instanz, weil z.B. die Antwort des Kontrollservers verloren gegangen ist, und der Ausführung einer Kopie zuverlässig zu unterscheiden.

[4] Die Erkennung von Replay-Attacken sind ein großes Problem im IoT-Bereich [60]. Bei einer Replay-Attacke wird versucht, eine Nachricht zwischen zwei Kommunikationspartnern (z.B. Client und Server) aufzuzeichnen und diese, bzw. auch nur Teile wie Benutzername und Passwort, wiederzuverwenden [61]. Eine Nonce kann eine solche Replay-Attacke verhindern, da für jede Nachricht eine neue Nonce verwendet werden muss und ein Wiedereinspielen einer Nachricht so erkannt werden kann [62].

4.4.5 Kontrollserver-Nonce

Der letzte Parameter ist ebenfalls ein Nonce-Wert. Anders als bei der zuvor beschriebenen Softwarekomponenten-Nonce wird die Kontrollserver-Nonce (Server-Nonce) auf Seite des Kontrollservers generiert. Bei jeder Anfrage an den Kontrollserver wird in der Antwort eine neue Nonce mitgesendet. Der Kontrollserver merkt sich die Nonce in Verbindung mit den anderen Werten des MISK-Identifikators und erwartet diese für die nächste Anfrage als Parameter im MISK-Identifikator. Die Server-Nonce ist nur für eine Anfrage gültig.

Das Ziel ist zum einem die Erkennung der zuvor erwähnten Replay-Attacke, zum anderen, dass eine SW-ID mehrfach verwendet werden kann. Die Server-Nonce ist für eine SW-ID einmal gültig. Wenn eine neue Softwarekomponente mit einer bereits verwendeten SW-ID instanziiert wird, muss diese auch die aktuelle Nonce zu der SW-ID kennen. Sollte es einem Angreifer gelingen, die Server-Nonce ebenfalls zu kopieren und in der Anfrage an den Kontrollserver mitzuschicken, würde diese Anfrage als gültig gewertet werden. Allerdings ist nun die nächste Anfrage der Softwarekomponenten-Instanz, welcher ursprünglich die Server-Nonce zugewiesen wurde, ungültig, da die Server-Nonce bereits für die SW-ID verwendet wurde. Die SWKP-Nonce kann nicht als Indikator verwendet werden, da sich diese nach jedem Neustart ändert.

4.5 Varianten zur Erstellung des Identifikators

Die Struktur des eindeutigen MISK-Identifikators wurde bereits im vorigen Abschnitt genauer erläutert. Da die Erstellung und Verwendung des MISK-Identifikators jedoch von verschiedenen Faktoren abhängen, z.B. ob Intel SGX auf einem Gerät verfügbar ist, werden vier verschiedene Varianten zur Erstellung des MISK-Identifikators eingeführt. Diese werden nachfolgend näher beschrieben. Die Abfolge der Prüfungen und welche Variante bei welcher Bedingung verwendet wird, ist in Abbildung 4.2 dargestellt.

Die Ausgangsituation, welche für alle vier Varianten gleich ist, sieht vor, dass der Code der Softwarekomponente in einem lauffähigen Zustand ausgeliefert ist. In der Softwarekomponente ist MISK bereits integriert und alle benötigten zusätzlichen Komponenten, wie beispielsweise der Enklaven Code für Variante 1, ist enthalten. Die Softwarekomponente kann nun mit einer SW-ID instanziiert wird. Während der ersten Instanziierung wird die zu verwendende Variante geprüft und die bereits beschriebene Softwarekomponenten-Nonce (SWKP-Nonce) erstellt. Bei den Varianten 2 und 3 wird zusätzlich die Geräte-ID ausgelesen bzw. generiert. Die Parameter werden im MISK-Identifikator gesetzt, bis auf den noch fehlenden Parameter Kontrollserver-Nonce (Server-Nonce). Fehlt dieser Parameter bei einer Anfrage im MISK-Identifikator, wird eine Anfrage vom Kontrollserver als erstmalige Registrierungsanfrage gewertet. Damit die Registrierung erfolgreich ist, muss die SW-ID gültig sein und darf vorher noch nicht verwendet worden sein. Um eine bereits verwendete SW-ID zur erneuten Registrierung verwenden zu können, sind manuelle Schritte notwendig, die Teil eines Lizenz-

modells und nicht dieses Konzeptes sind. Ist die SW-ID gültig, wird die Server-Nonce generiert und zusammen mit den anderen Parametern auf Seiten des Kontrollservers gespeichert. Die Server-Nonce wird in der Antwort an die Softwarekomponenten-Instanz zurückgesendet. Die Server-Nonce ist nun für die nächste Anfrage der Softwarekomponenten-Instanz gültig. Bei jeder weiteren Anfrage, die nun als Identifizierungsanfragen bezeichnet werden, wird in der Antwort des Kontrollservers eine neue Server-Nonce mitgesendet, die wieder nur für die nächste Identifizierungsanfrage gültig ist. Das beschriebene Verhalten wird in Abbildung 4.3 dargestellt.

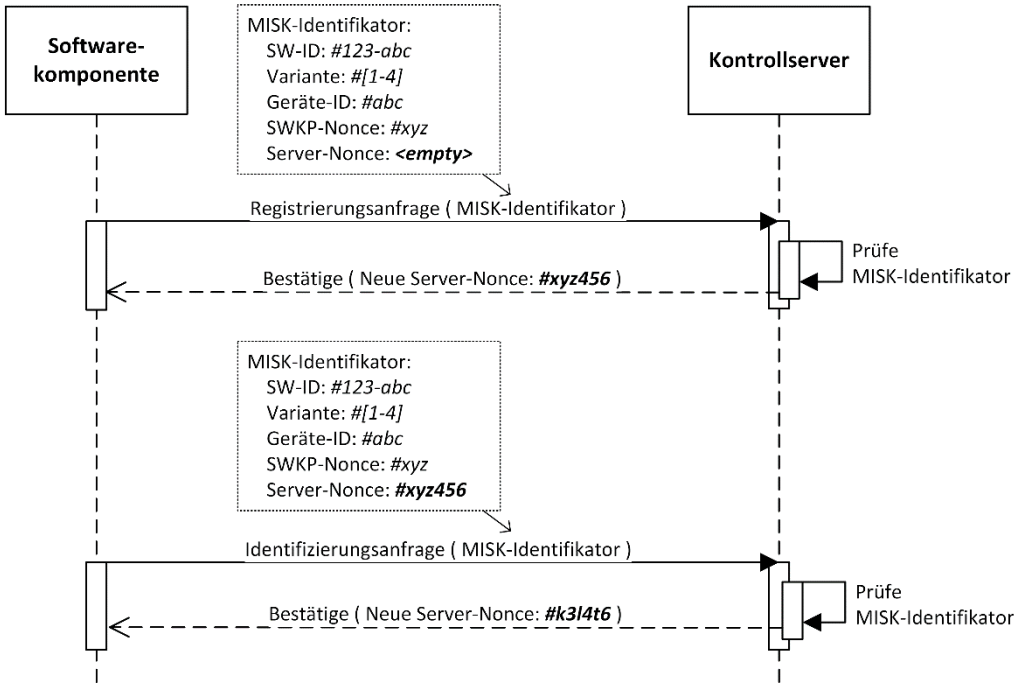

Abbildung 4.3: Verwendung des Parameters Kontrollserver-Nonce bei der Registrierungs- und Identifizierungsanfrage

Die Identifizierung und die damit verbundene Überprüfung auf rechtmäßige Ausführung einer Softwarekomponenten-Instanz kann dabei entweder bei jedem Start einer Softwarekomponente durchgeführt werden, somit wird nur die Inbetriebnahme kontrolliert, oder aber als Mechanismus zur Identifizierung von Anfragen an andere Anwendungen oder Services in einem verteilten System. Der Unterschied der beiden Szenarien ist in Abbildung 4.4 nochmals verdeutlicht. Das Prüfverfahren, welche Variante verwendet wird, muss bei jedem Start einer Softwarekomponente durchgeführt werden. Nur so kann festgestellt werden, ob es zwischen den Daten im bereits gespeicherten MISK-Identifikator und der aktuellen Systemumgebung Unterschiede gibt.

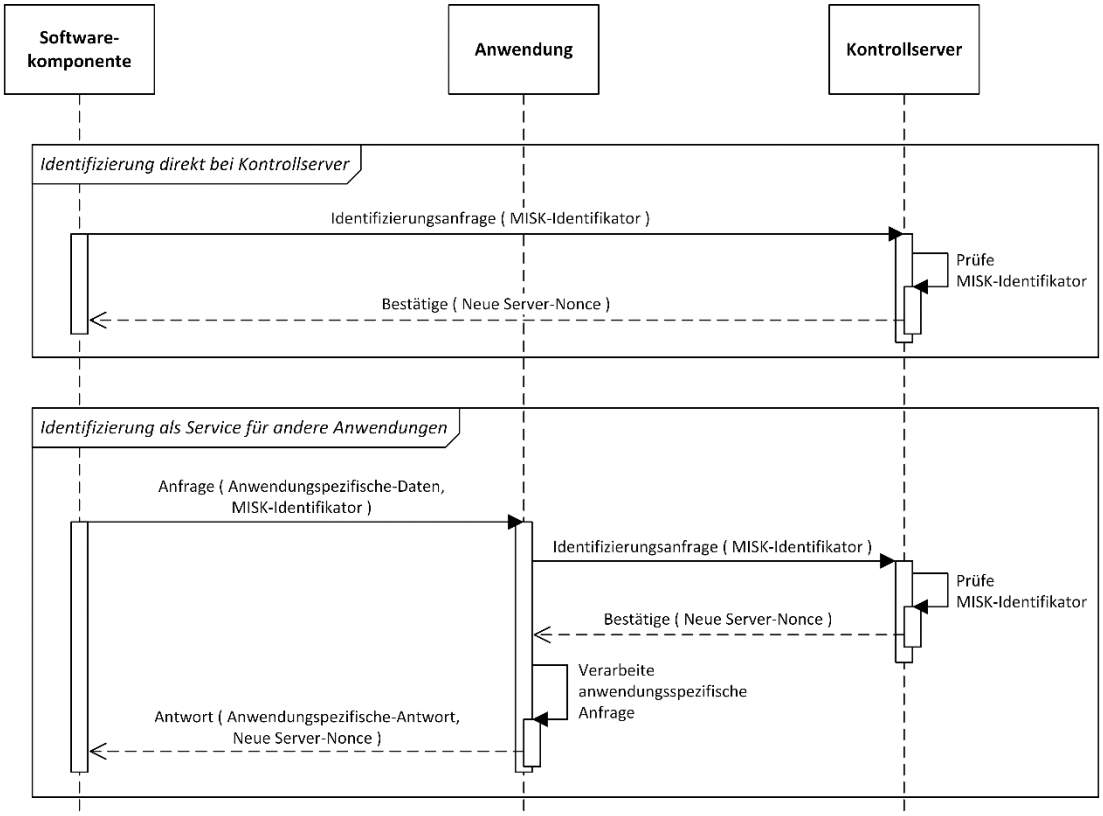

Abbildung 4.4: Zwei unterschiedliche Szenarien zur Identifizierung beim Kontrollserver

4.5.1 Variante 1 (Sicherheitstechnologie)

Variante 1 ist durch den Einsatz der Sicherheitstechnologie die Variante mit dem höchsten Vertrauenslevel und daher wird diese Variante an erster Stelle geprüft. Als Sicherheitstechnologie wird, wie in Abschnitt 4.3 beschrieben, Intel SGX verwendet. Die in Abschnitt 3.2.1 beschriebenen Sicherheitsmechanismen, wie der sicheren Ausführungsumgebung (Enklave), der Attestierung des ausgelieferten Quellcodes und der Datenverschlüsselung (Sealing), garantieren im Vergleich zu den Varianten 2 bis 4 die höchste Sicherheit zur Vermeidung, dass der Identifizierungsmechanismus umgangen werden kann. Voraussetzung zur Verwendung von Variante 1 ist, dass Intel SGX von dem Prozessor des Geräts unterstützt wird, Intel SGX im BIOS aktiviert ist und dass das Intel SGX Platform Software Package (PSW) installiert ist. Ist eine der Bedingungen nicht erfüllt, muss eine der anderen Varianten gewählt werden.

Die zuvor beschriebene allgemeine Ausgangssituation und in Abbildung 4.3 dargestellte Registrierung unterscheidet sich bei Variante 1 in einigen Punkten. Der Grund dafür ist, dass durch Intel SGX zusätzliche Mechanismen zur Verfügung stehen, die zur Erhöhung der Sicherheit eingesetzt werden. Die verwendeten Mechanismen sind die Enklave (ein durch den Prozessor geschützter Adressraum), die Attestierung (ein Prozessorbasierter Signierungs- und Bestätigungsprozess) und das Sealing (eine spezielle Prozessorbasierte Datenverschlüsselung).

Für das Konzept kann eine Enklave als sicherer Container innerhalb einer Softwarekomponenten-Instanz betrachtet werden, auf deren Inhalt auch die Softwarekomponenten-Instanz nur über zuvor definierte Schnittstellenfunktion (ECALLs und OCALLs) zugreifen kann [46]. Das bedeutet, dass über die Schnittstellen der Enklave sichergestellt werden kann, dass bestimmte Daten nie den gesicherten Speicherbereich der Enklave verlassen. Die Überprüfung, dass der Quellcode und somit auch die Schnittstellen einer Enklave nachträglich nicht verändert wurden, kann der von Intel SGX angebotenen Attestierungsservice verwendet werden. Bei der Verwendung von Intel SGX gibt es für MISK zwei entscheidende Einschränkungen, die beachtet werden müssen: Zum einen können aus einer Enklave heraus keine externen Ein- und Ausgabe-Aufrufe gemacht werden, d.h. die Kommunikation ist immer nur über Prozesse auf demselben Gerät möglich. Zum anderen kann nicht verhindert werden, dass andere Softwarekomponenten-Instanzen auf demselben Gerät die Daten, die durch Sealing verschlüsselt wurden, theoretisch auch wieder entschlüsseln können [46]. Der Schlüssel für das Sealing wird vom Prozessor abgefragt. Hierfür gibt es, wie in Abschnitt 3.2.1 genauer beschrieben, zwei verschiedene Möglichkeiten. Beide Möglichkeiten verhindern jedoch nicht, dass zwei Instanzen einer Enklave denselben Schlüssel abfragen können. Ohne diese Einschränkungen könnte eine Enklave direkt mit dem Kontrollserver kommunizieren und die erhaltenen Daten, wie die Server-Nonce, könnten über das Sealing verschlüsselt auf dem Gerät gespeichert werden, ohne dass die Daten über den unsicheren Bereich außerhalb der Enklave übertragen werden müssen.

Zur Umgehung der zuvor genannten Problematik, dass eine Enklave keine externen Aufrufe machen kann, wird das Diffie-Hellman-Schlüsselaustausch-Protokoll (DHS-Protokoll) eingesetzt. Dieses erlaubt den sicheren Austausch eines geheimen Schlüssels über einen unsicheren Kanal [13]. Das DHS-Protokoll wird im Abschnitt 2.1 genauer beschrieben.

Nachfolgend wird der Ablauf der Registrierung und Identifizierung zwischen einer Softwarekomponenten-Instanz und dem Kontrollserver mit den zuvor beschriebenen Sicherheitsmechanismen von Intel SGX in Verbindung mit dem DHS-Protokoll beschrieben. Die wichtigsten Schritte des beschriebenen Szenarios, werden in Abbildung 4.5 dargestellt. Die Nummern der Abbildung sind nachfolgend entsprechend der Beschreibung gelistet.

1) Die Softwarekomponente wird mit einer SW-ID instanziiert und das Prüfverfahren liefert Variante 1 (Sicherheitstechnologie) als Ergebnis.
   Im nächsten Schritt wird die Enklave von der Softwarekomponenten Instanz aus instanziiert. Der Code der Enklave wird zusammen mit der Softwarekomponente ausgeliefert. Die ECALL-Funktion der Enklave zur Einleitung des Identifizierungsprozesses wird gestartet. Dabei wird die SW-ID an die Enklave mit übergeben.
   Die Enklave erkennt, dass noch kein MISK-Identifikator für die SW-ID existiert und sendet eine Registrierungsanfrage an den Kontrollserver. Dies geschieht über einen Aufruf aus der Enklave heraus (OCALL) über die Softwarekomponenten-Instanz, da die Enklave, wie bereits erwähnt, keinen direkten externen Aufruf machen kann.

2) Der Kontrollserver erkennt anhand der fehlenden Server-Nonce, dass es eine Registrierungsanfrage ist und prüft zunächst die SW-ID auf ihre Gültigkeit.

3) Ist die SW-ID gültig, wird im nächsten Schritt eine sogenannte Quote von der Enklave angefordert. Diese kann als spezieller signierter Report oder Prüfsumme über den Quellcode und Daten einer Enklave gesehen werden [48]. Die Signierung wird mit einem privaten Schlüssel des Intel Prozessors durchgeführt.

4) Die Quote wird zurück an den Kontrollserver gesendet. Diese kann nun prüfen, ob die Prüfsumme der Enklave mit der ursprünglichen Version der ausgelieferten Enklaven Version übereinstimmt. Damit kann sichergestellt werden, dass der Quellcode der Enklave nicht nachträglich geändert wurde und somit auch das Verhalten.

5) Stimmt die Prüfsumme überein, kann über einen Attestierungsservice von Intel geprüft werden, ob die Quote wirklich von einem Intel Prozessor durchgeführt wurde. Die Überprüfung wird über den zugehörigen öffentlichen Schlüssel des Intel Prozessors gemacht. Somit ist sichergestellt, dass auf dem Gerät, auf dem die Softwarekomponenten-Instanz ausgeführt wird, eine sichere Ausführungsumgebung mit unveränderten Quellcode der Enklave ausgeführt wird.

6) Ist die Überprüfung erfolgreich abgeschlossen, wird das DHS-Protokoll eingeleitet und die notwendigen Werte für den DHS, welche in Abbildung 4.5 mit g, p und A benannt sind, werden an die Softwarekomponenten-Instanz gesendet. Diese übermittelt die Daten weiter an die Enklave.

7) In der Enklave wird ein neuer Wert b generiert und ein Schlüssel K kann aus den Werten b, g, p und A berechnet werden[5]. Der Schlüssel K ist nun die neue Server-Nonce.

8) Die Server-Nonce kann nun über das Sealing sicher aus der Enklave gespeichert werden. Das Sealing gewährleistet, dass die verschlüsselten Daten nur von demselben Prozessor wieder entschlüsselt werden können, daher wird bei Variante 1 auch keine zusätzliche Geräte-ID benötigt.

9) Im nächsten Schritt wird ein vorberechnete Wert B an den Kontrollserver übermittelt.

10) Diese kann nun auch den Wert K (denselben symmetrischen Schlüssel) berechnen. Die Werte a und b wurden dabei nie übermittelt, sondern nur jeweils die vorberechneten Werte A und B. Dadurch ist sichergestellt, dass weder der Schlüssel K, noch die notwendigen Werte zur Berechnung von K, im ungesicherten Speicherbereich der Softwarekomponenten-Instanz waren.

[5] Der genaue Ablauf und die Funktionsweise des DHS wird in Abschnitt 2.1 erläutert.

Der Kontrollserver speichert nun zusätzlich zu den Werten des MISK-Identifikators aus der Registrierungsanfrage den Schlüssel K als Server-Nonce. Bei der nächsten Anfrage der Softwarekomponenten-Instanz wird nun vom Kontrollserver überprüft, ob zu der SW-ID eine gültige Server-Nonce mitgesendet wurde.

Damit nun nicht für jede Identifizierungsanfrage jeweils zwei Anfragen zum Austausch der Server-Nonce über das DHS-Protokoll benötigt werden, unterscheidet sich die Identifizierungsanfrage zur Registrierungsanfrage darin, dass die Werte g, p und A auf Seiten der Softwarekomponenten Instanz berechnet werden und bei der Identifizierungsanfrage mit übertragen werden. Der Kontrollserver kann nach der erfolgreichen Prüfung der übermittelten Server-Nonce den Wert b generieren. Anschließend kann wieder K, die nächste Server-Nonce, berechnet werden und B kann an die Enklave der Softwarekomponenten-Instanz in der Antwort mitgesendet werden, so dass auch hier wieder K berechnet werden kann.

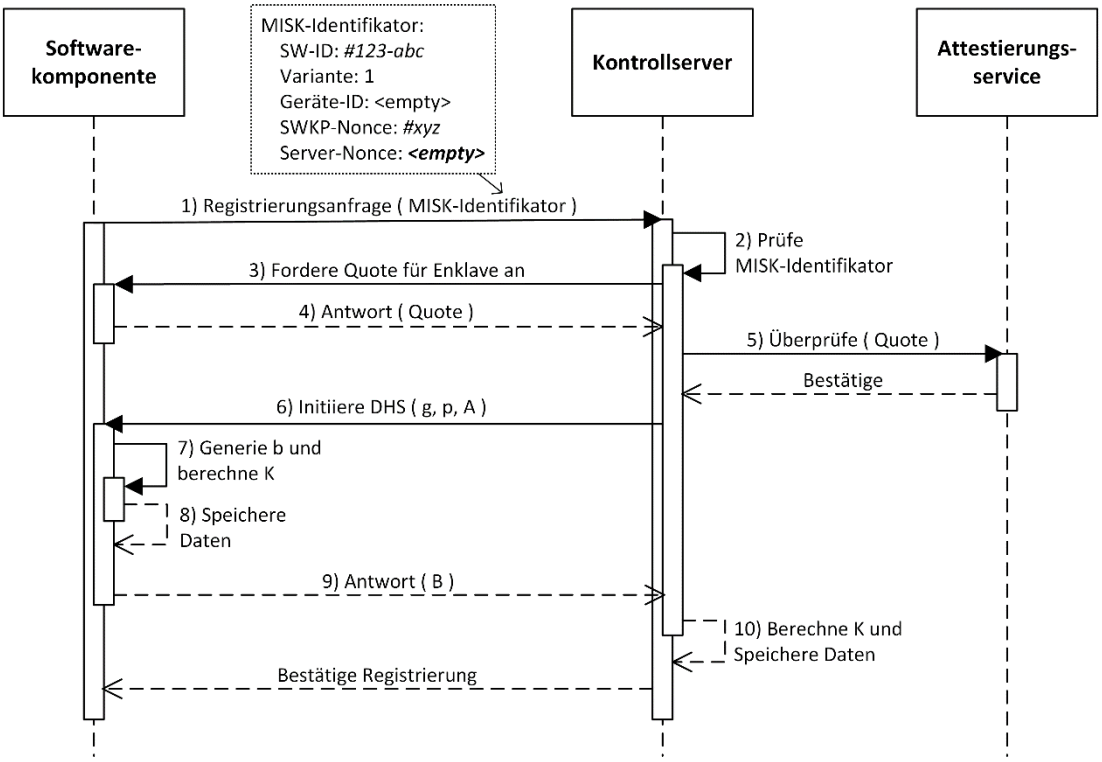

Abbildung 4.5: Protokoll zur Registrierung einer Softwarekomponenten-Instanz mit Erstellung des MISK-Identifikators durch Variante 1

Durch Verwendung des DHS-Protokolls kann nun verhindert werden, dass die Server-Nonce außerhalb des gesicherten Bereichs der Enklave im Klartext übertragen wird, obwohl die Enklave nicht direkt mit dem Kontrollserver kommunizieren kann. Hierdurch kann verhindert

werden, dass z.B. durch das Auslesen von Speicherbereichen (Seitenkanalangriffen) Informationen wie die Server-Nonce erlangt werden.

Die zweite Einschränkung, dass Enklaven der gleichen Version[6] dieselben Daten wieder entschlüsseln können, die über das Sealing einer anderen Enklaven-Instanz gespeichert wurden, kann nicht verhindert werden. Das bedeutet, dass bei der Ausführung einer zweiten Softwarekomponenten-Instanz auf demselben Gerät, mit der gleichen SW-ID, die Server-Nonce der ersten Softwarekomponenten-Instanz ausgelesen und für die zweite Instanz verwendet werden kann. Die Server-Nonce kann zwar nur einmalig verwendet werden, da jedoch beide Softwarekomponenten-Instanzen den Zugriff auf die Server-Nonce haben, können auch beide Instanzen im Wechsel die Server-Nonce verwenden. Um dies teilweise zu verhindern, kann der Kontrollserver die SWKP-Nonce verwenden. Da diese bei jedem Start einer Softwarekomponenten-Instanz anders ist, kann die Server-Nonce zwar nicht fest an eine SWKP-Nonce gebunden werden, wie es beispielsweise mit der Geräte-ID bei Variante 2 und 3 gemacht wird, jedoch kann sich der Kontrollserver die Historie der verwendeten SWKP-Nonces zu einer SW-ID merken. Mehrere Anfragen mit derselben SW-ID und SWKP-Nonce im MISK-Identifikator sind zulässig, bis eine neue SWKP-Nonce verwendet wird. Die neue SWKP-Nonce wird als Neustart der Softwarekomponenten-Instanz betrachtet. Kommt jedoch danach erneut eine Anfrage mit einer zuvor verwendeten SWKP-Nonce, wird diese Anfrage abgelehnt. Da jede erstellte SWKP-Nonce eindeutig sein muss, weiß der Kontrollserver, dass eine der vorigen Server-Nonces von einer anderen Softwarekomponenten-Instanz verwendet wurde. Aufgrund dessen, dass das Teilen der Server-Nonce zwischen verschiedenen Instanzen bei jeder Variante möglich ist, bei Variante 4 sogar auf unterschiedlichen Geräten, wird der Mechanismus zur Überprüfung der Historie der SWKP-Nonce für jede der vier Varianten verwendet.

Ein zweiter Mechanismus zur Vermeidung des geteilten Zugriffs auf die Server-Nonce, der bei der Implementierung der Enklave verwendet werden kann, ist, dass der Pfad zu den verschlüsselten Daten im Dateisystem relativ zur Softwarekomponenten-Instanz ist und in der Enklave hinterlegt ist und nicht beispielsweise beim Aufruf der Schnittstellenfunktion der Enklaven mitgegeben wird. Dadurch hat jede Softwarekomponenten-Instanz einen eigenen Speicherbereich für die verschlüsselten Daten. Dies verhindert zwar nicht, dass der Speicherbereich kopiert werden kann, erschwert jedoch den gleichzeitigen Betrieb von mehreren Softwarekomponenten-Instanzen, mit der gleichen SW-ID auf demselben Gerät. Zusätzlich könnte die verschlüsselte Datei eine Signatur enthalten, die den Speicherort enthält. Entspricht die Signatur nicht dem aktuellen Speicherort, kann die Enklave dies als Indikator verwenden, dass die Datei von einem anderen Speicherort kopiert wurde. Eine Einschränkung bei dieser Erwei-

[6] Die Verschlüsselung der Daten über das Intel SGX Sealing basiert auf einem kryptographischen Schlüssel, der vom Prozessor stammt, und dem kryptographischen Wert (Prüfsumme) der Enklave. Ist der kryptographische Wert gleich, können die Instanzen dieselben Daten wieder entschlüsseln [48]. Da mit einer Softwarekomponenten Version auch der Enklave-Quellcode mit ausgeliefert wird, ist bei der mehrfachen Instanziierung einer Softwarekomponente auch die kryptographische Identität der Enklaven gleich.

terung ist jedoch, dass die Information, was der aktuelle Speicherort der Softwarekomponenten-Instanz ist, auch von dieser an die Enklave übergeben werden muss, da eine Enklave in einem gesonderten Speicherbereich außerhalb des Betriebssystems läuft und daher keine direkten Zugriffe auf einen externen Speicherbereich machen kann [46]. Das bedeutet, die Information über den Speicherort der Softwarekomponenten-Instanz kommt aus dem unsicheren Bereich und ist daher als nicht vertrauenswürdig einzuschätzen.

Zusammenfassend werden für Variante 1 im MISK-Identifikator die SW-ID, die Variante, die SWKP-Nonce und die Server-Nonce benötigt. Die Geräte-ID wird nicht benötigt, da die verschlüsselten Daten nur durch den gleichen Prozessor wieder entschlüsselt werden können, wodurch sichergestellt ist, dass eine Softwarekomponente mit einer bereits verwendeten SW-ID nur auf demselben Gerät (bzw. abhängig von dem Prozessor) wiederverwendet werden kann.

Anmerkung: Da es sich, wie in Abschnitt 3.2.1 beschrieben, bei Intel SGX um eine Erweiterung der Prozessor-Instruktionen handelt, kann Intel SGX auch in einer virtualisierten Umgebung verwendet werden, wenn die zusätzlichen Prozessor-Instruktionen virtualisiert verfügbar sind. Für die Virtualisierungstechnologien KVM [63], XEN [64] und VirtualBox [65] existieren beispielsweise bereits Lösungen zur Verwendung von Intel SGX [66, 67]. Dies ist auch ein Grund, warum auf die Verwendung der Gerät-ID bei Variante 1 verzichtet wird (siehe dazu Abschnitt 4.5.3).

4.5.2 Variante 2 (Hersteller Geräte-Identifikator)

Ist Variante 1 nicht verwendbar, wird geprüft, ob ein spezieller vom Gerätehersteller gesetzter eindeutiger Geräte-Identifikator bzw. Seriennummer vorhanden ist. Ein solcher Geräte-Identifikator, der in einen Hardwarebaustein (Read-only Memory (ROM)) bei der Produktion hinterlegt wird, kommt häufig bei Industriegeräten zum Einsatz. Die Überprüfung kann beispielsweise über den Abgleich einer Liste mit Gerätemodellen gemacht werden, bei welchen bekannt ist, dass ein solcher Identifikator vorhanden ist. Für jedes Gerätemodell ist dann auch bekannt, wie der jeweilige Geräte-Identifikator ausgelesen werden kann, z.B. durch Verwendung einer vom Hersteller bereitgestellten Bibliothek.

Der ausgelesene Geräte-Identifikator wird dann als Geräte-ID Parameter im MISK-Identifikator gesetzt. Aus Gründen der Datensicherheit kann auf die Geräte-ID, vor der Übermittlung an den Kontrollserver, eine sichere Hash-Funktion[7] angewendet werden, da zur eindeutigen Identifizierung eines Geräts nicht der Wert selbst relevant ist, sondern nur, dass die Geräte-ID für ein Gerät fälschungssicher und reproduzierbar erstellt werden kann. Würde nun ausschließlich die Geräte-ID zusammen mit der SW-ID als Parameter für den MISK-

[7] Als sichere Hash-Funktionen werden zum Zeitpunkt der Erstellung der Arbeit Hashfunktion der Secure Hash Algorithm (SHA) 2 und 3 Familien gewertet [68, 69].

Identifikator verwendet werden, könnte so zwar ein Gerät eindeutig identifiziert werden, jedoch nicht die einzelnen Softwarekomponenten-Instanzen. Laufen auf einem Gerät mehrere Softwarekomponenten-Instanzen mit derselben SW-ID, ist eine Unterscheidung anhand der Geräte-ID nicht möglich. Daher werden auch bei dieser Variante zusätzlich die SWKP-Nonce und die Server-Nonce benötigt. Die Geräte-ID bietet hier jedoch einen zusätzlichen Sicherheitsmechanismus, da zwei Softwarekomponenten-Instanzen mit der gleichen SW-ID auf zwei unterschiedlichen Geräten anhand der Geräte-ID im MISK-Identifikator leicht unterscheidbar sind. Der Mechanismus mit den zwei Nonces wird daher für den Fall benötigt, wenn mehr als eine Softwarekomponenten-Instanz auf einem Gerät ausgeführt wird.

4.5.3 Variante 3 (Generierter Geräte-Identifikator)

Sind Variante 1 und 2 nicht anwendbar, wird überprüft, ob es sich um eine virtualisierte Umgebung handelt. Ist dies der Fall, wird Variante 4 gewählt, andernfalls die in diesem Abschnitt beschriebene Variante 3 (generierter Geräte-Identifikator).

Das Ziel dieser Variante ist es, wie bei der zuvor beschriebenen Variante 2, die eindeutige Identifizierung eines Geräts als zusätzlichen Sicherheitsmechanismus zu verwenden. Zur eindeutigen Identifizierung eines Geräts können verschiedene Hardwaremerkmale verwendet werden. Ein Beispiel aus der Praxis ist hierfür die Zuordnung einer *Microsoft Windows 10 Lizenz* zur Hardware. Ändern sich zu viele Hardwarekomponenten eines Geräts, kann dieses nicht mehr eindeutig identifiziert werden und die Lizenz verliert ihre Gültigkeit [70, 71]. Welche Hardwarewarekomponenten dabei genau berücksichtig werden, wird von Microsoft nicht bekanntgegeben. Generell eignen sich zur Identifizierung aber die Hardwarekomponenten, die selten oder gar nicht geändert werden, wie beispielsweise die Hauptplatine (Mainboard) oder der darauf verbaute Prozessor (CPU (Central Processing Unit)). Nachfolgend kann zur besseren Verständlichkeit ein Standard Computer (Personal Computer (PC)) als Gerät betrachtet werden.

Eine Voraussetzung für den Zugriff auf die Hardwarekomponenten, wie die bereits erwähnte Hauptplatine oder der Prozessor, ist, dass keine Virtualisierungschicht wie ein Hypervisor zwischen dem Gerät und der Softwarekomponenten-Instanz liegt. Diese würde den direkten Zugriff auf die Hardwarekomponenten verhindern [72, 73]. Deshalb muss die bereits erwähnte Prüfung auf eine virtualisierte Umgebung durchgeführt werden, um sicherzustellen, dass die benötigten Hardwareinformationen verfügbar sind.

Mögliche Informationen der Hardwarekomponenten zur Generierung einer eindeutigen Geräte-ID

Nachfolgend werden die verschiedenen Hardwarekomponenten und weitere abrufbare Systeminformationen auf die Verwendbarkeit hin geprüft, sowie die Vor- und Nachteile betrachtet.

Hauptplatine (Mainboard)

Die Hauptplatine, oft auch Mainboard oder Motherboard genannt, bildet die Basis für andere Hardwarekomponenten eines Geräts, wie der Prozessor, dem Arbeitsspeicher usw. [74]. Eine Hauptplatine kann mittels einer festen Seriennummer identifiziert werden und ist aufgrund der zentralen Funktion eine gute Möglichkeit ein Gerät zu identifizieren.

Prozessor (CPU)

Der Prozessor (CPU (Central Processing Unit)) befindet sich auf der Hauptplatine und kann, wie auch die Hauptplatine, über eine Seriennummer identifiziert werden. Da der Austausch eines Prozessors eher selten ist, bietet auch diese Information eine gute Möglichkeit, ein Gerät zu identifizieren.

Geräte-Seriennummer

Damit auch Hersteller ihre Geräte eindeutig identifizieren können, z.B. um Garantiefälle und andere Service-Angebote einem Gerät zuordnen zu können, versehen die Hersteller die Geräte oftmals mit einer eindeutigen Seriennummer. Ist eine solche Seriennummer vorhanden und kann über einen Standardmechanismus, wie z.B. durch die Verwendung einer Java Bibliothek zum Auslesen von Hardwareinformationen, ausgelesen werden, ist diese Geräte-Seriennummer ein zuverlässiger Identifikator. Da eine Eindeutigkeit über verschiedene Hersteller hinweg nicht garantiert ist, könnte beispielsweise die Modellbezeichnung als zusätzliche Information zur Erstellung der Geräte-ID verwendet werden. Im Unterschied zu dem in Abschnitt 4.5.2 Ansatz, kann bei, im Vorfeld nicht bekannten, Gerätemodellen nicht davon ausgegangen werden, dass eine solche Geräte-Seriennummer immer verfügbar ist.

Modellbezeichnung

Nahezu alle Geräte verfügen über eine Modellbezeichnung, die als Zeichenkette ausgelesen werden kann. Eine solche vom Hersteller vergebene Modellbezeichnung sieht für einen Laptop des Herstellers HP, der Modellreihe ZBook, wie folgt aus: *HP ZBook 15 G3*. Da die Modellbezeichnung bei allen baugleichen Geräten identisch ist, eignet sich diese Information nicht zur eindeutigen Identifizierung eines Geräts.

Speicher

Speicherbausteine sind auf den meisten Geräten in Form von Hauptspeicher (Arbeitsspeicher) und Massenspeicher, wie Festplatten oder Halbleiterfestplatten (Solid State Drive (SSD)), verfügbar. Informationen über den verfügbaren Speicher und Speicherausnutzung eignen sich nicht zur Erstellung eines Identifikators, da sich die Werte im laufenden Betrieb ständig ändern. Besser geeignet sind hier Informationen über die Speicherbausteine selbst, wie beispielsweise Hersteller, verfügbare Gesamtgröße und Seriennummern. Allerdings können Festplatten bzw. SSDs bei vielen Geräten leicht getauscht werden, wenn beispielsweise die aktuelle Speichergröße nicht mehr ausreicht.

Peripheriegeräte

Als Peripheriegeräte werden Geräte bezeichnet, die an ein anderes Gerät extern angeschlossen werden und zum Austausch von Informationen verwendet werden, wie z.B. Bildschirme, Tastaturen und USB-Geräte (Universal-Serial-Bus-Geräte) [74]. Von den Peripheriegeräten können verschiedenen Informationen gesammelt werden, wie z.B. die Seriennummern der angeschlossenen Bildschirme. Aus den verschiedenen Informationen ließe sich eine eindeutige Geräte-ID erstellen. Ein Vorteil ist, dass Geräte-Informationen vieler verschiedener Geräte verwendet werden können, und die Wahrscheinlichkeit, dass aus der Summe der Informationen eine eindeutige Geräte-ID entsteht, ist hoch. Der Nachteil dieser Kategorie von Geräten ist jedoch, dass diese leicht ausgetauscht werden können und dadurch keine zuverlässige Basis zur Erstellung der Geräte-ID bilden. Wird ein Peripheriegerät angeschlossen oder entfernt, muss auch jedes Mal die Geräte-ID angepasst werden. Zudem muss entschieden werden, bei welcher Abweichung eine eindeutige Identifizierung eines Geräts über die Peripheriegeräte nicht mehr möglich ist: Müssen z.B. mindestens fünf Peripheriegeräte angeschlossen sein und davon müssen 80% gleichbleiben?

Netzwerkschnittstelle

Moderne Geräte verfügen über verschiedene Komponenten (Netzwerkschnittstellen), sowohl hardwarebasierte als auch virtualisierte, zur Verbindung mit einem Netzwerk, wie z.B. mit dem Internet [74]. Da das zentrale Ziel eines Netzwerks der Informationsaustausch ist, müssen die Geräte, bzw. die Netzwerkschnittstellen, eindeutig identifiziert werden können. Dies geschieht z.B. über die IP-Adresse (Internetprotokoll-Adresse) oder die MAC-Adressen (Media-Access-Control-Adresse). IP-Adressen werden dabei als Kommunikationsadressen verwendet [75] und MAC-Adressen als Geräte- bzw. Physische-Adressen, z.B. bei der Verwendung von Ethernet [76, 77]. Ein Problem bei der Verwendung von IP-Adressen als Identifikator ist, dass sich diese über die Zeit ändern kann, z.B. bei Verwendung von DHCP (Dynamic Host Configuration Protocol). Auch MAC-Adressen sind keine zuverlässigen Identifikatoren, da eine MAC-Adresse überschrieben werden kann [78]. Aufgrund dessen, dass ein Gerät über viele verschiedene Netzwerkschnittstellen verfügen kann, könnte hier eine Variante gewählt werden, welche die Summe der verfügbaren Informationen verwendet und daraus einen eindeutigen Identifikator bildet. Auch externe Informationen über andere Geräte, die sich im selben Netzwerk befinden, könnten zur Erstellung eines kontextbasierten Identifikators Verwendung finden. Jedoch gibt es dabei zwei Probleme: Zum einen müsste es dem ans Netzwerk angeschlossene Gerät möglich sein, diese Informationen zu empfangen, sowohl technisch als auch aus Datensicherheitsgründen. Zum anderen entsteht hier dieselbe Problematik, wie bereits im Abschnitt Peripheriegeräte beschrieben: Wie viele externe Informationen werden benötigt, um eine zuverlässige Geräte-ID zu bilden, und ab welchem Grad der Veränderung dieser Informationen ist ein Gerät nicht mehr eindeutig identifizierbar?

Betriebssystem

Informationen zum Betriebssystem können in Form von Angaben zum Hersteller (z.B. Microsoft), der Betriebssystem-Familie (z.B. Windows), die allgemeine Betriebssystem-Version (wie z.B. XP, Vista oder 10) und einer Build-Nummer abgefragt werden. Wobei die letzte

Information, die Build-Nummer, sich nach jedem Update des Betriebssystems ändert und sich daher zur Identifikation nicht eignet. Bei den anderen Angaben besteht dieselbe Problematik wie bereits bei der Modellbezeichnung beschrieben, dass es keine eindeutige Information darstellt, da davon ausgegangen werden muss, dass es eine Vielzahl an Geräten mit der gleichen Betriebssystemversion gibt.

Auswahl der Hardwarekomponenten zur Generierung einer eindeutigen Geräte-ID

Die Untersuchung auf Eignung der einzelnen Hardwarekomponenten-Informationen im vorhergehenden Abschnitt hat gezeigt, dass sich nicht alle Informationen zur Erstellung einer eindeutigen Geräte-ID eignen. Eine Zusammenfassung mit Bewertung der Zuverlässigkeit der einzelnen Hardwarekomponenten ist in Tabelle 4.2 zu sehen. Ein weiterer wichtiger Aspekt, den es bei der Auswahl zu berücksichtigen gibt, ist, dass nicht jede Information bei jedem Gerät zur Verfügung steht bzw. auch, dass sich die Hardwarekomponenten über die Zeit ändern, wie es beispielsweise beim Tausch einer Festplatte der Fall ist. Aus diesem Grund wird zur Erstellung der Geräte-ID nicht nur eine Hardwarekomponenten-Information verwendet, sondern mehrere in Kombination.

Hardwarekomponente	Zuverlässigkeit
Hauptplatine (Mainboard)	◕
Prozessor (CPU)	◕
Geräte-Seriennummer	●
Modellbezeichnung	○
Speicher	◔
Peripheriegeräte	◔
Netzwerkschnittstelle	◔
Betriebssystem	○

Legende:
Nicht zuverlässig: ○ Sehr zuverlässig: ●

Tabelle 4.2: Bewertung der Zuverlässigkeit der einzelnen Hardwarekomponenten

Als zuverlässigste Information zur Identifizierung eines Geräts wird aufgrund der Unveränderbarkeit die Geräte-Seriennummer angesehen. Da eine Eindeutigkeit über verschiedene Hersteller hinweg jedoch nicht sicher ist, wird zusätzlich die Information über den Hersteller und die Modellbezeichnung zur Generierung der Geräte-ID verwendet.

Aufgrund dessen, dass eine gerätespezifische Seriennummer nicht mit Sicherheit für jedes Gerät zur Verfügung steht, wird für die Geräte-ID auch die Seriennummer der Hauptplatine verwendet. Aufgrund ihrer zentralen Funktion und dass ein Wechsel dieser Komponente eher

selten ist, wird auch diese als zuverlässige Information gewertet und für die Geräte-ID verwendet. Dabei muss jedoch mindestens eine der beiden Informationen, die Geräte-Seriennummer oder die Seriennummer der Hauptplatine verfügbar sein, andernfalls muss Variante 4 (Virtualisierte Umgebung) zum Einsatz kommen.

Zusätzlich zu den beiden Seriennummern und der Modellbezeichnung, wird die Seriennummer des Prozessors als vierte Information zur Generierung der Geräte-ID verwendet. Diese Information wird, wie in Tabelle 4.2 zu sehen, ebenfalls als zuverlässig betrachtet. Der Vorteil bei der Verwendung mehrerer Hardwarekomponenten-Informationen ist zum einen, dass die Summe der Informationen die Zuverlässigkeit bei der eindeutigen Identifizierung erhöht, zum anderen, dass dadurch auf Seiten des Kontrollservers mit verschiedenen Strategien entschieden werden kann, welcher Grad an Veränderung von Hardwarekomponenten zulässig ist. Ein Beispiel für eine solche Strategie ist: Steht die Geräte-Seriennummer zur Verfügung, darf die Hauptplatine oder der Prozessor getauscht werden. Ist jedoch keine Geräte-Seriennummer vorhanden, darf die Hauptplatine und der Prozessor nicht getauscht werden, da dann diese zwei Komponenten die zuverlässige Identifizierung des Geräts ausmachen. Das Schema der generierten Geräte-ID sieht dabei wie folgt aus:

```
Geräte-ID:
 1) Geräte-Seriennummer
 2) Hauptplatinen-Seriennummer
 3) Prozessor-Seriennummer
 4) Modellbezeichnung
```

Wie bereits bei Variante 2 beschrieben, kann auch bei dieser Geräte-ID eine sichere Hashfunktion auf die Informationen der Geräte-ID angewendet werden, um so die Übertragung und Speicherung von gerätespezifischen Informationen zu vermeiden. Anders als bei Variante 2 darf jedoch nicht auf die gesamte Geräte-ID eine Hashfunktion angewendet werden, da ansonsten die Zuordnung, welche Information der Geräte-ID sich geändert hat, nicht mehr möglich ist. Somit wäre auch eine Strategie zum Umgang mit veränderten Hardwarekomponenten auf Seiten des Kontrollservers nicht mehr möglich. Aus diesem Grund muss die Hashfunktion auf alle vier Werte einzeln angewendet werden. Ebenso gleich wie bei Variante 2, wird auch hier der beschriebene Mechanismus mit den beiden Nonces für den Fall benötigt, dass mehr als eine Softwarekomponenten-Instanz auf einem Gerät ausgeführt wird, andernfalls wäre das Tupel von SW-ID und Geräte-ID bereits eindeutig.

4.5.4 Variante 4 (Virtualisierte Umgebung)

Variante 4 kommt zum Einsatz, wenn keine der zuvor beschriebenen Varianten eingesetzt werden kann. Dies ist z.B. bei virtualisierten Umgebungen der Fall, wo kein Zugriff auf die

benötigten Hardwaremerkmale zur Erstellung einer Geräte-ID möglich ist. Wie in **Anforderung 4** beschrieben, soll MISK jedoch auch virtualisierte Umgebungen unterstützen.

Die Erkennung bereits bekannter Virtualisierungsumgebungen, wie VMware [79], Oracle VirtualBox [65] oder KVM [63], kann relativ zuverlässig über verschieden Merkmale wie Benennungen von Systemkomponenten oder MAC-Adressen geschehen [80]. Jedoch können diese Benennungen oder MAC-Adressen zum einen überschrieben werden, zum anderen können neue Virtualisierungstechnologien hinzukommen, die zum Zeitpunkt der Implementierung noch nicht bekannt sind. Daher sollte Variante 4 auch zum Einsatz kommen, falls einer der relevanten Geräte-Identifikatoren für Variante 3 nicht verfügbar ist, bzw. mit einem nicht gültigen Wert belegt ist.

Für Variante 4 wird der bereits zuvor beschriebene Mechanismus mit der SWKP-Nonce und der Server-Nonce verwendet. Dadurch kann erkannt werden, wenn mehrere Softwarekomponenten-Instanzen gleichzeitig mit der gleichen SW-ID ausgeführt werden. Anders als bei den zuvor beschriebenen Varianten besteht hier jedoch nicht die Möglichkeit die SW-ID an ein Gerät zu binden, da es bei einer virtualisierten Umgebung generell keine verlässlichen Informationen gibt, die nicht, wie bei **Möglichkeit 2** beschrieben, kopiert werden könnten. Daher ist es bei Variante 4 möglich, eine Softwarekomponente mit einer bereits verwendeten SW-ID auch auf einem anderen Gerät zu starten, vorausgesetzt, die benötigte Server-Nonce ist auch verfügbar. Damit sich nicht mehrere Softwarekomponenten-Instanzen dauerhaft die Daten eines MISK-Identifikators teilen bzw. synchronisieren können, sollte auch hier, wie schon bei Variante 1 beschrieben, der Kontrollserver sich die zuletzt verwendeten SWKP-Nonces merken, um eine mehrfach Nutzung einer SW-ID durch mehrere Softwarekomponenten-Instanzen zu erkennen. Im Vergleich zu den anderen drei Varianten, bietet Variante 4 den geringsten Schutz vor der mehrfachen Verwendung einer SW-ID und hat daher das geringste Vertrauenslevel.

Ein Angriffsmodell, das durch Variante 4 nicht gelöst werden kann, besteht bei einer dauerhaften, aber nicht zeitgleichen Ausführung von mehreren Softwarekomponenten-Instanzen mit der gleichen SW-ID auf unterschiedlichen Geräten. Bei Variante 1 wird dies durch Intel SGX verhindert und bei den Varianten 2 und 3 durch die Verwendung der Geräte-ID. Ein Szenario hierfür könnte folgendes sein: Zwei Softwarekomponenten werden immer im Wechsel gestartet, z.B. läuft die eine in der Nacht und die andere am Tag. Vor jedem Start muss lediglich ein Weg gefunden werden, um die Server-Nonce zwischen den beiden Instanzen zu synchronisieren. Durch den Neustart wiederholt sich die SWKP-Nonce nicht und der Kontrollserver kann dadurch den Wechsel in der Historie der SWKP-Nonces nicht erkennen. Ein solches Szenario wäre beispielsweise für zwei weit auseinanderliegende Zeitzonen denkbar, wenn in beiden Zeitzonen der Betrieb der Softwarekomponente jeweils am Tag ausreichend ist.

Zur Umgehung dieses Angriffsmodell könnte eine Art Heartbeat (Herzschlag), ein sich in regelmäßigen Abständen wiederholendes kurzes Signal, von der Softwarekomponenten-Instanz an den Kontrollserver gesendet werden. Bleibt dieses Signal aus bzw. kommt es für längere

Zeit nicht, muss eine erneute Registrierung durchgeführt werden. Dies hätte jedoch den großen Nachteil, dass jeder Neustart auch eine erneute Registrierung mit sich bringen würde, welche zusätzliche manuelle Schritte durch den Benutzer zum Zurücksetzen der SW-ID mit sich bringt. Des Weiteren würde ein Heartbeat einen zusätzlichen Datenverkehr im Netzwerk bedeuten.

Ein weiterer Mechanismus zur Vermeidung des Angriffsmodels könnte dadurch erreicht werden, dass der Kontrollserver eine Softwarekomponente über das Netzwerk erreichen kann. Bei der Registrierungsanfrage wird zusätzlich eine Adresse angegeben, z.B. eine IP-Adresse oder URL (Uniform Resource Locator), über welche die Softwarekomponenten-Instanz erreichbar ist. Bei jeder Identifizierungsanfrage wird die Adresse bestätigt oder aktualisiert. Ändert sich die Adresse, versucht der Kontrollserver die, für die SW-ID zuvor hinterlegte, Adresse zu kontaktieren. Erhält der Kontrollserver eine Antwort, weiß er, dass mehrere Instanzen mit der SW-ID ausgeführt werden und kann die neue Identifizierungsanfrage blockieren. Zudem kann der Kontrollserver, wie schon bei der SWKP-Nonce, sich die letzten Adressen merken und einen sich wiederholenden Wechsel feststellen. Der Nachteil bei dieser Erweiterung ist jedoch, dass jede Softwarekomponenten-Instanz über eine feste von außen erreichbarer Adresse verfügen muss, was aufgrund von Firewalls, die in den meisten Netzwerken eingesetzt werden, nicht möglich ist. Daher eignet sich auch die zweite Erweiterung nicht als Lösung.

4.6 Zeitbasierte Gültigkeit

In den vorigen Abschnitten wurde gezeigt, dass die gleichzeitige Ausführung von mehreren Softwarekomponenten-Instanzen mit der gleichen SW-ID unter bestimmten Voraussetzungen verhindert werden kann, z.B. wenn diese auf unterschiedlichen Geräten oder mit unterschiedlichen Varianten ausgeführt werden sollen. Werden zwei Instanzen mit der gleichen SW-ID auf demselben Gerät ausgeführt, ist eine parallele Ausführung jedoch solange möglich, bis eine der Softwarekomponenten-Instanzen erneut eine Identifizierungsanfrage an den Kontrollserver sendet. Bei Variante 4 ist dies auch auf verschiedenen Geräten möglich. Bei der im vorigen Abschnitt 4.5 beschriebenen Möglichkeit, die Server-Nonce zwischen zwei Softwarekomponenten-Instanzen zu synchronisieren, könnten die beiden Instanzen dauerhaft mit einer SW-ID ausgeführt werden. Um dies zu vermeiden, wird zusätzlich ein zeitbasierter Gültigkeitsmechanismus für die Server-Nonce eingeführt. Der Nachfolgend beschriebene Gültigkeitsmechanismus kann für alle vier Varianten von MISK verwendet werden.

Wird eine Server-Nonce vom Kontrollserver generiert, erhält diese eine Gültigkeitsdauer von einer festgelegten Zeit t_{valid}. Innerhalb dieser Zeit muss eine Softwarekomponenten-Instanz erneut eine Identifizierungsanfrage mit der aktuell gültigen Server-Nonce an den Kontrollserver stellen, um die nächste gültige Server-Nonce zu erhalten. Wird die Gültigkeitsdauer von t_{valid} beispielsweise auf eine Stunde gesetzt, hätten zwei Softwarekomponenten-Instanzen nur noch die Möglichkeit, sich innerhalb von t_{valid} zu synchronisieren. Läuft die Zeit t_{valid} ab, ohne dass eine erneute Identifizierungsanfrage gestellt wurde, ist die Server-Nonce ungültig und keine der beiden Softwarekomponenten-Instanzen kann diese weiterverwenden. An

dieser Stelle wäre dann eine erneute Registrierung bzw. Zurücksetzen der SW-ID nötig, die manuelle Schritte eines Benutzers erfordern. Der beschriebene zeitbasierte Gültigkeitsmechanismus wird in Abbildung 4.6 nochmals verdeutlicht.

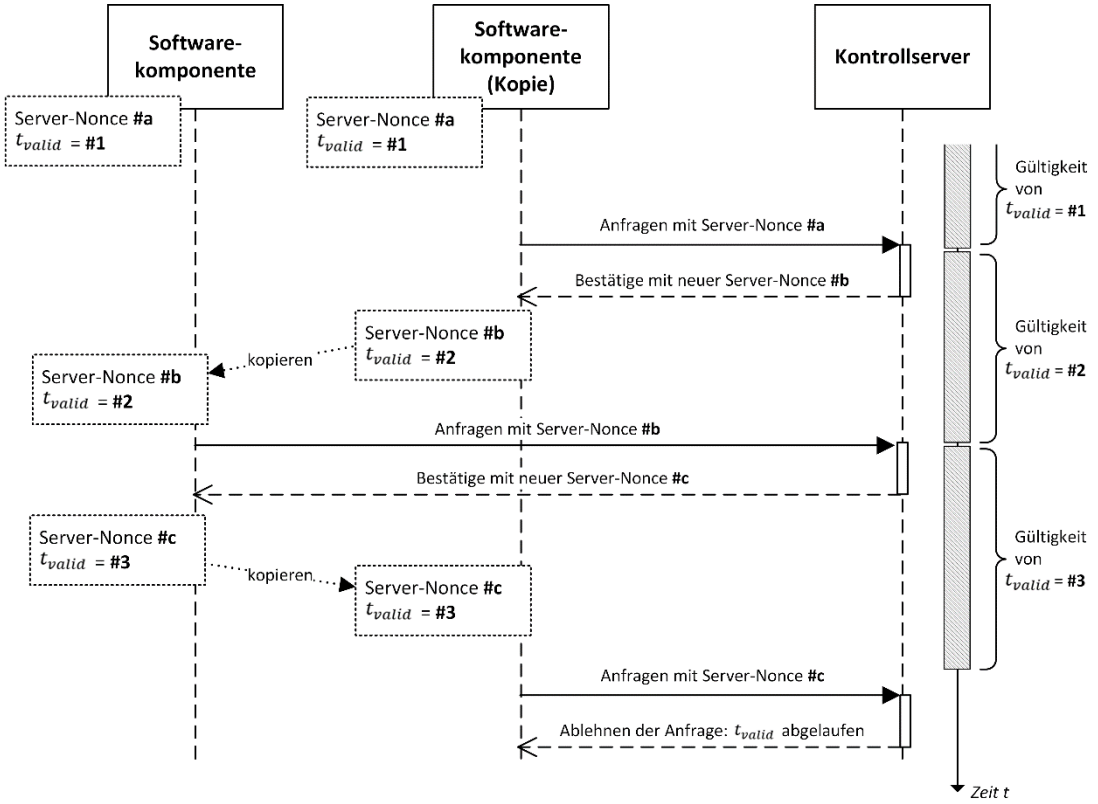

Abbildung 4.6: Server-Nonce mit zeitbasierter Gültigkeit

Die Zeit in der zwei Softwarekomponenten-Instanzen mit einer SW-ID erfolgreich eine Identifizierungsanfrage stellen können, kann durch diesen Mechanismus eingeschränkt werden. Damit die Ausführung von mehreren Softwarekomponenten-Instanzen mit einer SW-ID ganz verhindert werden kann, muss eine zusätzliche Einschränkung zu der zeitbasierten Server-Nonce eingeführt werden. Hierfür wird zu der definierten Gültigkeitsdauer t_{valid}, ein zweites Zeitfenster t_{renew} festgelegt. Die beiden Zeitfenster sind in Abbildung 4.7 chronologisch dargestellt. In dem Zeitfenster t_{renew} muss nun die nächste Identifizierungsanfrage gestellt werden. Zusätzlich ist in der Zeit von t_{renew} nur genau eine Identifizierungsanfrage zulässig.

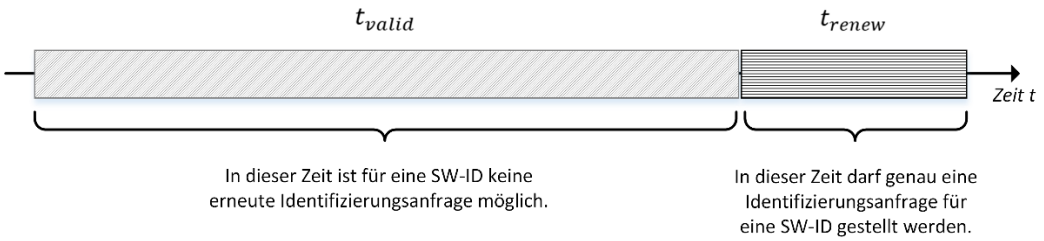

Abbildung 4.7: Die definierten Zeitfenster t_{valid} und t_{renew} für eine Server-Nonce

Der Wert von t_{renew} sollte dabei kleiner als der von t_{valid} sein. Wird t_{valid} beispielsweise auf eine Stunde gesetzt, wäre ein guter Wert für t_{renew} eine Minute, da nun nur noch in der Zeit von t_{renew} die Möglichkeit besteht, eine Kopie einer Softwarekomponenten-Instanz erfolgreich auszuführen. Alle anderen Kopien einer Softwarekomponenten-Instanz können nun keine neue Server-Nonce zu einer SW-ID bekommen. Das bedeutet, egal wie viele Kopien einer Instanz in der Zeit t_{valid} erstellt werden, kann keine in dem Zeitfenster von t_{valid} erfolgreich gestartet werden. Im Zeitfenster von t_{renew} kann anschließend nur eine Softwarekomponenten-Instanz eine neue gültige Server-Nonce bekommen, nämlich die, die als erstes eine Identifizierungsanfrage stellt. Bei der Umsetzung dieses Verfahrens muss darauf geachtet werden, dass der Verlust der Gültigkeit einer Server-Nonce dazu führt, dass eine Softwarekomponenten-Instanz nicht mehr weiter ausgeführt werden darf.

Ein Nachteil der zeitbasierten Gültigkeit einer Server-Nonce ist, dass eine bestimmte Aktion zu einem fixen Zeitpunkt, bzw. Zeitfenster, durchzuführen ist, um das Protokoll einzuhalten. Zum einen bedeutet das, dass der Mechanismus Nebenläufigkeit (Parallelität) in der Ausführung benötigt und die Ausführung einer Softwarekomponenten-Instanz unterbrechen können muss. Zum anderen wird eine zuverlässige Verbindung zum Kontrollserver im Zeitfenster t_{renew} vorausgesetzt. Des Weiteren muss berücksichtigt werden, welche Schritte notwendig sind, wenn eine Softwarekomponenten-Instanz für einen längeren Zeitraum als $t_{valid} + t_{renew}$ nicht ausgeführt wird. Wenn es eine geplante Unterbrechung ist, könnte beispielsweise das Zeitfenster t_{valid} um einen Wert t_{extend} einmalig verlängert werden, für nicht geplante Unterbrechungen wären dann jedoch wieder manuelle Schritte eines Benutzers nötig, um eine SW-ID erneut verwenden zu können.

5 Implementierung

In diesem Kapitel wird die prototypische Umsetzung des Konzepts, welches im vorangegangenen Kapitel 4 vorgestellt wurde, beschrieben. Zunächst werden die gewählten Technologien erläutert und anschließend werden die Architektur und die Schnittstellen der einzelnen Komponenten erklärt. Im Anschluss werden die Umsetzung des Prüfverfahren zur Variantenauswahl und die Anfragenvalidierung auf Seiten des Kontrollservers erläutert.

5.1 Auswahl der Technologien

Zur Umsetzung des Konzepts wurden die Programmiersprachen Java und C/C++ gewählt. Java wurde gewählt, da diese Programmiersprache im IIoT-Umfeld, in welchem das Konzept entwickelt wurde, die meist verwendete ist und somit die beste Kompatibilität zu den anderen Softwarekomponenten bietet. Zur Verwaltung der verwendeten Abhängigkeiten und zum Builden[1], wurde das Build-Tool Apache Maven [81] gewählt. Auch hier ist die Entscheidung durch die weite Verbreitung im Einsatzumfeld getroffen worden.

Wie bereits in Abschnitt 3.2.1 beschrieben, können Intel SGX Anwendungen nur mit C/C++ umgesetzt werden, da nur für diese Programmiersprache ein Software Development Kit (SDK) für Intel SGX verfügbar ist [46]. Aus diesem Grund wurde der Teil des Konzepts, der Intel SGX verwendet, in C/C++ entwickelt.

Zum Auslesen der Hardwareinformation, die zur Erstellung der Geräte-ID benötigt werden, wurde die Open-Source Bibliothek OSHI (Operating System & Hardware Information) [82] gewählt. OSHI unterstützt verschiedene Betriebssystem wie Windows, Linux, Mac OS und andere Unix Systeme und lässt sich über Maven einbinden. Ein großer Vorteil von OSHI ist, dass es bereits eine Liste von MAC-Adressen und Systembezeichner der gängigsten Virtualisierungstechnologien, wie beispielsweise VMware [79], Oracle VirtualBox [65] oder KVM [63], und für Docker mitbringt. Diese Liste kann zur Überprüfung auf eine virtualisierte Ausführungsumgebung verwendet werden.

[1] Der Vorgang zur Erstellung einer ausführbaren Version eines Programms.

5.2 Architektur und Schnittstellen

Die prototypische Implementierung besteht aus mehreren Teil-Projekten bzw. Komponenten. Dies ist zum einen dadurch Begründet, dass zwei verschiedene Programmiersprachen verwendet werden, zum anderen auf die logische Trennung von prüfender Seite (Kontrollserver) und überprüfender Seite (MISK). Die Komponenten sind eine Basiskomponente, im Folgenden MISK (Mechanismus zur eindeutigen Identifizierung von Softwarekomponenten) genannt, ein Intel SGX Prüf-Programm, ein Intel SGX Enklaven-Programm und der Kontrollserver. Die Unterteilung ist in Abbildung 5.1 dargestellt.

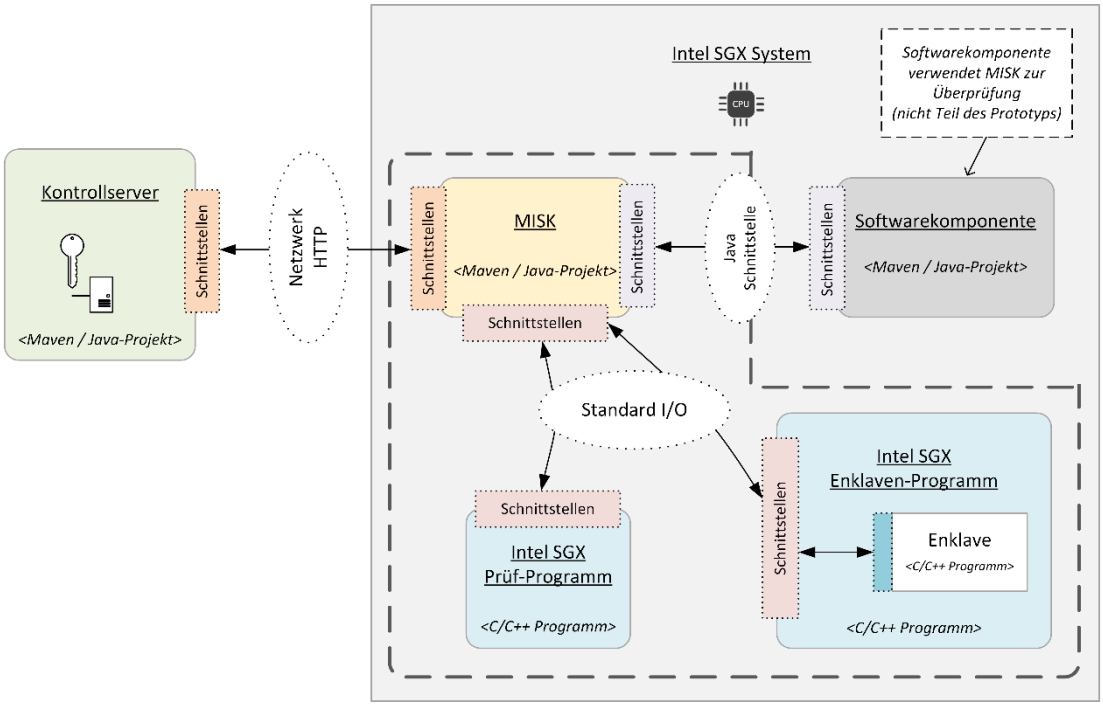

Abbildung 5.1: Architektur der prototypischen Implementierung des Konzepts

Die Basiskomponente MISK, welche als Java-/Maven-Projekt umgesetzt wurde, hat Schnittstellen zu allen anderen beteiligten Komponenten. Die Schnittstelle zur Softwarekomponente, die MISK zur Überprüfung verwendet, ist als Java-Methodenaufruf implementiert. Die Kommunikation zwischen MISK und dem Kontrollserver findet über HTTP-Aufrufe statt. Der Kontrollserver definiert diese Schnittstellen als Webservice, die dann von MISK aufgerufen werden.

Die Schnittstellen zwischen MISK und den beiden in C/C++ implementierten Komponenten, dem Intel SGX Prüf-Programm und dem Intel SGX Enklaven-Programm, sind über Standard

Ein- und Ausgabe realisiert. Dafür wird die ausführbare Datei[2] aus MISK heraus mit einem Start-Parameter aufgerufen. Der Start-Parameter definiert die Aktion die vom C/C++ Programm ausgeführt werden soll. Über den Standard Ausgabestrom (OutputStream) kann MISK das Ergebnis bzw. den Rückgabewert des C/C++ Programms einlesen. Die C/C++ Programme stattdessen über das Java Native Interface (JNI) einzubinden, zeigte sich als schwierig, da Intel SGX einige spezielle Bibliotheken und Datentypen, wie beispielsweise die der Intel SGX Trusted Cryptography Library [46], benötigt.

Wie in Abschnitt 3.2.1 genauer beschrieben, bestehen Intel SGX Enklaven Projekte immer aus zwei Teilprojekten: Eins, das die Enklave selbst implementiert und eins, das dazu benötigt wird die Schnittstellenfunktion (ECALLs und OCALLs) der Enklave aufzurufen, die sogenannte Enklaven Anwendung. Da eine Enklave selbst keine direkten Ein- und Ausgaben machen kann, müssen alle Schnittstellen der Enklave in einer speziellen EDL-Datei deklariert werden. Die ECALLs definieren Funktionen der Enklave und müssen daher in der Enklave implementiert werden. Die OCALLs definieren Funktionen aus der Enklave heraus und müssen daher in der Enklaven Anwendung implementiert werden. In Abbildung 5.1 sind die Enklave und die Enklaven Anwendung als „Intel SGX Enklaven-Programm“ zusammengefasst.

Zwischen der Enklaven Anwendung und der Enklave wurden für den Prototyp drei ECALLs und ein OCALL definiert. Die ECALLs sind zur Initiierung des DHS-Protokolls vom Kontrollserver (angestoßen durch den Kontrollserver bei einer Registrierungsanfrage), zur Initiierung des DHS-Protokolls durch MISK (angestoßen durch MISK bei einer Identifizierungsanfrage) und zur Übergabe des DHS-Parameters B (bei einer Identifizierungsanfrage). Die OCALL-Funktion ist zur Ausgabe der Ergebnisse bzw. Rückgabewerte aus der Enklave. In der Enklaven Anwendung wird in der Implementierung der OCALL-Funktion die Ausgabefunktion *printf()* mit dem Rückgabewert aus der Enklave aufgerufen und kann dann von einem anderen Programm über den Standard Ausgabestrom der C/C++ Anwendung eingelesen werden.

5.3 Prüfverfahren und Variantenauswahl

Das in Abschnitt 4.4 eingeführte Prüfverfahren, welches das System auf verschiedene Eigenschaften hin überprüft, wie z.B. ob Intel SGX verfügbar ist oder, ob es sich um eine virtualisierte Umgebung handelt, wird durch die Basiskomponente MISK und dem Intel SGX Prüf-Programm durchgeführt. Die einzelnen Schritte des Prüfverfahrens und die jeweils resultierende Variante, sind in einer Übersicht in Abbildung 4.2 dargestellt.

[2] Die prototypische Implementierung wurde mit Windows als Betriebssystem entwickelt, weswegen die ausführbare Datei eine EXE-Datei (Executable-Datei) ist.

Das Prüfverfahren dient zur Ermittlung, welche der vier Varianten für den MISK-Identifikator auf einem System verwendet werden soll. Zur Einleitung des Prüfverfahrens wird die Basiskomponente MISK von einer Softwarekomponenten-Instanz mit einer SW-ID als Parameter instanziiert. Bei der Instanziierung wird eine SWKP-Nonce durch MISK generiert. Damit eine SWKP-Nonce eindeutig ist und keine Wiederholungen aufweist, werden zur Generierung mehrere Faktoren verwendet. Zur Eindeutigkeit innerhalb einer Java Virtual Machine (JVM) wird der Hash-Code der MISK-Instanz und die der Nonce-Instanz selbst verwendet. Jedes Objekt in einer JVM wird über diesen Hash-Code eindeutig identifiziert, weswegen diese Eigenschaft mit verwendet wird [83]. Des Weiteren wird über die Klasse *java.util.Random* eine Pseudozufallszahl [84] zwischen eins und einer Millionen generiert. Zusätzlich wird die aktuelle Systemzeit in Millisekunden verwendet, um dadurch einen sich stetig ändernde Wert mit einzubeziehen. Über die einzelnen Werte wird eine SHA-256 Hash-Funktion (Apache DigestUtils [85]) angewendet und die Ergebnisse werden konkateniert. Zur Vermeidung, dass jede Nonce eine Länge von 1024 Bit (viermal 256 Bit) hat, wird über das konkatenierte Ergebnis nochmal dieselbe Hash-Funktion angewendet, um so eine konstante Länge von 256 Bit für die Nonce-Werte zu erreichen. Die Eindeutigkeit der Nonce-Werte wurde durch mehrere Testläufe[3] validiert, wodurch auf einem System nach über dreißig Millionen Generierungen[4] keine Kollision gefunden wurde.

Die MISK-Instanz kann nun von der Softwarekomponenten-Instanz zur Durchführung der Identifizierung verwendet werden. Die Identifizierung wird über einen Methodenaufruf gestartet. Hierdurch wird das Prüfverfahren zur Variantenauswahl eingeleitet. Zur Prüfung, ob Intel SGX Verfügbar ist, wird das Intel SGX Prüf-Programm aufgerufen. Dieses überprüft mittels von Intel SGX mitgelieferten Funktionen, ob auf dem Gerät Intel SGX verwendet werden kann. Dafür müssen mehrere Bedingungen überprüft werden. Zunächst wird überprüft, ob das Intel SGX Platform Software Package (PSW) installiert ist. Hierfür wird geprüft, ob spezielle DLL (Dynamic Link Library) auf dem System vorhanden sind, welche mit dem PSW mit installiert werden. Das PSW liefert die nötigen Funktionen und Bibliotheken zur Ausführung eines Intel SGX Programms mit. Anschließend kann über eine Intel SGX Funktion Anhand der CPUID festgestellt werden, ob der Prozessor Intel SGX unterstützt. Im nächsten Schritt wird geprüft, ob Intel SGX im BIOS aktiviert ist. Auch dies wird über eine Funktion des PSW durchgeführt, da eine solche Abfrage zur Prüfung auf BIOS Einstellungen ansonsten aus einem Standard Prozess nicht möglich ist. Das Ergebnis wird vom Intel SGX Prüf-Programm zurückgeliefert. Ist Intel SGX verfügbar, wird dies durch einen auf beiden Seiten fest definierten Wertes (Konstante) angegeben: *MESSAGE_SGX_SUPPORTED*. Für alle anderen Zustände sind ebenfalls fest definierte Ausgaben implementiert, wie z.B., dass das PSW nicht

[3] Es wurden über hundert Testläufe mit der finalen Implementierung durchgeführt.

[4] Die Zahl von ca. dreiunddreißig- fünfunddreißig Millionen generierten Nonce-Werten, ließ sich auf dem Test-System in performanter Zeit berechnen, ohne dass Mechanismen zur externen Auslagerung der gespeicherten Nonce-Werte verwendet werden mussten.

installiert ist: *MESSAGE_SGX_PSW_NOT_INSTALLED*. Das Intel SGX Prüf-Programm benötigt zur Überprüfung keine Enklave. Alle Funktionen können direkt in einer C/C++ Anwendung aufgerufen werden.

Ist Intel SGX verfügbar, wird Variante 1 für den MISK-Identifikator gewählt. Andernfalls wird im nächsten Schritt überprüft, ob es sich bei dem aktuellen Gerät, um ein bekanntes Gerät handelt, bei dem eine Geräte-Seriennummer vorhanden ist. Dafür werden der Geräteherstellers und die Modellbezeichnung mithilfe der OSHI-Bibliothek ausgelesen und mit einer statischen Liste abgeglichen. Ist das Gerät bekannt, wird die Geräte-Seriennummer ausgelesen und als Geräte-ID für den MISK-Identifikator verwendet. Zuvor wird über die Geräte-Seriennummer eine SHA-256 Hash-Funktion angewendet, wodurch zum einen ein einheitliches Format für die Geräte-ID erzielt wird, zum anderen die Geräte-Seriennummer nicht im Klartext auf dem Kontrollserver gespeichert wird.

Wurde Variante 2 nicht gewählt, wird nun überprüft, ob es sich bei dem aktuellen System um ein virtualisiertes System handelt. Hierfür werden mithilfe der OSHI-Bibliothek die MAC-Adressen und die Modellbezeichnung, welche für virtuelle Umgebungen oft dem Namen des Herstellers entsprechen, z.B. „VMWare“ oder „Microsoft Virtual PC“, überprüft. Wurde keine virtualisierte Umgebung erkannt, wird im nächsten Schritt zusätzlich geprüft, ob entweder die Geräte-Seriennummer oder Hauptplatinen-Seriennummer verfügbar ist und mit einem gültigen Wert belegt sind. Bei Docker ist die Geräte-Seriennummer beispielsweise mit „0“ angegeben. Diese zusätzliche Prüfung ist notwendig, da es keine Garantie auf Vollständigkeit für die Listen mit MAC-Adressen und Modellbezeichnungen gibt und diese sich zukünftig ändern können. Zur zuverlässigen Identifizierung eines Geräts, wird jedoch mindestens ein, als zuverlässig betrachteter, Hardware-Identifikator benötigt (siehe Abschnitt 4.5.3).

Wurde eine virtualisierte Umgebung erkannt, bzw. waren nicht alle notwendigen Hardware-Identifikatoren auslesbar, wird Varianten 4 gewählt, andernfalls kommt Variante 3 zum Einsatz. Für Variante 3 werden über die OSHI-Bibliothek die Geräte-Seriennummer, die Hauptplatinen-Seriennummer, die Prozessor-Seriennummer und die Modellbezeichnung ausgelesen. Über die einzelnen Werte wird eine SHA-256 Hash-Funktion angewendet. Damit die Geräte-ID dadurch keine Länge von 1024 Bit bekommt, wird auf jeden Wert nochmal die Java-Funktion *String.format()* angewendet, um die einzelnen Werte von 256 Bit auf 32 Bit zu kürzen. Die *String.format()* Funktion hat den Nachteil, dass das Ergebnis auf die Eingabe zurückzuführen ist. Für die Eingabe „1“ ist das Ergebnis „31“ und für „2“ ist das Ergebnis „32“, was der hexadezimalen Repräsentation in der ASCII-Zeichentabelle entspricht. Dies ist jedoch an dieser Stelle unproblematisch, da zuvor eine sichere Hash-Funktion angewendet wurde, die einen solchen Rückschluss auf die ursprüngliche Eingabe nicht zulässt. Die so gekürzten Werte werden als Geräte-ID für den MISK-Identifikator verwendet. Für Variante 3 hat die Geräte-ID eine konstante Länge von 152 Bit (dreimal 8 Bit kommen für ein zusätzliches Trennzeichen zwischen den einzelnen Werten hinzu).

Das Ergebnis des Prüfverfahrens ist nun ein MISK-Identifikator, mit gesetzter SW-ID, der Variante, eine Geräte-ID für die Varianten 2 und 3 und der SWKP-Nonce. Der Ablauf der

beschriebenen Prüfung ist für eine Registrierungs- und Identifizierungsanfrage gleich. Diese Überprüfung findet vor jeder Anfrage an den Kontrollserver statt, um so eine Änderung der Ausführungsumgebung erkennen zu können. Der Unterschied zwischen einer Registrierungs- und Identifizierungsanfrage ist die, ob in der MISK-Instanz bereits eine Server-Nonce gespeichert ist. Ist keine Server-Nonce vorhanden, bleibt der Wert im MISK-Identifikator „null“.

5.4 Anfragenvalidierung

Die Aufgabe des Kontrollservers ist die Überprüfung des MISK-Identifikators, zur Feststellung, ob die enthaltene SW-ID von der anfragenden Instanz rechtmäßig verwendet wird. Die Regeln zur Überprüfung werden im Folgenden erläutert. Eine Übersicht der einzelnen Schritte, die der Kontrollserver für die Anfragenvalidierung durchführt, ist in Abbildung 5.2 zu sehen.

Der MISK-Identifikator ist das Ergebnis des Prüfverfahrens, durch welches die Parameter gesetzt wurden und die zu verwendete Variante gewählt wurde. Der MISK-Identifikator wird über einen HTTP-POST an den Kontrollserver gesendet. Für die Varianten 2 bis 4 wird dieselbe Schnittstellenfunktion des Kontrollservers verwendet, für Variante 1 wird der MISK-Identifikator über eine separate Schnittstellenfunktion an den Kontrollserver gesendet, da bei der Durchführung des DHS-Protokolls die Antwort vom Kontrollserver noch nicht die neue Server-Nonce enthält. Der Kontrollserver verwendet das MISK Maven-Projekt als Abhängigkeit und kann daher dieselben Datentypen und Funktionen, wie beispielsweise die Klasse für den MISK-Identifikator (*MiskIdentifier*) und die Funktion zur Nonce Erstellung, verwenden.

Empfängt der Kontrollserver eine Anfrage von einer MISK-Instanz, werden zunächst die Parameter SW-ID, Variante und SWKP-Nonce auf ihre Gültigkeit überprüft, d.h. die Werte müssen gesetzt sein und ein gültiges Format haben. Die Variante muss einer der vier als ENUM definierten Werte entsprechen und die SW-ID muss eine dem Kontrollserver bekannte sein, was in der Implementierung über eine statische Liste abgeprüft wird. Ist einer der Werte nicht gültig, erhält die MISK-Instanz eine Fehlermeldung als Antwort, mit dem HTTP-Status „*400 Bad Request*“. Diese Fehlermeldung wird immer dann gesendet, sobald eine Prüfung nicht dem erwarteten Ergebnis entspricht. Die genaue Fehlerursache zur Anzeige auf Seiten der MISK-Instanz wird in der Antwort gesetzt und ist für jeden Fehlerfall spezifisch.

Im nächsten Schritt wird die Variante geprüft. Ist die Variante 2 oder 3, muss auch die Geräte-ID gesetzt sein. Zur Unterscheidung von Registrierungs- und Identifizierungsanfrage, wird die Server-Nonce geprüft. Ist diese nicht gesetzt und die SW-ID wurde bisher noch nicht registriert, interpretiert der Kontrollserver die Anfrage als Registrierungsanfrage. Hierfür wird im nächsten Schritt zwischen Variante 1 und den Varianten 2 bis 4 unterschieden. Bei Variante 1 wird das dafür spezifische Verfahren mit dem DHS-Protokoll eingeleitet, wo als Ergebnis auf beiden Seiten die Server-Nonce generiert wird. Für die Varianten 2 bis 4 wird eine neue Nonce generiert, welche der MISK-Instanz als Antwort gesendet wird. Für alle Varianten gleich, wird der MISK-Identifikator mit der generierten Server-Nonce in einer Schlüssel-Wert-Paar Struktur gespeichert. Die SW-ID dient dabei als Schlüssel. Zusätzlich wird die SWKP-Nonce in

einer Liste, die ebenfalls einer SW-ID zugeordnet ist, gespeichert. Diese Liste dient zur Überprüfung der Historie der zuletzt verwendeten SWKP-Nonces für eine SW-ID.

Handelt es sich bei Anfrage um eine Identifizierungsanfrage, muss die verwendete SW-ID bereits registriert sein und ein gespeicherter MISK-Identifikator vorhanden sein. Der gespeicherte MISK-Identifikator wird nun mit dem der Anfrage abgeglichen. Für eine gültige Prüfung muss die Server-Nonce, die Variante und bei Variante 2 und 3, auch die Geräte-ID übereinstimmen. Die SWKP-Nonce muss hingegen nicht übereinstimmen, da sich diese durch einen Neustart ändert. Jedoch wird hier zusätzlich in der für die SW-ID gespeicherten Historie der zuletzt verwendeten SWKP-Nonces überprüft, dass kein Wechsel stattgefunden hat. Ist die SWKP-Nonce gleich wie die zuvor verwendete, ist das valide. Sind die beiden Werte unterschiedlich, darf die bei der aktuellen Anfrage verwendet SWKP-Nonce nicht in der Historie auftauchen. Für die prototypische Implementierung wurde die Länge der Historie aus Gründen der Performanz auf 100 gesetzt.

Sind alle Werte der Identifizierungsanfrage gültig, wird nun für Variante 1 wieder das DHS-Protokoll zur Festlegung der nächsten Server-Nonce durchgeführt. Für die Varianten 2 bis 4 wird eine neue Server-Nonce generiert und der MISK-Instanz in der Antwort zurückgesendet. Wieder für alle Varianten gleich, wird die SWKP-Nonce Historie aktualisiert und die neue Server-Nonce, die für die nächste Anfrage gültig ist, abgespeichert.

Der in Abschnitt 4.6 beschriebene zeitbasierte Gültigkeitsmechanismus für die Server-Nonce, wurde Aufgrund der beschriebenen Einschränkungen in Bezug auf die zeitliche Abhängigkeit und den Eingriff in die Ausführung der Softwarekomponenten-Instanz in der prototypischen Implementierung nicht für die Anfragenvalidierung verwendet.

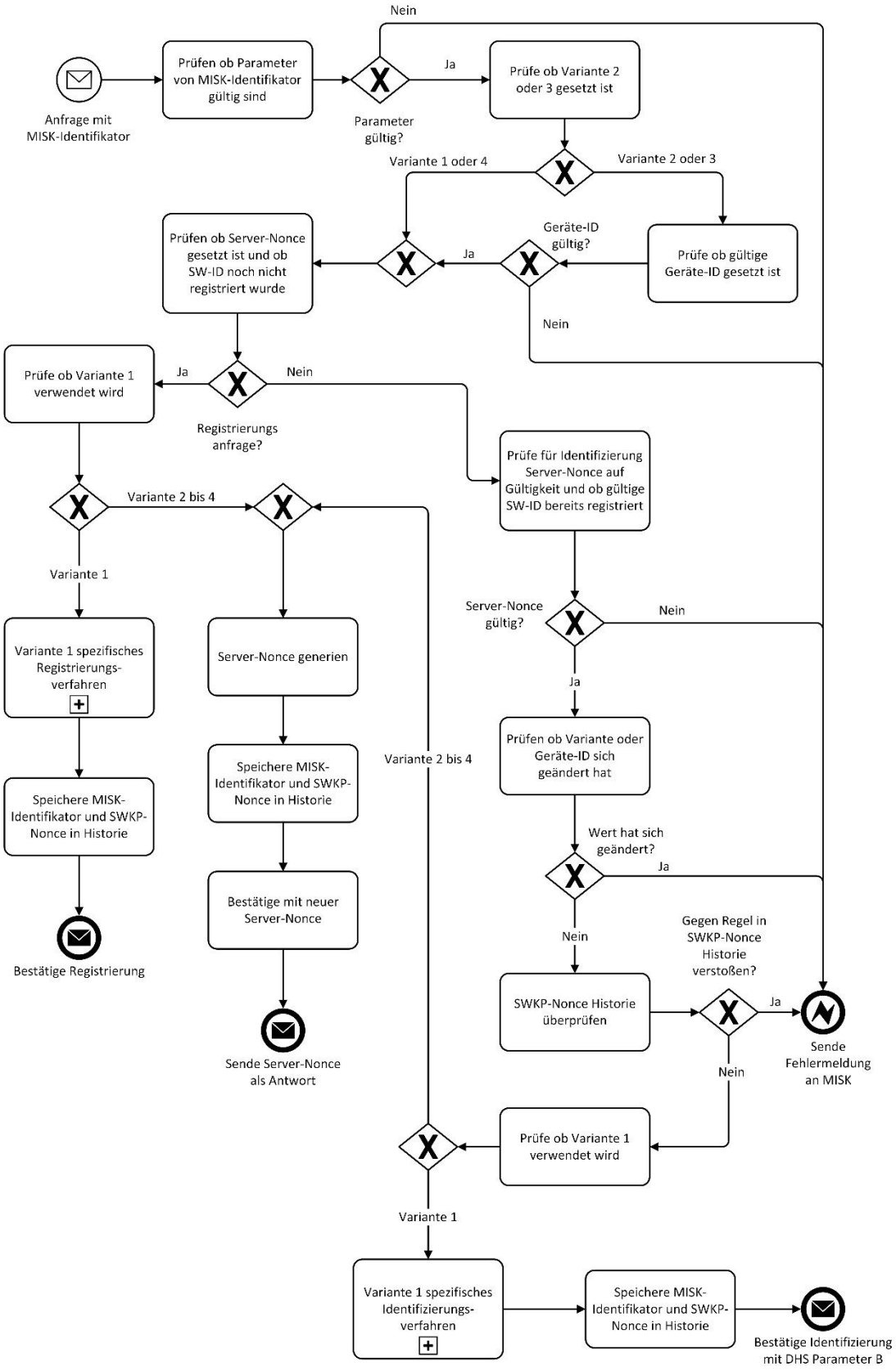

Abbildung 5.2: Schritte der Anfragenvalidierung auf Seite des Kontrollservers

6 Evaluierung

In Kapitel 4 wurden die Anforderungen und das Konzept beschrieben. Anschließend wurde in Kapitel 5 eine prototypische Implementierung zum Konzept vorgestellt. In diesem Kapitel wird zunächst überprüft, ob alle Möglichkeiten zur mehrfachen Ausführung einer Softwarekomponente mit einer SW-ID unterbunden werden können. Im Anschluss werden die Anforderung und das Konzept gegenübergestellt und eine festgestellte Schwachstelle wird abschließend nochmals kritisch betrachtet.

6.1 Überprüfung der Möglichkeiten zur mehrfachen Ausführung

In Abschnitt 4.1 wurden zwei Möglichkeiten in Betracht gezogen, wie ein Angreifer vorgehen kann, um eine Softwarekomponente mit einer SW-ID mehrfach auszuführen. An dieser Stelle werden die beiden Möglichkeiten nochmals betrachtet und überprüft, wie diese nun durch MISK unterbunden werden können.

Möglichkeit 1

Der Angreifer hat eine zur Ausführung notwendige SW-ID im Klartext, die bereits durch eine andere laufende Instanz der Softwarekomponente verwendet wird. Diese wird nun zur Ausführung einer weiteren Instanz verwendet.

Vermeidung von Möglichkeit 1: Diese Möglichkeit wird durch die Verwendung der Server-Nonce verhindert. Die SW-ID allein reicht nicht aus, um eine Softwarekomponente mit einer bereits registrierten SW-ID auszuführen. Zusätzlich verhindert bei Variante 1 der Mechanismus des Intel SGX Sealing, dass die Daten auf einem anderen Gerät verwendet werden können. Für Variante 2 und 3 wird die Geräte-ID verwendet, die eine SW-ID durch die Registrierung an ein Gerät bindet. Die Implementierung hat gezeigt, dass der Kontrollserver die Möglichkeit 1 zuverlässig erkennt.

Möglichkeit 2

Der Angreifer hat eine Kopie einer bereits instanziierten Softwarekomponente mit hinterlegter SW-ID. Eine solche Kopie kann beispielsweise das Abbild (Image) einer virtuellen Maschine sein.

Vermeidung von Möglichkeit 2: Bei dieser Möglichkeit werden alle gespeicherten Daten kopiert, auch die im MISK-Identifikator enthaltene SW-ID und die Server-Nonce. Die anderen Parameter wie Variante, Geräte-ID und SWKP-Nonce werden bei

jedem Start neu durch das Prüfverfahren gesetzt. Soll die Kopie auf einem anderen Gerät ausgeführt werden, ist dies nur bei Variante 4 möglich und dies auch nur dann, wenn auch die ursprüngliche Instanz die SW-ID mit Variante 4 registriert hatte. Hat das Prüfverfahren beim Start der Kopie dieselbe Variante als Ergebnis, kann die Kopie erfolgreich ausgeführt werden, ohne dass dies der Kontrollserver erkennt, da eine neue SWKP-Nonce grundsätzlich als Neustart einer Softwarekomponenten-Instanz betrachtet wird. Wird zusätzlich das in Abschnitt 4.6 vorgestellte zeitbasierte Gültigkeitsverfahren für die Server-Nonce verwendet, kann die Kopie nur in dem Zeitfenster von t_{renew} gestartet werden und die durch t_{valid} definierte Gültigkeitsdauer darf noch nicht verstrichen sein. Auch ohne die zeitbasierte Gültigkeit gilt für jede Kopie, dass diese nur solange erfolgreich gestartet werden kann, solange in der Zwischenzeit keine andere Identifizierungsanfrage an den Kontrollserver mit der kopierten SW-ID und Server-Nonce durchgeführt wurde, da ansonsten die Server-Nonce nicht mehr gültig ist.

In diesem Abschnitt konnte gezeigt werden, dass die im Systemmodell beschrieben Möglichkeiten, zur Umgehung der rechtmäßigen Ausführung, durch MISK grundsätzlich verhindert werden können. Möglichkeit 1 kann ganz unterbunden werden und Möglichkeit 2 ist nur unter bestimmten Voraussetzungen möglich. Jedoch wird durch das Ausführen der Kopie, die Server-Nonce, die von der ursprünglichen Instanz verwendet wurde, ungültig.

6.2 Anforderungsabgleich

Nachfolgend werden die in Abschnitt 4.2 aufgestellten Anforderungen auf ihre Erfüllung hin untersucht.

Anforderung 1

Gleichzeitig ausgeführte Instanzen von Softwarekomponenten mit derselben SW-ID sollen während des Betriebs erkannt werden.

Abgleich: Diese Anforderung konnte insoweit erfüllt werden, dass zum einen die Geräte-ID dazu verwendet kann, um zwei Instanzen einer Softwarekomponente mit derselben SW-ID auf zwei verschiedenen Geräten, auch als zwei unterschiedliche Instanzen identifiziert werden können. Zum anderen können zwei parallel ausgeführte Instanzen über die Eindeutigkeit der SWKP-Nonce unterschieden werden.

Anforderung 2

MISK soll in verschiedene Softwareprodukte integrierbar sein, die bereits über eine SW-ID, wie z.B. einem Lizenzschlüssel, verfügen.

Abgleich: MISK wurde als eigenständiger Mechanismus konzeptioniert. Durch die Implementierung als Java-/Maven-Projekt konnte gezeigt werden, dass MISK als eigenständiges Modul von anderen Softwarekomponenten verwendet werden kann. Die

SW-ID wird von Seiten der Softwarekomponente als Parameter übergeben und wird vom Kontrollserver auf ihre Gültigkeit hin überprüft.

Anforderung 3

Das Format der SW-ID für MISK soll variable sein und nicht von einem strikten Schema, wie z.B. einem 25 Zeichen langem Lizenzschlüssel, abhängig sein.

Abgleich: Konzeptionell kann für MISK jedes Format für eine SW-ID verwendet werden. In der Implementierung wird für die SW-ID eine Zeichenkette (String) verwendet und gibt somit keine Einschränkung an das Format der SW-ID vor.

Anforderung 4

MISK soll auch in virtualisierten Umgebungen funktionieren.

Abgleich: Durch das entwickelte Prüfverfahren und die verschiedenen Varianten, eine speziell zur Verwendung für virtualisierten Umgebungen, wird diese Anforderung erfüllt. Des Weiteren wurde die prototypische Implementierung in einer virtuellen Maschine, mit Oracle VirtualBox als Hypervisor und Ubuntu/Linux als Betriebssystem, erfolgreich getestet. Auch die Ausführung in einem Docker Container funktionierte wie erwartet.

Anforderung 5

Der Anwender einer Softwarekomponente mit MISK soll keine spezielle Hardware benötigen, wie beispielsweise einem USB-Dongle (Universal-Serial-Bus-Dongle). Handelsübliche Hardware (Commodity Hardware) im Bereich Workstation und Server, sollen für den Betrieb ausreichend sein.

Abgleich: Durch die prototypische Implementierung als Java-/Maven-Projekt und durch Berücksichtigung von verschiedenen Ausführungsumgebungen durch das Prüfverfahren, konnte gezeigt werden, dass MISK auf jedem Gerät auf dem eine JVM läuft, ausgeführt werden kann. Aufgrund dessen, dass durch das Prüfverfahren verschiedene Varianten zur Ausführung zur Verfügung stehen, ließe sich auch eine spezielle Hardware, wie beispielsweise ein USB-Dongle, als weitere Variante in das Konzept integrieren, ohne diese Anforderung nicht zu erfüllen.

Anforderung 6

Die Funktion von MISK soll nicht abhängig von einer speziellen Sicherheitstechnologie sein.

c) Unterstützt das Gerät eine bestimmte Sicherheitstechnologie, kann diese von MISK eingesetzt werden.

d) Unterstützt das Gerät diese Sicherheitstechnologie nicht, bzw. ist diese deaktiviert, soll MISK dennoch funktionieren.

Abgleich: Durch den Einsatz der verschiedenen Varianten, kann MISK auch auf Geräten ohne Intel SGX verwendet werden. Zur Überprüfung wurde der implementierte Prototyp auf einem Gerät mit und ohne Intel SGX erfolgreich ausgeführt.

Anforderung 7

Als Betriebssysteme sollen sowohl Windows- als auch Linux-Betriebssysteme unterstützt werden.

Abgleich: Die gewählte Sicherheitstechnologie Intel SGX ist sowohl auf Linux als auch auf Windows Systemen verwendbar [48]. Auch die für die Implementierung gewählte Programmiersprache Java, macht diesbezüglich keine Einschränkungen. Wie beim **Abgleich** zu **Anforderung 4** beschrieben, wurde der Prototyp erfolgreich auf verschiedenen Systemen getestet, darunter auch ein Windows (Windows 10) und Linux (Ubuntu 16.04) Betriebssystem.

Anforderung 8

MISK soll auch ohne Verbindung zum Kontrollserver funktionieren, jedoch unter bestimmten Einschränkungen (z.B. nur für einen gewissen Zeitraum oder mit eingeschränkter Funktionalität).

Abgleich: Konzeptionell gibt MISK kein Verhalten für die Softwarekomponente vor, was bei einer fehlenden Verbindung zum Kontrollserver geschieht, nur das eine Identifizierungsanfrage dann nicht erfolgreich durchgeführt werden kann. In der Implementierung wurde dieser Fall berücksichtigt und MISK liefert als Ergebnis an die Softwarekomponenten-Instanz zurück, dass die Identifizierungsanfrage fehlgeschlagen ist. Welchen Einfluss das auf die Funktion der Softwarekomponenten-Instanz hat, muss jeweils auf der Seite der Softwarekomponente entschieden werden.

Anforderung 9

Die Verarbeitungsgeschwindigkeit (Performanz) der Softwarekomponente soll durch MISK nicht signifikant verschlechtert werden.

Abgleich: Für den Einsatz von MISK wurden zwei verschiedene Szenarien entwickelt: Eins zur direkten Identifizierung beim Kontrollserver und eins zur Verwendung als Identifizierungsservice für andere Anwendungen (siehe Abbildung 4.4). Das erste Szenario benötigt dabei nur einmalig beim Start eine Interaktion mit dem Kontrollserver, wodurch eine Einschränkung während des Betriebs der Softwarekomponenten-Instanz verhindert wird.

In diesem Abschnitt wurde durch den Abgleich von Konzept und Anforderungen gezeigt, dass die Anforderungen alle erfüllt werden konnte.

6.3 Diskussion

Bisher konnte in diesem Kapitel gezeigt werden, dass die Möglichkeiten zur Ausführung einer Softwarekomponenten-Instanz mit einer bereits verwendeten SW-ID in fast allen Fällen unterbunden werden kann und dass die Anforderungen durch das entwickelte Konzept erfüllt wurden. Die in Kapitel 1 gestellte Forschungsfrage, wie ein Mechanismus zur Erkennung mehrfach instanziierte Software mit gleicher SW-ID konzeptioniert sein muss, konnte ebenfalls durch den im Konzept vorgestellten Mechanismus zur eindeutigen Identifizierung von Softwarekomponenten (MISK) beantwortet werden.

Ein wichtiges Prinzip, das beim Entwurf des Konzept beachtet wurde, ist das Kerckhoffs Prinzip, oft auch als "security through obscurity" bekannt [56]. Dieses Prinzip sagt aus, dass die Sicherheit eines Verfahrens nicht von dessen Geheimhaltung abhängig sein darf, sondern lediglich durch die Geheimhaltung eines Schlüssels. Bei dem Konzept von MISK ist dieser Schlüssel der MISK-Identifikator, im speziellen die Server-Nonce. Das Verfahren bzw. Protokoll von MISK kann öffentlich bekannt. Durch den Einsatz von hardwarebasierter Sicherheitstechnologien, hier Intel SGX, kann die Geheimhaltung der Informationen zusätzlich geschützt werden.

Eine Angriffsmöglichkeit die bei Verwendung des DHS-Protokolls zu beachten ist, ist der Man-In-The-Middle-Angriff. Dieser wurde in Abschnitt 2.2 bereits in Zusammenhang mit dem DHS-Protokoll genauer beschrieben. Das DHS-Protokoll wird nur für die Variante 1 verwendet, um die Softwarekomponenten-Instanz als unsicheren Kanal auszuschließen. Bei Variante 1 wird nur die Enklave und der Kontrollserver als sichere Ausführungsumgebung betrachtet. Wie im Systemmodell beschrieben, wird für die Netzverbindung zwischen Softwarekomponente und Kontrollserver angenommen, dass diese sicher ist, z.B. durch Verwendung von TLS. Daher besteht die Möglichkeit für einen Man-In-The-Middle-Angriff prinzipiell nur zwischen der Softwarekomponente, die MISK verwendet, und der Enklave. Da die Enklave keine direkten Ein- und Ausgaben machen kann, muss diese über Schnittstellen mit der unsicheren Seite kommunizieren. Bei einem Verbindungsaufbau zum Kontrollserver, kann dessen Authentizität leicht über das für TLS verwendete Zertifikat von der Enklave überprüft werden, beispielsweise durch ein Challenge-Response-Verfahren. Die andere Richtung ist hingegen deutlich schwerer, da hierfür ein Zertifikat für die Enklave benötigt wird. Das Zertifikat müsste für jede Enklave einer Softwarekomponenten-Instanz ein anderes sein. Hierbei entsteht jedoch wieder die Problematik, dass auch ein Zertifikat, wie die per Software definierten Identifikatoren, leicht kopiert werden kann. Des Weiteren müssten alle vergebenen Zertifikate über eine Zertifizierungsstelle (Certification Authority (CA)) verwaltet werden. Ein Man-In-The-Middle-Angriff im Zusammenhang mit dem verwendeten DHS-Protokoll, ist somit bei der Verwendung von MISK prinzipiell möglich und sollte daher als Angriffsmöglichkeit und somit als potenzielle Schwachstelle beachtet werden.

7 Zusammenfassung und Ausblick

Generell findet Identifizierung über spezielle und eindeutige Eigenschaften statt. Personen können über einen Fingerabdruck oder ein biometrisches Bild eindeutig identifiziert werden, Geräte über eine Seriennummer oder andere fest verbaute Hardwarekomponenten. Softwarekomponenten können ebenfalls einen eindeutigen per Software definierten Identifikator (SW-ID) haben, wie beispielsweise einen Lizenzschlüssel. Im Gegensatz zu den zuvor genannten Identifikatoren, kann eine SW-ID leicht kopiert werden. Eine Unterscheidung der Softwarekomponenten ist dann ohne einen weiteren Mechanismus nicht mehr möglich. Zur Kontrolle der Einhaltung von Lizenzbedingungen und Nutzungsrechten ist eine Unterscheidung jedoch notwendig. Ziel dieser Arbeit war es daher einen Mechanismus zur eindeutigen Identifizierung von Softwarekomponenten (MISK) zu entwerfen, um die mehrfache Ausführung von Instanzen einer Softwarekomponente mit einer kopierten bzw. mehrfach verwendeten SW-ID zu erkennen. Des Weiteren sollte untersucht werden, ob sich eine hardwarebasierte Sicherheitstechnologien zur Lösung des Problems verwenden lässt.

Die Recherche zum Stand der Technik hat gezeigt, dass das Identifizierungsproblem im IoT-Umfeld zu den offenen Forschungsproblemen gehört und es bisher keine allgemein anwendbare Lösung gibt. Die meisten Lösungsansätze setzen darauf, dass die Software über eindeutige Merkmale des Geräts, auf dem diese ausgeführt wird, identifiziert werden kann, oder aber darauf, dass eine vergebene SW-ID zur eindeutigen Identifikation nicht weitergegeben wird. Für das Konzept wurden beide Lösungsansätze weiterverfolgt. Sind eindeutige hardwarebasierte Merkmale eines Geräts vorhanden, werden diese mit zur Identifizierung verwendet. Die gewählte Sicherheitstechnologie Intel SGX wird dafür eingesetzt, dass die zur Identifizierung notwendigen Daten nicht weitergeben werden können. Damit die Ausführung einer Softwarekomponente jedoch auch auf Geräten ohne Intel SGX und ohne Zugriff auf bestimmte Hardwaremerkmale möglich ist, wurde ein mehrstufiges Prüfverfahren entwickelt, dass die verschiedenen Systembedingungen erkennt und so eine Ausführung auch in virtualisierten Umgebungen ermöglicht. Durch die prototypische Implementierung des Konzepts konnte gezeigt werden, dass sich der entwickelte Mechanismus als Modul in bestehende Softwarekomponenten integrieren lässt und die gestellten Anforderungen erfüllt werden konnten. Die parallele Ausführung mehrerer Instanzen einer Softwarekomponente mit derselben SW-ID kann durch MISK zuverlässig erkannt werden, auch wenn nicht in jeder Situation entscheidbar ist, welches die originale Instanz und welche die Instanz einer Kopie ist.

7.1 Ausblick

Der in dieser Arbeit vorgestellte Mechanismus zur eindeutigen Identifizierung von Softwarekomponenten (MISK) verwendet für Variante 1 (Sicherheitstechnologie) das DHS-Protokoll zum sicheren Austausch der Server-Nonce. In Abschnitt 6.3 wurde auf die Möglichkeit zur Durchführung eines Man-In-The-Middle-Angriffs auf das DHS-Protokoll hingewiesen. Zur Vermeidung dieser Angriffsmöglichkeit auf das DHS-Protokoll, wird ein sicheres Verfahren zur Feststellung der Authentizität beider Seiten, der Enklave und dem Kontrollserver, benötigt, z.B. durch Verwendung von Zertifikaten. Dabei muss jedoch berücksichtigt werden, wie dass Zertifikat sicher in den Speicher der Enklave gelangen kann. Soll ein Zertifikat erst nach der Auslieferung an die Enklave gesendet werden, geht dies nur über denselben ungesicherten Weg, über den auch das DHS-Protokoll durchgeführt wird. Wird hingegen jede Enklave bereits mit einem Zertifikat ausgeliefert, würde dies einen enormen Verwaltungsaufwand bedeuten, da jede Enklave eine andere kryptographische Prüfsumme hätte, diese jedoch für den Attestierungsvorgang dem Kontrollserver bekannt sein muss. Hierzu könnte untersucht werden, ob ein TPM zur sicheren Verwaltung der Zertifikate verwendet werden kann und wie sich diese Erweiterung in das Konzept integrieren lässt.

Als hardwarebasierte Sicherheitstechnologie wurde bisher nur Intel SGX im Konzept berücksichtigt. Dies hat den Nachteil, dass eine sichere Ausführungsumgebung (TEE) nur auf einer eingeschränkten Anzahl an Geräten zur Verfügung steht. Eine weitere Untersuchung könnte daher in die Richtung gehen, auch andere hardwarebasierte Sicherheitstechnologien die eine TEE bereitstellen, wie beispielsweise ARM TrustZone, auf die Verwendbarkeit hin zu überprüfen, um dadurch auf einer größeren Anzahl von Geräten eine sichere Ausführungsumgebung unterstützen zu können. Aber auch Sicherheitstechnologien die kein TEE implementieren könnten in Betracht gezogen werden. Beispielsweise könnte untersucht werden, ob ein TPM zur Verwaltung der Server-Nonce verwendet werden kann oder sich anderweitig zu Identifizierungszwecken einsetzen lässt. Die Verwendung von TPMs hätte den großen Vorteil, dass eine Vielzahl an Geräten unterstützt werden könnte, da diese mittlerweile in fast allen Geräten verbaut sind.

Weiterhin könnten auch PUFs als zusätzliche Variante in das Konzept integriert werden. Die Verwendung einer PUF hätte den Vorteil, dass diese, z.B. in Form eines USB-Dongles, auf verschiedenen Geräten verwendet werden kann und somit die Migration einer Softwarekomponente von einem Gerät auf ein anderes leicht durchzuführen ist. Die PUF könnte den Lizenzschlüssel bzw. die SW-ID enthalten oder zur Generierung eines eindeutigen Schlüssels verwendet werden. Hierzu müsste jedoch zusätzlich ein Mechanismus eingesetzt werden, der die mehrfache Ausführung von Softwarekomponenten-Instanzen auf einem Gerät unter Verwendung derselben PUF erkennt bzw. verhindert.

Literatur

[1] Mario Hermann, Tobias Pentek und Boris Otto, „Design Principles for Industrie 4.0 Scenarios: A Literature Review“.

[2] BSA The Software Alliance, *2018 BSA GLOBAL SOFTWARE SURVEY: Software Management: Security Imperative, Business Opportunity.* [Online] Verfügbar unter: https://gss.bsa.org/wp-content/uploads/2018/05/2018_BSA_GSS_Report_en.pdf.

[3] V. Lohan und R. P. Singh, „Research challenges for Internet of Things: A review“ in *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*, Gurgaon, Okt. 2017 - Okt. 2017, S. 109–117.

[4] Z. Ning, F. Zhang, W. Shi und W. Shi, „Position Paper: Challenges Towards Securing Hardware-assisted Execution Environments“ in *Proceedings of the Hardware and Architectural Support for Security and Privacy*, Toronto, ON, Canada, 2017, S. 1–8.

[5] B. McGillion, T. Dettenborn, T. Nyman und N. Asokan, „Open-TEE -- An Open Virtual Trusted Execution Environment“ in *2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland, Aug. 2015 - Aug. 2015, S. 400–407.

[6] B. Ngabonziza, D. Martin, A. Bailey, H. Cho und S. Martin, „TrustZone Explained: Architectural Features and Use Cases“ in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, Pittsburgh, PA, USA, Nov. 2016 - Nov. 2016, S. 445–451.

[7] J.-E. Ekberg, K. Kostiainen und N. Asokan, „The Untapped Potential of Trusted Execution Environments on Mobile Devices“, *IEEE Secur. Privacy*, Jg. 12, Nr. 4, S. 29–37, 2014.

[8] M. Sabt, M. Achemlal und A. Bouabdallah, „Trusted Execution Environment: What It is, and What It is Not“ in *2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland, Aug. 2015 - Aug. 2015, S. 57–64.

[9] GlobalPlatform, *Trusted Execution Environment (TEE).* [Online] Verfügbar unter: https://globalplatform.org/technical-committees/trusted-execution-environment-tee-committee/.

[10] P. Maene *et al.*, „Hardware-Based Trusted Computing Architectures for Isolation and Attestation“, *IEEE Trans. Comput.*, Jg. 67, Nr. 3, S. 361–374, 2018.

[11] S. Pinto und N. Santos, „Demystifying Arm TrustZone“, *ACM Comput. Surv.*, Jg. 51, Nr. 6, S. 1–36, 2019.

[12] A. Beutelspacher, J. Schwenk und K.-D. Wolfenstetter, Hg., *Moderne Verfahren der Kryptographie*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015.

[13] S. Wendzel, *IT-Sicherheit für TCP/IP- und IoT-Netzwerke*. Wiesbaden: Springer Fachmedien Wiesbaden, 2018.

[14] W. Diffie und M. Hellman, „New directions in cryptography“, *IEEE Trans. Inform. Theory*, Jg. 22, Nr. 6, S. 644–654, 1976.

[15] J. Buchmann, *Einführung in die Kryptographie*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.

[16] K. Freiermuth, J. Hromkovič, L. Keller und B. Steffen, *Einführung in die Kryptologie*. Wiesbaden: Springer Fachmedien Wiesbaden, 2014.

[17] R. Plaga, „Was sind „Physical Unclonable Functions“ und welchen Zielen dienen sie?“, *Datenschutz Datensich*, Jg. 39, Nr. 4, S. 219–223, 2015.

[18] S. Tajik, *On the Physical Security of Physically Unclonable Functions*. Cham: Springer International Publishing, 2019.

[19] S. Bhunia, S. Ray und S. Sur-Kolay, *Fundamentals of IP and SoC Security*. Cham: Springer International Publishing, 2017.

[20] A. Čolaković und M. Hadžialić, „Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues“, *Computer Networks*, Jg. 144, S. 17–39, 2018.

[21] H. Aftab *et al.*, „Analysis of identifiers in IoT platforms“, *Digital Communications and Networks*, http://dx.doi.org/10.1016/j.dcan.2019.05.003, 2019.

[22] K. Chih Chang, R. Nokhbeh Zaeem und K. S. Barber, „Internet of Things: Securing the Identity by Analyzing Ecosystem Models of Devices and Organizations“, *The 2018 AAAI Spring Symposium Series*, https://www.aaai.org/ocs/index.php/SSS/SSS18/paper/view/17491/15393.

[23] M. Alam, R. H. Nielsen und N. R. Prasad, „The evolution of M2M into IoT“ in *2013 First International Black Sea Conference on Communications and Networking (BlackSeaCom)*, Batumi, Georgia, Jul. 2013 - Jul. 2013, S. 112–115.

[24] L. Atzori, A. Iera und G. Morabito, „The Internet of Things: A survey“, *Computer Networks*, Jg. 54, Nr. 15, S. 2787–2805, 2010.

[25] S. Haller, „The Things in the Internet of Things“, https://www.researchgate.net/publication/228488111_The_Things_in_the_Internet_of_Things, 2010.

[26] M. Beltrán, „Identifying, authenticating and authorizing smart objects and end users to cloud services in Internet of Things“, *Computers & Security*, Jg. 77, S. 595–611, 2018.

[27] S. Chun, J. Jung, X. Jin, G. Cho und K.-H. Lee, „Poster Abstract: Semantically Enriched Object Identification for Internet of Things“ in *2014 IEEE International Conference on Distributed Computing in Sensor Systems*, Marina Del Rey, CA, USA, Mai. 2014 - Mai. 2014, S. 141–142.

[28] B. Ovilla-Martinez und L. Bossuet, „Restoration protocol: Lightweight and secur devices authentication based on PUF“ in *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Abu Dhabi, Okt. 2017 - Okt. 2017, S. 1–6.

[29] O. Salman, S. Abdallah, I. H. Elhajj, A. Chehab und A. Kayssi, „Identity-based authentication scheme for the Internet of Things“ in *2016 IEEE Symposium on Computers and Communication (ISCC)*, Messina, Italy, Jun. 2016 - Jun. 2016, S. 1109–1111.

[30] H. Noguchi, M. Kataoka und Y. Yamato, „Device Identification Based on Communication Analysis for the Internet of Things“, *IEEE Access*, Jg. 7, S. 52903–52912, 2019.

[31] A. Kaur, I. Batra und A. Bakshi, „An Intelligent Context Aware Based Access Control Framework to Prevent Attacker Nodes in Internet of Things“ in *2018 4th International Conference on Computing Sciences (ICCS)*, Jalandhar, Aug. 2018 - Aug. 2018, S. 218–227.

[32] D. Li, W. Peng, W. Deng und F. Gai, „A Blockchain-Based Authentication and Security Mechanism for IoT“ in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, Hangzhou, Jul. 2018 - Aug. 2018, S. 1–6.

[33] S. Huh, S. Cho und S. Kim, „Managing IoT devices using blockchain platform“ in *2017 19th International Conference on Advanced Communication Technology (ICACT)*, Pyeongchang, Kwangwoon Do, South Korea, Feb. 2017 - Feb. 2017, S. 464–467.

[34] P. Fremantle, B. Aziz und T. Kirkham, „Enhancing IoT Security and Privacy with Distributed Ledgers - A Position Paper“ in *IoTBDS 2017: Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security : Porto, Portugal, April 24-26, 2017*, Porto, Portugal, 2017, S. 344–349.

[35] *Alliance for Internet of Things Innovation (AIOTI).* [Online] Verfügbar unter: https://aioti.eu/.

[36] Alliance for Internet of Things Innovation, „Identifiers in Internet of Things (IoT)“, https://aioti.eu/wp-content/uploads/2018/03/AIOTI-Identifiers_in_IoT-1_0.pdf.pdf, 2018.

[37] T. Berners-Lee, „Uniform Resource Identifier (URI): Generic Syntax“, Jan. 2005. [Online] Verfügbar unter: https://tools.ietf.org/html/rfc3986.

[38] *das European Research Cluster On The Internet Of Things (IERC).* [Online] Verfügbar unter: http://www.internet-of-things-research.eu/.

[39] IERC - European Research Cluster On The Internet Of Things, *EU-China Joint White Paper on Internet-of-Things Identification.* [Online] Verfügbar unter: http://www.internet-of-things-research.eu/pdf/IERC_Position_Paper_EU-China_IoT_Identification_Final.pdf.

[40] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade und J. Del Cuvillo, „Using innovative instructions to create trustworthy software solutions“ in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy - HASP '13*, Tel-Aviv, Israel, 2013, S. 1.

[41] F. McKeen *et al.*, „Innovative instructions and software model for isolated execution“ in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy - HASP '13*, Tel-Aviv, Israel, 2013, S. 1.

[42] S. Brenner, T. Hundt, G. Mazzeo und R. Kapitza, „Secure Cloud Micro Services Using Intel SGX“ in *Lecture Notes in Computer Science, Distributed Applications and Interoperable Systems*, L. Y. Chen und H. P. Reiser, Hg., Cham: Springer International Publishing, 2017, S. 177–191.

[43] S. Mofrad, F. Zhang, S. Lu und W. Shi, „A comparison study of intel SGX and AMD memory encryption technology“ in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy - HASP '18*, Los Angeles, California, 2018, S. 1–8.

[44] Intel, Hg., „Intel 64 and IA-32 Architectures Software Developers Manual: Volume 3D: System Programming Guide, Part 4“, Sep. 2016. [Online] Verfügbar unter: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf.

[45] V. Costan und S. Devadas, „Intel SGX Explained“, *IACR Cryptology ePrint Archive*, Jg. 2016, https://eprint.iacr.org/2016/086.pdf, 2016.

[46] Intel, Hg., „Intel Software Guard Extensions (Intel SGX) SDK for Windows OS: Developer Reference“ Revision Number 2.3, Mrz. 2019.

[47] S. Arnautov *et al.*, „SCONE: Secure Linux Containers with Intel SGX“ in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, S. 689–703.

[48] Intel, Hg., „Intel Software Guard Extensions (Intel SGX): Developer Guide“ Revision Number 2.3, Mrz. 2019.

[49] I. Anati, S. Gueron, S. P. Johnson und V. R. Scarlata, „Innovative Technology for CPU Based Attestation and Sealing“, 2013. [Online] Verfügbar unter: https://software.intel.com/en-us/articles/innovative-technology-for-cpu-based-attestation-and-sealing.

[50] J. van Bulck *et al.*, „Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution“ in *Proceedings of the 27th USENIX Security Symposium*, 2018.

[51] ARM Limited, „ARM Security Technology: Building a Secure System using TrustZone Technology“ Reference no. PRD29-GENC-009492C, Apr. 2009. [Online] Verfügbar unter: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.

[52] Trusted Computing Group, Hg., „Trusted Platform Module Library: Part 1: Architecture“ Revision 01.38, Sep. 2016.

[53] W. Arthur, D. Challener und K. Goldman, *A Practical Guide to TPM 2.0*. Berkeley, CA: Apress, 2015.

[54] T. Akeem Yekini, F. Jaafar und P. Zavarsky, „Study of Trust at Device Level of the Internet of Things Architecture“ in *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, Hangzhou, China, Jan. 2019 - Jan. 2019, S. 150–155.

[55] S. Han, W. Shin, J.-H. Park und H. Kim, „A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping“ in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, S. 1229–1246.

[56] B. C. Witt, *IT-Sicherheit kompakt und verständlich: Eine praxisorientierte Einführung*, 1. Aufl. Wiesbaden: Friedr.Vieweg & Sohn Verlag/GWV Fachverlage GmbH Wiesbaden, 2006.

[57] M. Glinz, „On Non-Functional Requirements“ in *15th IEEE International Requirements Engineering Conference (RE 2007)*, Delhi, Okt. 2007 - Okt. 2007, S. 21–26.

[58] L. Sun, *Amazon Could Dent Intel's Near-Monopoly on Data Center Chips.* [Online] Verfügbar unter: https://www.fool.com/investing/2018/12/13/amazon-could-dent-intels-near-monopoly-on-data-cen.aspx.

[59] P. Rogaway, „Nonce-Based Symmetric Encryption“ in *Lecture Notes in Computer Science, Fast Software Encryption*, T. Kanade et al., Hg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, S. 348–358.

[60] K. Zhao und L. Ge, „A Survey on the Internet of Things Security“ in *2013 Ninth International Conference on Computational Intelligence and Security*, Emeishan 614201, China, Dez. 2013 - Dez. 2013, S. 663–667.

[61] Bundesanzeiger Verlag GmbH; Deutschland, *IT-Grundschutz-Kompendium*, 2. Aufl. Köln: Reguvis Bundesanzeiger Verlag, 2019.

[62] K. Marton, A. Suciu, C. Săcărea und O. Creţ, „Generation and testing of random numbers for cryptographic applications“, *Proceedings of the Romanian Academy - Series A: Mathematics, Physics, Technical Sciences, Information Science*, Jg. 13, S. 368–377, https://pdfs.semanticscholar.org/c4a4/97ad7cd33b9a75c0d2cec0b8cebbf84e2784.pdf, 2012.

[63] *Kernel Virtual Machine.* [Online] Verfügbar unter: http://www.linux-kvm.org/page/Main_Page.

[64] *Xen Project: The Linux Foundation Projects.* [Online] Verfügbar unter: https://xenproject.org/.

[65] *VirtualBox.* [Online] Verfügbar unter: https://www.virtualbox.org/.

[66] Intel, *Intel® Software Guard Extensions SDK for Linux*: SGX VIRTUALIZATION.* [Online] Verfügbar unter: https://01.org/intel-software-guard-extensions/sgx-virtualization.

[67] Dimitrios Desyllas, *Intel SGX Enabled VMS.* [Online] Verfügbar unter: http://sgx-vms.tk/.

[68] Q. H. Dang, „Secure Hash Standard (SHS)". FIPS PUB 180-4, U.S. Department of Commerce.

[69] D. Wätjen, *Kryptographie*. Wiesbaden: Springer Fachmedien Wiesbaden, 2018.

[70] Kortenbrede Datentechnik, *Lizenzschlüssel und Hardware-ID von Windows 10.* [Online] Verfügbar unter: https://kortenbrede-datentechnik.de/KDT/archives/16-Lizenzschluessel-und-Hardware-ID-von-Windows-10.html.

[71] Microsoft, *Reaktivieren von Windows 10 nach einer Änderung der Hardware.* [Online] Verfügbar unter: https://support.microsoft.com/de-de/help/20530/windows-10-reactivating-after-hardware-change.

[72] C. Baun, M. Kunze und T. Ludwig, „Servervirtualisierung", *Informatik Spektrum*, Jg. 32, Nr. 3, S. 197–205, 2009.

[73] B.-A. Yassour, M. Ben-Yehuda und O. Wasserman, „Direct device assignment for untrusted fully-virtualized virtual machines", https://www.researchgate.net/publication/228750224_Direct_device_assignment_for_untrusted_fully-virtualized_virtual_machines, 2008.

[74] W. Schiffmann, H. Bähring und U. Hönig, *Technische Informatik 3: Grundlagen der PC-Technologie*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[75] D. Katusic *et al.*, „Universal identification scheme in machine-to-machine systems" in *Proceedings of the 12th International Conference on Telecommunications*, 2013, S. 71–78.

[76] F. Morkowsky, *Grundlagen der Netzwerktechnik*. Norderstedt: BoD E-Short, 2015.

[77] M. A. Dye *et al.*, *Netzwerkgrundlagen: CCNA Exploration Companion Guide*. München: Addison-Wesley, 2008.

[78] K. Fischer, J. Gessner und S. Fries, „Secure Identifiers and Initial Credential Bootstrapping for IoT@Work" in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, Palermo, Italy, Jul. 2012 - Jul. 2012, S. 781–786.

[79] *VMware.* [Online] Verfügbar unter: https://www.vmware.com/.

[80] T. Raffetseder, C. Kruegel und E. Kirda, „Detecting System Emulators“ in *Lecture Notes in Computer Science, Information Security*, J. A. Garay, A. K. Lenstra, M. Mambo und R. Peralta, Hg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, S. 1–18.

[81] *Apache Maven Projekt.* [Online] Verfügbar unter: https://maven.apache.org/.

[82] *GitHub: Native Operating System and Hardware Information.* [Online] Verfügbar unter: https://github.com/oshi/oshi.

[83] Oracle, *Object (Java Platform SE): Docs Oracle.* [Online] Verfügbar unter: https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#hashCode--.

[84] Oracle, *Random (Java Platform SE): Docs Oracle.* [Online] Verfügbar unter: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html.

[85] Apache, *Class DigestUtils.* [Online] Verfügbar unter: https://commons.apache.org/proper/commons-codec/apidocs/org/apache/commons/codec/digest/DigestUtils.html.

Alle Links wurden zuletzt am 08.10.2019 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift