Fachhochschule Münster
Fachbereich Elektrotechnik und Informatik

Bachelorarbeit
zur Erlangung
des akademischen Grades
Bachelor of Science (B.Sc.)
im Studiengang Informatik

Analyse des Protokolls S7CommPlus im Hinblick auf verwendete Kryptographie

Erstprüfer	Prof. Dr.-Ing. Sebastian Schinzel
Zweitprüfer	Maik Brüggemann

vorgelegt am	26. März 2018
letzte Änderung	5. Dezember 2019
von	Felix Weißberg
Matrikel-Nr.	830102

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Münster, 5. Dezember 2019 Felix Weißberg

Inhaltsverzeichnis

Abbildungsverzeichnis III

Tabellenverzeichnis IV

Glossar V

Akronyme VI

1. Vernetzte Industrie 1
- 1.1. Verwandte Arbeiten . . . 2
- 1.2. Aufbau der Arbeit . . . 2

2. Grundlagen 3
- 2.1. SPS . . . 3
- 2.2. Das Protokoll S7CommPlus . . . 4
  - 2.2.1. Aufbau . . . 5
  - 2.2.2. Opcodes . . . 6
  - 2.2.3. Funktionen . . . 7
- 2.3. Kryptographische Funktionen . . . 10
  - 2.3.1. Notationen . . . 10
  - 2.3.2. Zufallswerte . . . 10
  - 2.3.3. Zeichenfolgen . . . 10
  - 2.3.4. Hash-Funktionen . . . 11
  - 2.3.5. MACs . . . 11
  - 2.3.6. Verschlüsselungsverfahren . . . 12

3. Analyse 13
- 3.1. Teststellung . . . 13
- 3.2. Vorgehen . . . 14
- 3.3. Codeobfuscation . . . 15

4. Ergebnisse 17
- 4.1. Handshake . . . 17
  - 4.1.1. Clientseitige Vorbereitung . . . 18
  - 4.1.2. Nachricht 1 . . . 19

4.1.3. Nachricht 2 . . . 19
4.1.4. Nachricht 3 . . . 19
4.1.5. Nachricht 4 . . . 23
4.1.6. Zusammenfassung . . . 23
4.2. Integritätsgeschützte Kommunikation . . . 25
4.3. Client Authentifizierung . . . 25
4.3.1. Nachricht 1 . . . 26
4.3.2. Nachricht 2 . . . 26
4.3.3. Nachricht 3 . . . 26
4.3.4. Nachricht 4 . . . 27
4.3.5. Zusammenfassung . . . 27

5. Evaluation **28**
5.1. Client-Authentifizierung . . . 28
5.2. AES Komplexitätreduktion . . . 28
5.3. TIA-Portal unabhängige Kommunikation . . . 31

6. Weitere Forschungsfragen **32**
6.1. Key Exchange Verfahren . . . 32
6.2. Deobfuscation . . . 32
6.3. Sicherheitsanalyse . . . 32

7. Fazit **34**

Literaturverzeichnis **35**

A. Anhang **37**
A.1. S7-1500 Public Connection Key . . . 37
A.2. S7-1500 g . . . 37
A.3. Siemens-AES-Counter-Inkrementierung (Python) . . . 38
A.4. Auf einer SBOX basierende Verschlüsselungsfunktion (Python) . . . 39

Abbildungsverzeichnis

2.1. Siemens Simatic S7-1500 3
2.2. Siemens TIA-Portal Screenshot 5
2.3. S7CommPlus Message Frame. Typ: v1/v2 6
2.4. S7CommPlus Message Frame. Typ: v3 7
2.5. “Opcode dependend” Struktur für die Opcodes: “Request”/“Response” . 8
2.6. Struktur des “Function Set”-Feldes für die Funktion CreateObject 8
2.7. Struktur des Objekts der Funktion CreateObject 9
2.8. Struktur des “Function Set” Feldes für die Funktion SetMultipleVariables 9

3.1. Teststellung 13
3.2. Wireshark-Analyse einer S7CommPlus-Nachricht 15
3.3. Verschleierter Code im TIA-Portal 16

4.1. Handshake Ablauf. 17
4.2. Aufbau des “SecurityKeyEncryptedKey” Datenblobs 20
4.3. Berechnungen und Kommunikation im Handshake 24
4.4. Berechnungen und Kommunikation zur Client-Authentifizierung 27

5.1. KXV für S7-1500 Public Key und g und Masterkey, der aus Null-Bytes besteht 31

Tabellenverzeichnis

2.1. S7CommPlus Protokoll-Stack 4

4.1. PLC Zugriffsschutzstufen 26

Glossar

Client Initiator des Verbindungsaufbaus zur SPS.

SPS Speicherprogrammierbare Steuerung. (Um-)Programmierbares Gerät zur automatisierten Steuerung von Maschinen.

TIA-Portal Totally Integrated Automation Portal, ein Windows Programm zum Programmieren und Kommunizieren mit Siemens SPSen.

Akronyme

AES advanced encryption standard.

COTP connection oriented transport protocol.

FBD function block diagram.

ISO International Organization for Standardization.

IV Initialisierungsvektor.

LAD ladder logic.

MAC message authentication code.

MITM man in the middle.

SCL structured text.

TPKT ISO transport service on top of the TCP.

VLQ variable-length quantity.

1. Vernetzte Industrie

Informationsverarbeitende Systeme sind heutzutage aus der Industrie nicht mehr wegzudenken. Waren es vor 50 Jahren noch hauptsächlich Menschen, die die Abläufe steuerten, fand mit Beginn der Entwicklung von Computern auch immer mehr Automatisierung Einzug in die Industrie. Dazu zählen nicht nur teil- oder vollautomatisierte Produktionsstraßen in Fabriken, sondern auch die Verwaltung und Überwachung von Kraftwerken und Infrastruktur jedweder Art.

Um dieses hohen Verwaltungsaufwandes Herr zu werden, ist ein Gerät nötig, in dem sich Abläufe implementieren lassen. So müsste dieses Gerät nach Auswertung von Sensordaten, verarbeitet durch die einprogrammierte Logik, Aktoren ansteuern können, um so einen der Prozesse zu übernehmen, für den zuvor ein Mensch benötigt wurde. Speicherprogrammierbare Steuerungen (SPS) leisten, neben weiteren Eigenschaften, genau dies. Dabei bedeutet speicherprogrammierbar, dass das Programm, das die SPS ausführt, sich umprogrammieren oder überschreiben lässt. Dies geschieht immer häufiger "aus der Ferne", also ohne direkten physischen Zugriff auf das Gerät. Des Weiteren lassen sich so auch oft Informationen über den Gerätestatus einholen, genauso wie Einstellungen am Gerät vornehmen.

Zu diesem Zweck werden die SPS beispielsweise in mittlerweile 93,3% deutscher Maschinenbauunternehmen [1] über Ethernet in das Unternehmensnetzwerk eingebunden, um dann über, meist proprietäre, Protokolle mit ihnen zu kommunizieren. So benutzen SPS der Siemens AG, die nach Herstellerangaben Marktführer im Bereich der SPS [2] in Europa ist, beispielsweise das eigens entwickelte Protokoll S7CommPlus.

Anders als in früheren Versionen des Protokolls werden in den neueren Versionen deutlich mehr kryptographische Verfahren implementiert, um die Authentizität und Integrität der Kommunikation sicherzustellen. Es ist allerdings wenig darüber bekannt, welche Verfahren im Detail eingesetzt werden. Das Ziel dieser Arbeit ist das Schließen dieser Wissenslücke durch eine Analyse des Protokolls. Weiterhin soll dadurch eine Grundlage geschaffen werden, auf der aufbauend Interoperabilität von SPS, die mittels des S7CommPlus Protokolls kommunizieren, und dem Open Source Tool "ICShell" [3] hergestellt werden kann.

1.1. Verwandte Arbeiten

Eine erste Analyse des Protokolls S7CommPlus in Bezug auf verwendete Kryptographie wurde bereits von Cheng Lei, Li Donghong und Ma Liang durchgeführt [4]. Dabei wurden unter anderem Stellen im Protokoll identifiziert, die verschlüsselt sind. Weiter lieferte diese Arbeit Ansatzpunkte für eine tiefergehende Analyse der Implementierung der Verfahren im TIA-Portal. Eine detaillierte Beschreibung verwendeter kryptographischer Verfahren fand dabei allerdings nicht statt. Eine weitere Arbeit stellt das Wireshark Dissector Plugin von Thomas Wiens [5] dar. Dieses bietet einen umfangreichen Einstieg in das Protokoll. Der Arbeitsaufwand, das Protokoll an sich analysieren zu müssen, konnte durch dieses Dissector Plugin auf ein Minimum reduziert werden.

1.2. Aufbau der Arbeit

Diese Arbeit ist in vier Teile aufgeteilt:

(I) Zu Beginn werden Grundlagen zu Industriesteueranlagen und dem von Siemens verwendeten Protokoll S7CommPlus vermittelt. Außerdem werden Notationen festgelegt und Signaturen von Funktionen beschrieben, die für die im Protokoll verwendeten kryptographischen Verfahren von Relevanz sind. Dazu dient das zweite Kapitel.

(II) Weiter wird in Kapitel drei auf das Vorgehen bei der Analyse der im Protokoll verwendeten kryptographischen Verfahren eingegangen.

(III) In Kapitel vier werden die Ergebnisse aus der Analyse vorgestellt.

(IV) Zuletzt werden Eigenschaften des Protokolls beschrieben, die aus Erkenntnissen aus der Analyse hervorgehen, sowie Forschungsfragen gestellt, die sich aus Erkenntnissen und Problemen bei der Analyse ergeben haben. Dazu dienen das fünfte bzw. sechste Kapitel.

2. Grundlagen

In diesem Kapitel werden Grundlagen geschaffen, die für das Verständnis der folgenden Analyse notwendig sind. Dazu wird zuerst ein Überblick über SPS gegeben. Fortgefahren wird mit einer Einführung in das Protokoll S7CommPlus. Abschließend werden für die Analyse relevante Funktionen definiert beziehungsweise für bekannte Funktionen deren Signaturen beschrieben.

2.1. SPS

Eine SPS, oder auch Industriesteueranlage, ist ein Gerät, dass mittels Sensoren, Aktoren und einer Logik in der Lage ist, (Teil-)Prozesse zu automatisieren.

Siemens bietet eine Reihe verschiedener SPS an, die sich durch ihren Einsatzzweck unterscheiden. Dabei unterscheidet Siemens zwischen Basic-, Advanced-, Distributed- und Software-Controllern. Basic-Controller beispielsweise eignen sich für Automatisierungsaufgaben mit geringerem Leistungsbedarf, während sich Advanced-Controller auch für komplexere Aufgaben einsetzen lassen. Konkret wurde für diese Arbeit mit der Siemens Simatic S7-1500 (CPU 1511-1 PN, Firmwareversion V1.8) eine SPS aus der Advanced-Kategorie von Siemens verwendet. Diese ist in 2.1 abgebildet. Einige Einsatzzwecke dieser Anlage sind zum Beispiel [6]:

Abbildung 2.1.: Siemens Simatic S7-1500

- Die Lichtsteuerung im Roveredo Tunnel
- Spanplattenfertigung bei Pfleiderer
- Verpackungsmaschinenherstellung bei Sollas

Eine Programmierung und Konfiguration der SPS kann über das von Siemens entwickelte Programm "Totally Integrated Automation Portal" (TIA-Portal) vorgenommen werden. Dazu kann die SPS zum Beispiel über Ethernet mit einem Windows-PC verbunden werden, auf dem das TIA-Portal installiert ist. Dieses dient dann als Programmierumgebung. Über die Programmiersprachen structured text (SCL), ladder logic (LAD) oder function block diagram (FBD) lässt sich dann ein Programm entwickeln, dass auf die SPS übertragen werden kann. Außerdem kann eine Verbindung zu der SPS hergestellt werden, um so Statuswerte auszulesen, den Betriebszustand der Anlage zu verändern und Konfigurationen sowie Programme in die SPS zu laden. Ein Screenshot des TIA-Portals ist in Abbildung 2.2 zu sehen.

Zur Übertragung des Programms und der Konfiguration nutzt das TIA-Portal das von Siemens entwickelte Protokoll S7CommPlus[1], falls die Ziel-SPS dieses Protokoll unterstützt. Das folgende Kapitel wird einen Überblick über den Aufbau und die Funktionsweise des Protokolls liefern. Des Weiteren gibt Siemens an, Maßnahmen zum Schutz des Zugriffes auf die Anlage ergriffen zu haben, sowie einen Manipulationsschutz, der die Kommunikationsintegrität sicher stellt [6]. Eine Analyse des Protokolls soll zeigen, wie diese Maßnahmen konkret umgesetzt wurden.

2.2. Das Protokoll S7CommPlus

S7CommPlus ist ein von Siemens entwickeltes, proprietäres Binärprotokoll zur Kommunikation mit Siemens-SPS. Über dieses Protokoll lassen sich die SPS programmieren und verwalten, also beispielsweise Variablen auslesen und schreiben, sowie der Betriebszustand verändern. S7CommPlus setzt dabei auf dem, durch die International Organization for Standardization (ISO) standardisierten, nachrichtenorientierten Transportprotokoll connection oriented transport protocol (COTP) [7] auf. Dieses kann mithilfe von TPKT [8] auch über TCP transportiert werden, indem dieses unter anderem Nachrichtengrenzen einführt. Der gesamte Protokoll-Stack ist in 2.1 zu sehen.

Im Folgenden wird davon ausgegangen, dass vor Beginn einer Kommunikation mittels S7CommPlus jeweils eine Verbindung über COTP aufgebaut wurde. Betrachtet wird dann nur noch die Kommunikation über das S7CommPlus Protokoll. Auf darunterliegende Protokollschichten wird nicht weiter eingegangen.

S7CommPlus
COTP
TPKT
TCP
IP

Tabelle 2.1.: S7CommPlus Protokoll-Stack

Mittels des anfangs erwähnten Wireshark Dissector Plugins von Thomas Wiens [5] lässt sich der Aufbau der Nachrich-

[1] Kein offizieller durch Siemens gewählter Name

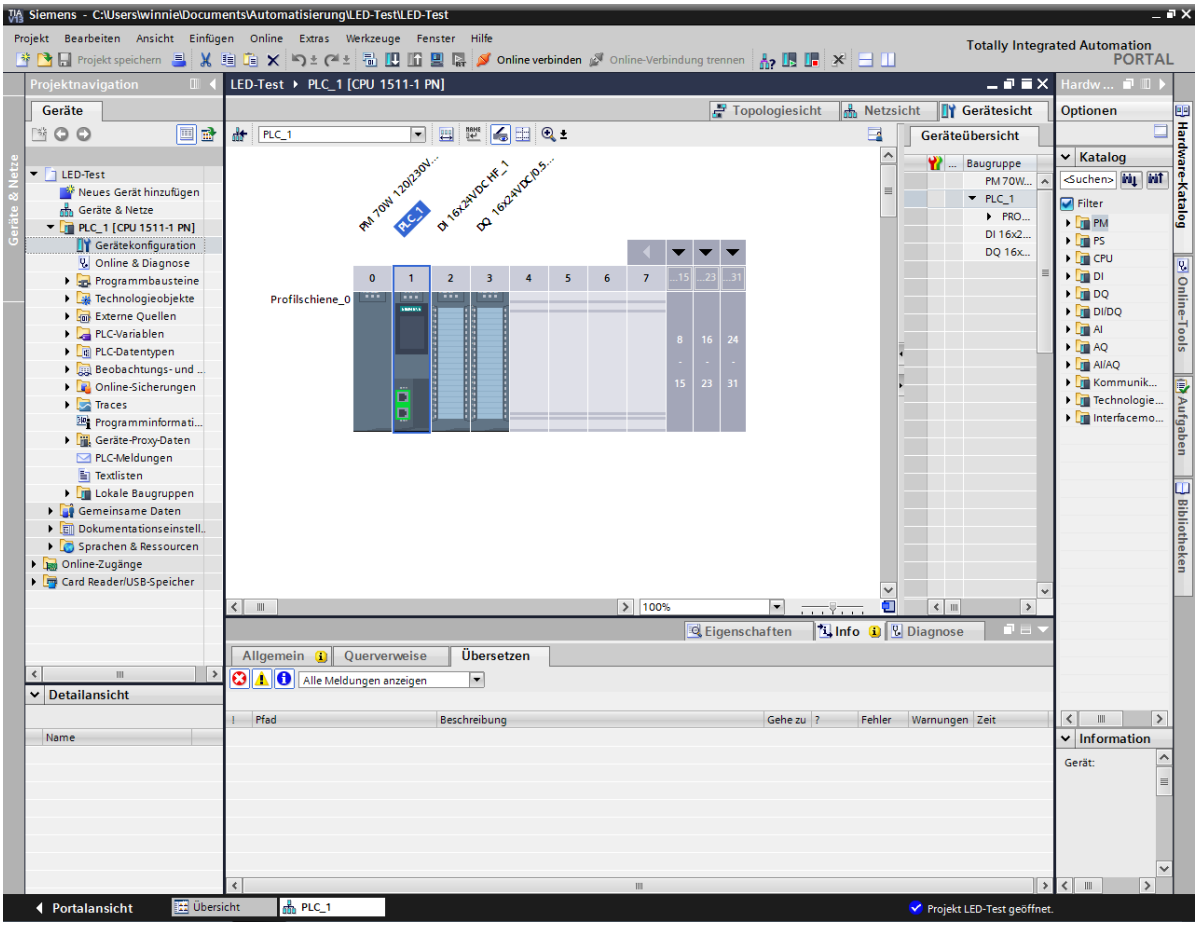

Abbildung 2.2.: Siemens TIA-Portal Screenshot

ten ableiten. Ein grundlegendes Verständnis davon soll im Folgenden geschaffen werden.

2.2.1. Aufbau

Zu Beginn einer jeden S7CommPlus-Nachricht wird das Protokoll und der Nachrichtentyp spezifiziert. Von besonderer Relevanz sind die Nachrichtentypen “v1” und “v2”, die im Verbindungsaufbau verwendet werden, sowie der Nachrichtentyp “v3”, der darauf folgend verwendet wird. Dabei unterscheidet sich die Struktur der Nachrichtentypen “v1” und “v2” nicht. Diese bestehen aus den drei Bereichen: “Header”, “Data” und “Trailer”. Nachrichten des Typs “v3” unterscheiden sich hauptsächlich durch das Vorhandensein eines vierten Bereichs “Integrity Part”, welcher dem Integritätsschutz der Nachricht dient, von den anderen beiden Typen. Die Struktur von Nachrichten des Typs “v1” und “v2” ist in

Abbildung 2.3 zu sehen. Abbildung 2.4 veranschaulicht die Struktur des Nachrichtentyps "v3".

In den folgenden Abbildungen, die den Aufbau des Protokolls veranschaulichen, wird stellenweise die Bezeichnung variable-length quantity (VLQ) als Längenangabe verwendet. Eine VLQ wird benutzt, um beliebig große Integer darzustellen. Dabei wird eine Zahl so kodiert, dass das höchstwertige Bit eines Bytes jeweils angibt, ob weitere Bytes folgen. Eine detaillierte Beschreibung von VLQ lässt sich in [9] finden.

Message Frame v1/v2

Header	
Name	Länge
Protocol Identifier	1 Byte
Message Type	1 Byte
Length	2 Byte

Data	
Name	Länge
Opcode	1 Byte
Opcode dependend	Dynamisch
Unknown	4 Byte

Trailer	
Name	Länge
Protocol Identifier	1 Byte
Protocol Version	1 Byte
Length	2 Byte

Abbildung 2.3.: S7CommPlus Message Frame. Typ: v1/v2

2.2.2. Opcodes

Wie in den Abbildungen 2.3 und 2.4 zu sehen ist, haben alle Nachrichtentypen im Datenbereich das Feld "Opcode" gemein. Über dieses wird der Aufbau des Feldes "Opcode dependend" und damit auch der Kontext, in dem die Nutzdaten stehen, festgelegt. Zwei dieser Opcodes sind "Request" und "Response". Die Struktur der Daten des Feldes "Opcode dependend" ist in Abbildung 2.5 dargestellt.

Message Frame v3	
Header	
Name	Länge
Protocol Identifier	1 Byte
Message Type	1 Byte
Length	2 Byte
Integrity Part	
Name	Länge
Length	1 Byte
Checksum	<Length> Byte
Data	
Name	Länge
Opcode	1 Byte
Opcode dependend	Dynamisch
Integrity ID	VLQ
Unknown	4 Byte
Trailer	
Name	Länge
Protocol Identifier	1 Byte
Protocol Version	1 Byte
Length	2 Byte

Abbildung 2.4.: S7CommPlus Message Frame. Typ: v3

2.2.3. Funktionen

Besonders interessant sind die Felder “Function” und “Function Set”. Über das Feld “Function” lässt sich bei einer “Request”-Nachricht eine Funktion angeben, die auf der Gegenseite ausgeführt wird. Über das Feld “Function Set” können der Funktion Parameter überreicht werden. Nach Ausführung der Funktion antwortet die Gegenseite mit einer “Response”-Nachricht und nutzt das Feld “Function Set” für Rückgabeparameter. Funktions- sowie Rückgabeparameter können, je nach Funktion, verschiedene Arten von Elementen, mit jeweils verschiedenen Strukturen, enthalten.

Von besonderer Relevanz sind die Funktionen “CreateObject” und “SetMultipleVariables”, da diese im Verbindungsaufbau eingesetzt werden, sowie die Funktionen “GetVarSubStreamed” und “SetVarSubStreamed”, die bei der clientseitigen Authentifizierung zum Einsatz kommen.

Opcode dependend	
Name	Länge
Reserved	2 Byte
Function	2 Byte
Reserved	2 Byte
Sequence No.	2 Byte
Session ID	4 Byte
Unknown 1	1 Byte
Function Set	Dynamisch

Abbildung 2.5.: "Opcode dependend" Struktur für die Opcodes: "Request"/"Response"

CreateObject "CreateObject" in einem Request-Kontext erzeugt ein Objekt einer bestimmten Klasse auf der Seite des Kommunikationspartners. Parameter lassen sich in Form eines Containers übergeben, der die, für die Funktion relevanten, Informationen bündelt. Das Feld "Function Set" folgt der in Abbildung 2.6 dargestellten Struktur. Über

Function Set	
Name	Länge
Item ID	4 Byte
Flags	VLQ
Datatype	1 Byte
Value	Datatype dependend
unknown 1	4 Byte
Object	Dynamisch

Abbildung 2.6.: Struktur des "Function Set"-Feldes für die Funktion CreateObject

das Feld "Item ID" wird so zunächst der Container spezifiziert, um dann das Objekt, welches in dem Container liegt, zu definieren. Diese Definition erfolgt innerhalb des "Object"-Feldes und besitzt die in Abbildung 2.7 dargestellte Struktur. Dabei wird über eine "Class ID" die Klasse des Objektes festgelegt. Weiterhin wird das Objekt über Attribute konkretisiert. Diese liegen im Feld "Object Data" und besitzen eine ID-Value Struktur. War es mittels des TIA-Portals möglich, einen Namen zu einer ID zu ermitteln, wird im Weiteren dieser Name verwendet, die IDs sind allerdings stets Zahlenwerte. Wird die Funktion "CreateObject" in einem Response-Kontext angegeben, wird die "Function Set"-Struktur um einen Rückgabewert erweitert. Des Weiteren wird dann das eigens erzeugte Objekt zurückgegeben.

SetMultipleVariables Mittels der Funktion "SetMultipleVariables" lassen sich Attributwerte von Objekten auf der Seite des Kommunikationspartners verändern. Dazu wird

Object	
Name	Länge
Tag ID	1 Byte
Relation ID	4 Byte
Class ID	VLQ
Class Flags	VLQ
Attribute ID	1 Byte
Object Data	Dynamisch
Tag ID	1 Byte

Abbildung 2.7.: Struktur des Objekts der Funktion CreateObject

über das Feld "Object ID" zuerst die ID des zu verändernden Objekts angegeben. Darauf folgen zwei Listen "Addresslist" und "Valuelist". Die erste enthält alle IDs der zu verändernden Attribute, die zweite die entsprechenden Werte zu den jeweiligen Attributen. Dieser Aufbau ist in Abbildung 2.8 dargestellt.

Function Set	
Name	Länge
Object ID	4 Byte
Item Count	1 Byte
Item Addr Count	1 Byte
Addresslist	Dynamisch
Valuelist	Dynamisch
Object Qualifier	Dynamisch

Abbildung 2.8.: Struktur des "Function Set" Feldes für die Funktion SetMultipleVariables

GetVarSubStreamed/ SetVarSubStreamed Diese Funktionen dienen dem Zweck, Variablen auf der Gegenseite zu lesen (GetVarSubStreamed) bzw. zu schreiben (SetVarSubStreamed). Die zu lesende bzw. zu schreibende Variable wird dabei über eine ID identifiziert. Die Antwort auf eine GetVarSubStreamed Request-Nachricht ist der Wert dieser Variable. Die Antwort auf eine SetVarSubStreamed Request-Nachricht ist ein Rückgabewert, der den Erfolg oder Misserfolg des Schreibens signalisiert.

2.3. Kryptographische Funktionen

Im Folgenden werden Funktionen, die für die Beschreibung der kryptographischen Verfahren im S7CommPlus Protokoll von Bedeutung sind, definiert. Falls es sich um wohl bekannte Funktionen handelt (bspw. Funktionen der SHA Familie), werden lediglich die Signaturen dieser Funktionen, wie sie hier verwendet werden, beschrieben.

2.3.1. Notationen

Sei $B := \{0,1\}^8$ die Menge aller Bytes. $x \in B^n$ mit $n \in \mathbb{N}$ heißt Bytefolge und besitzt eine Länge von n Byte. Weiter sei

$$\begin{aligned} \cdot \,||\, \cdot : B^m \times B^n &\to B^{m+n} \\ (a_0, a_1, .., a_{m-1}) \,||\, (b_0, b_1, .., b_{n-1}) &\mapsto (a_0, a_1, .., a_{m-1}, b_0, b_1, .., b_{n-1}) \end{aligned}$$

mit $a \in B^m, b \in B^n$ und $m, n \in \mathbb{N}$ die Konkatenation zweier Bytefolgen und

$$\begin{aligned} \cdot\,[\cdot : \cdot] : B^n \times \mathbb{N}_0 \times \mathbb{N}_0 &\to B^m \\ a[i:j] &\mapsto (a_i, a_{i+1}, .., a_{j-1}, a_j) \end{aligned}$$

mit $a \in B^n$, $i, j \in \mathbb{N}_0$, $n, m \in \mathbb{N}$ und $i \leq j < n$ das Entnehmen einer Teilfolge.

2.3.2. Zufallswerte

Werden für Verfahren Zufallswerte benötigt, so geschieht dies im TIA-Portal über eine pseudozufällige Funktion. Die Implementierung dieser Funktion ist Teil des TIA-Portals. Lediglich zur Initialisierung wird eine pseudozufällige Funktion verwendet, die durch Windows bereitgestellt wird. Im Folgenden sei <n> mit $n \in \mathbb{N}$ ein zufällig gewähltes Element aus B^n.

2.3.3. Zeichenfolgen

Eine Zeichenfolge ist eine Bytefolge, wobei dessen Bytes durch ihren entsprechenden ASCII-Representanten dargestellt werden. Beispiel:

$$\text{'ABC'} = ((0,1,0,0,0,0,0,1)(0,1,0,0,0,0,1,0)(0,1,0,0,0,0,1,1))$$

2.3.4. Hash-Funktionen

Für die spätere Beschreibung der Verfahren im Protkoll S7CommPlus relevante Hash-Funktionen sind:

SHA-1

SHA-1($data$) errechnet zu einem Eingabedatum $data$ dessen 20 Byte langen SHA-1 Hashwert. Für weitere Informationen zu SHA-1 siehe [10].

$$\text{SHA-1} : B^n \rightarrow B^{20},\ n \in \mathbb{N}$$

SHA-256

SHA-256($data$) errechnet zu einem Eingabedatum $data$ dessen 32 Byte langen SHA-256 Hashwert. Für weitere Informationen zu SHA-256 siehe [10].

$$\text{SHA-256} : B^n \rightarrow B^{32},\ n \in \mathbb{N}$$

2.3.5. MACs

Message authentication codes (MACs) werden zum Schutz der Integrität und der Authentizität, also des Ursprungs, von Nachrichten eingesetzt. Um einen MAC zu berechnen gibt es verschiedene Möglichkeiten. Von diesen werden zwei vorgestellt, die in Verfahren im Protokoll S7CommPlus Verwendung finden.

HMAC-SHA-256

HMAC-SHA-256 ist das HMAC-Verfahren unter der Verwendung von SHA-256 als Hash Funktion. Dabei berechnet HMAC-SHA-256($data, key$) für das Eingabedatum $data$ einen durch den Schlüssel key authentifizierten MAC. Für weitere Informationen zu HMAC-SHA-256 siehe [10].

$$\text{HMAC-SHA-256} : B^m \times B^n \rightarrow B^{32},\ m, n \in \mathbb{N}$$

CBC-MAC

CBC-MAC ist ein MAC Verfahren, dass auf einer Blockchiffre beruht. Der erste Block der Eingabedaten wird mit einem Initialisierungsvektor (IV) exklusiv-oder verknüpft und verschlüsselt. Das Ergebnis der Verschlüsselung wird dann jeweils immer mit dem nächsten Eingabedatenblock exklusiv-oder verknüpft und wieder verschlüsselt. Lediglich das Ergebnis des letzten verschlüsselten Blocks wird als MAC verwendet. Da der IV oft, wie auch in dem konkreten Fall im S7CommPlus Protokoll, ausschließlich aus Null-Bytes besteht, muss dieser nicht übertragen werden und fließt auch in die Signatur der Funktion nicht mit ein. Damit besteht

CBC-MAC$_E(data, key)$ dann nur noch aus der Verschlüsselungsfunktion E, den Eingabedaten $data$ und dem Schlüssel key für die Verschlüsselungsfunktion. Da CBC-MAC im Protokoll S7CommPlus mit einer Verschlüsselungsfunktion eingesetzt wird, die eine Blocklänge von 16 Byte hat, sei

$$\text{CBC-MAC}_E : B^n \times B^{16} \to B^{16},\ n \in \mathbb{N}$$

mit E$(block, key)$, dem zu verschlüsselnden Block $block$ unter Verwendung des Schlüssel key

$$E : B^{16} \times B^{16} \to B^{16}$$

das CBC-MAC Verfahren.

2.3.6. Verschlüsselungsverfahren

Wird im weiteren Verlauf dieser Arbeit das Verschlüsselungsverfahren advanced encryption standard (AES) erwähnt, ist AES mit einer Schlüssellänge von 128 Bit gemeint. Falls AES in einem bestimmten Betriebsmodus betrieben wird, wird dieser jeweils angegeben.

AES

AES-128$(block_{plain}, key)$ ohne einen bestimmten Betriebsmodus ist eine Blockchiffre, die einen 128 Bit großen Block $block_{plain}$ unter Verwendung eines 128 Bit langen Schlüssels key chiffriert.

$$\text{AES} : B^{16} \times B^{16} \to B^{16}$$

AES-CTR-Mode

AES-CTR$_{CTR}(data, key, ctr)$ beschreibt das Betreiben von AES im Counter Mode. Dabei wird eine Bytefolge $ctr \in B^{16}$ mit dem Schlüssel $key \in B^{16}$ mittels AES verschlüsselt, um eine pseudozufällige Bytefolge zu erzeugen. Dies wird wiederholt angewandt, wobei der Counter ctr nach jeder Iteration jeweils durch eine Funktion CTR verändert wird. So wird ein beliebig langer Keystream erzeugt, der mittels XOR die Eingabedaten verschlüsselt.

$$\text{AES-CTR}_{CTR} : B^n \times B^{16} \times B^{16} \to B^n$$

mit $n \in \mathbb{N}$ und

$$\text{CTR} : B^{16} \to B^{16}$$

3. Analyse

Im Folgenden wird näher auf die Analyse der im Protokoll S7CommPlus verwendeten Kryptographie eingegangen. Dabei beschränkt sich dieses Kapitel auf das Vorgehen bei der Analyse, während die Ergebnisse im nächsten Kapitel gesondert betrachtet werden.

3.1. Teststellung

In einem ersten Schritt wird eine Teststellung bestehend aus einer S7-1500 Anlage und einem Windows 7 System mit installiertem TIA-Portal in der Version 13 aufgebaut. Des Weiteren wird ein Debian-Linux System als Proxy zwischen Anlage und Windows System eingerichtet.

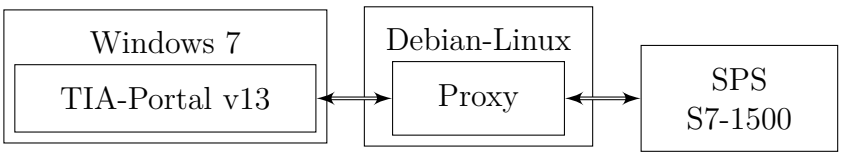

Abbildung 3.1.: Teststellung

Die Anwendung zur Realisierung des Proxys wurde selbst entwickelt, sodass auch das Verändern von Nachrichten einfach möglich ist. Wie in Abbildung 3.1 zu sehen ist, kommuniziert das TIA-Portal mit dem Proxy. Dieser leitet die Nachrichten an die SPS weiter, beziehungsweise Nachrichten, die von der SPS kommen, an das TIA-Portal. Dabei bietet er außerdem die Möglichkeit, Nachrichten zu verändern. Dazu lassen sich beim Start des Proxys "Manipulators" angeben. "Manipulators" sind Python-Dateien, in denen die Funktion *manipulate*(*data*) implementiert wird. Die Eingabe "data" stellt dabei die zu manipulierende Nachricht dar. Die Rückgabe der Funktion ist die manipulierte Nachricht. Jeder "Manipulator" kann dabei selbst entscheiden, ob die aktuelle Nachricht durch ihn verändert werden soll oder nicht. Konkret wurde dies umgesetzt, in dem jeder "Manipulator" um eine Funktion *match*(*data*) ergänzt wurde. Diese wird in der Funktion *manipulate* zu Beginn aufgerufen und überprüft, ob die vorliegende Nachricht mit einem bestimmten, vorher festgelegten, Muster übereinstimmt. Ist dies der Fall, wird die Nachricht verändert, ansonsten wird sie unverändert zurückgegeben.

3.2. Vorgehen

Mit Wireshark [11] und dem S7CommPlus Dissector Plugin [5] wird das Protokoll auf Nachrichtenteile hin untersucht, die mutmaßlich in Verbindung zur Sicherheit des Protokolls stehen. Mit der Sicherheit des Protokolls ist der Schutz von Integrität, Authentizität und Vertraulichkeit der Nachrichten gemeint. Dabei stellt sich heraus, dass ein Schutz der Vertraulichkeit im Protokoll nicht umgesetzt wird, da Nutzdaten unverschlüsselt übertragen werden. Des Weiteren stellt sich heraus, dass sich das Protokoll aus Sicherheitsperspektive in drei Teile unterteilen lässt:

1. Handshake
2. Integriätssumme
3. Client-Authentifizierung

Der Handshake findet zu Beginn einer Verbindung, im Verbindungsaufbau, mittels vier Nachrichten statt. Hier ist insbesondere ein Feld interessant, dass im S7CommPlus-Dissector als "Encrypted Key" bezeichnet wird. Abbildung 3.2 zeigt die Wireshark-Analyse der Nachricht mit dem "Encrypted Key" Feld und dessen, durch das S7CommPlus Dissector Plugin vermuteten, Aufbau. Die Integritätssumme befindet sich im "Integrity Part" in jeder v3 Nachricht und dient dem Schutz der Integrität der Nachrichten. Zuletzt wurde eine clientseitige Authentifizierung in Form von vier Nachrichten ausgemacht. Diese soll die Authentizität des Clients sicher stellen.

Die Art und Weise der Berechnung und der Zusammenhang dieser drei Protokollspezifika ist Ziel der Analyse.

Hierbei dient die Analyse des Feldes "Encrypted Key" im Handshake als Einstiegspunkt.

Um den Code im TIA-Portal zu finden, der dieses Feld berechnet, wird eine statische Analyse des .Net Codes des TIA-Portals durchgeführt. Dazu wird das Programm dnSpy [12] eingesetzt. Es zeigt sich dabei, dass die OMSP_core_managed.dll von besonderem Interesse ist, da diese eine Klasse mit dem Namen "ClientSession" enthält, welche für die Anlagenkommunikation auf .Net Ebene verantwortlich ist. Die Klasse "ClientSession" besitzt unter anderem die Methode "Login", die durch einen Native-Call in Assembly-Code fortgesetzt wird. Mit dem Programm x64dbg/x32dbg [13] kann dieser Assembly-Code von der Einsprungstelle an dynamisch analysiert werden. Das Ergebnis der dynamischen Codeanalyse ist das Auffinden des Codes, der für die Berechnung des Feldes "Encrypted Key" verantwortlich ist.

Durch ein Ausdehnen der dynamischen Analyse auf vormals nicht betrachtete Codepassagen können auch die Funktionen ermittelt werden, die für die Berechnung der Integritätsumme sowie die Client-Authentifizierung zuständig sind.

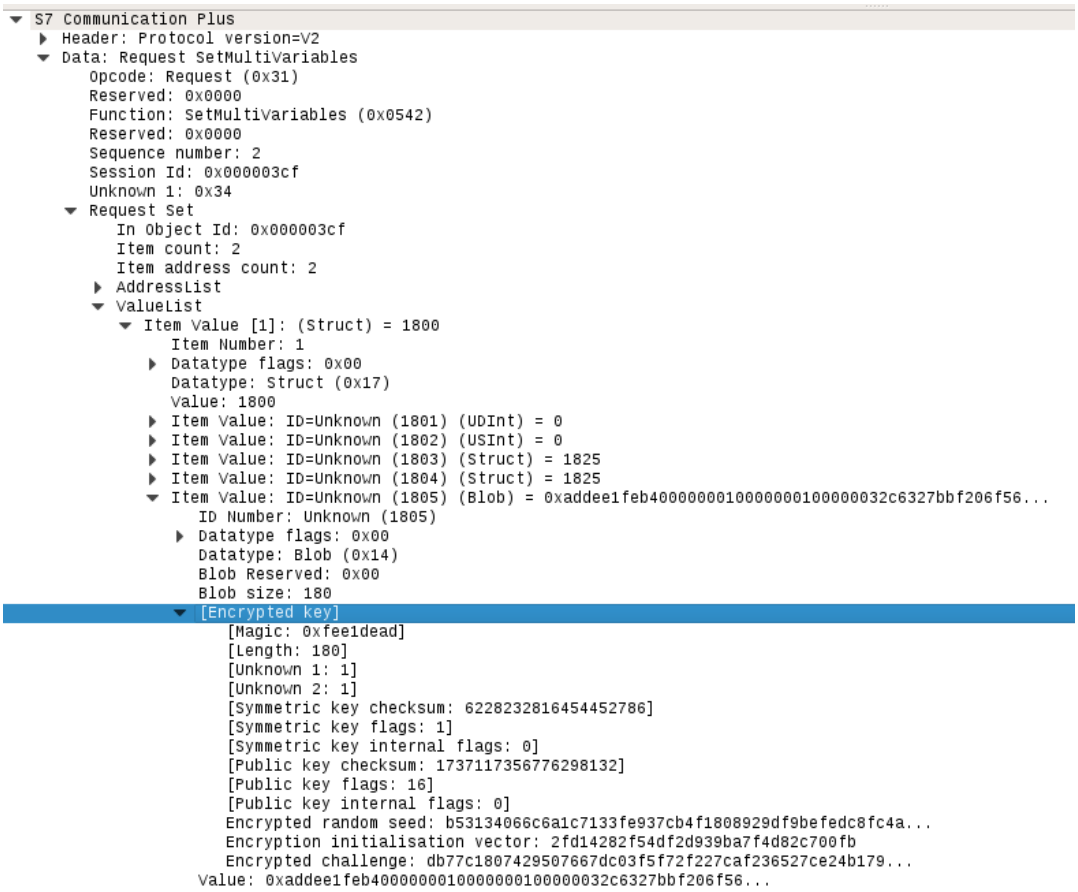

Abbildung 3.2.: Wireshark-Analyse einer S7CommPlus-Nachricht

Die Analyse des TIA-Portals bringt außerdem eine Zuordnung von im Protokoll verwendeten IDs zu Strings zum Vorschein. Dies führt stellenweise dazu, dass die Bezeichnungen für Felder und Elemente des Protokolls in dieser Arbeit von den Bezeichnungen aus dem Wireshark Dissector Plugin abweichen.

3.3. Codeobfuscation

Einige der für die Analyse relevanten Codepassagen sind auf Grund von eingesetzten Codeobfuscation-Techniken schwierig zu untersuchen. In solchen Fällen werden die Funktionssignaturen ermittelt. Außerdem werden die Funktionen extrahiert, um sie empirisch analysieren zu können. So ist es häufig möglich, auch ohne die genaue Funktionsweise zu kennen, den Zweck und einige Eigenschaften der Funktionen zu ermitteln.

Die Abbildung 3.3 zeigt eine der verschleierten Funktionen. Sie besteht aus ca. 21500 Assembler-Instruktionen. Von diesen sind ca. 21000 entweder XOR-, AND-, MOV- oder NOT-Operationen. Dabei besitzen 1400 der XOR- und AND-Operationen eine Konstante als Operanden. Zusammengenommen werden durch diese beiden Operationen so ca. 5200 Bytes an Konstanten verarbeitet. Diese können keinem bekannten Verfahren zugeordnet werden. Des Weiteren verzweigt sich der Code nicht, das heißt es gibt keine Sprung-Operationen (jmp, jne, call, etc.).

Diese Eigenschaften der Funktion führen dazu, dass ihre Analyse erschwert wird. Auf Versuche, die Funktion in eine für Menschen lesbarere Form zu überführen, also die Verschleierung des Codes rückgangig zu machen, wird verzichtet.

```
xor esi,5018A2D8
and esi,D858A3DC
mov ebx,ecx
and ebx,50B98BD4
xor esi,ebx
xor esi,edi
and esi,dword ptr ss:[ebp-2A4]
mov edi,ecx
and edi,98F9A918
xor edi,dword ptr ss:[ebp+68]
mov ebx,edx
xor edi,D8F90BD0
and edi,edx
xor esi,edi
mov edi,dword ptr ss:[ebp-2D8]
and edi,A25820
xor edi,88E08904
and edi,ecx
xor esi,edi
and ebx,5058C3F4
mov edi,ecx
xor edi,FFFAFEFB
and edi,501F93D6
xor edi,ebx
and edi,dword ptr ss:[ebp-2A4]
mov ebx,ecx
```

Abbildung 3.3.: Verschleierter Code im TIA-Portal

4. Ergebnisse

Die Kommunikation zwischen Client und SPS mittels des Protokolls S7CommPlus lässt sich im Hinblick auf sicherheitsrelevante Aspekte der Kommunikation in drei Phasen unterteilen: Handshake, integritätsgeschützte Kommunikation und Client-Authentifizierung. Im Folgenden werden die Ergebnisse aus der Analyse dieser drei Phasen in Bezug auf verwendete kryptographische Verfahren vorgestellt.

4.1. Handshake

Der Handshake findet zu Beginn einer S7CommPlus-Verbindung, im Verbindungsaufbau, in Form von vier Nachrichten statt. Aus Sicherheitsperspektive dient er dazu, einen Session-Key auszutauschen. Dieser wird für die integritätsgeschützte Kommunikation und die Client-Authentifizierung benötigt.

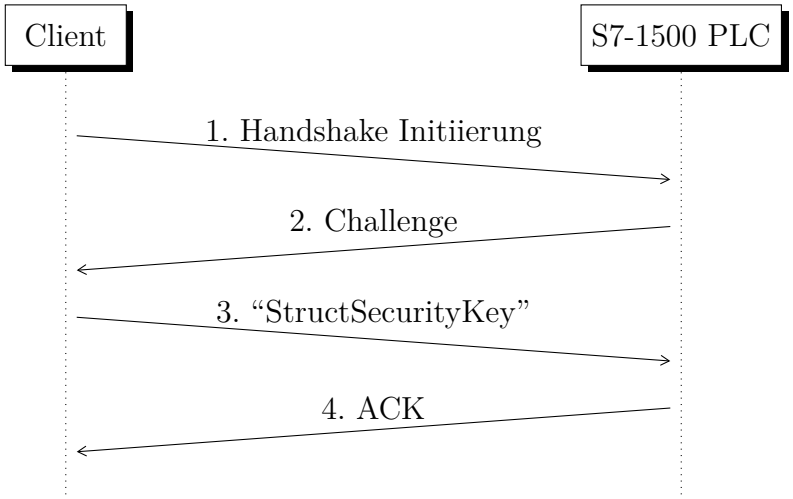

Abbildung 4.1.: Handshake Ablauf.

Der Handshake wird durch den Client mittels einer ersten Nachricht an die SPS initiiert. Die Antwort der SPS auf die erste Nachricht ist eine Challenge. Diese bildet den ersten Teil eines Challenge-Response Verfahrens. Die dritte Nachricht enthält eine Struktur

"StructSecurityKey", welche dem Austausch des Session-Keys, sowie als Antwort auf die Challenge dient. Die letzte Nachricht bescheinigt einen Erfolg des Verfahrens. Der Ablauf des Handshakes ist schematisch in Abbildung 4.1 dargestellt. Auf die einzelnen Nachrichten im Handshake, sowie insbesondere die Struktur "StructSecurityKey" wird im Folgenden näher eingegangen. Zuvor wird beschrieben, wie sich der Client auf den Handshake vorbereitet.

4.1.1. Clientseitige Vorbereitung

Vor Aufbau der Kommunikation erzeugt der Client einen Master-Schlüssel $K_M \in B^{60}$ und, davon abgeleitet, drei weitere Schlüssel $K_{enc}, K_{sbox}, K_{mac} \in B^{16}$. Dabei dient K_{enc} als Schlüssel für eine AES-Verschlüsselung. K_{sbox} wird als Schlüssel für die Initialisierung einer SBOX benutzt, die in einem späteren Schritt in einem Verschlüsselungsverfahren verwendet wird. K_{mac} wird innerhalb der Berechnung eines MAC benötigt. Die dazu verwendeten Funktionen D_1 und D_2 benutzen intern verschleierte Funktionen (Code Obfuscation). Eine Analyse dieser Funktionen wird dadurch erheblich erschwert, sodass diese Funktionen hier nur durch deren Signatur beschrieben werden, nicht aber deren Funktionsweise.

$$\begin{aligned} r &= <24> \\ K_M &= D_1(r) \\ K &= D_2(K_M) \\ K_{enc} &= K[0:15] \\ K_{sbox} &= K[16:31] \\ K_{mac} &= K[32:47] \end{aligned}$$

mit

$$\begin{aligned} D_1 &: B^{24} \rightarrow B^{60} \\ D_2 &: B^{60} \rightarrow B^{48} \end{aligned}$$

Des Weiteren erzeugt der Client einen Schlüssel $K_{session} \in B^{24}$ mit

$$K_{session} = <24>$$

Er besitzt außerdem den Public-Key $K_{pub} \in B^{40}$ der SPS, sowie eine weitere 40 Byte lange Bytefolge $g \in B^{40}$. K_{pub} und g sind bekannt (s. A.1 und A.2) und statisch, das heißt, diese Werte verändern sich auch zwischen Sessions nicht.

4.1.2. Nachricht 1

In einer ersten Nachricht von Client an SPS wird diese mittels der “CreateObject” Funktion angewiesen, ein Objekt der Klasse “ClassServerSession” zu erstellen.

4.1.3. Nachricht 2

Die Antwort der SPS auf die erste Nachricht enthält das erstellte Objekt. Dieses Objekt beinhaltet unter anderem das Attribut “ServerSessionChallenge”, ein 20 Byte langes Array, welches im Folgenden als $challengeFull$ bezeichnet wird. $ChallengeFull$ wird durch die SPS, mutmaßlich zufällig, gewählt. Durch das Verändern einzelner Bytes der $challengeFull$ im Proxy stellt sich heraus, dass nur 16 dieser 20 Bytes von Relevanz sind. Veränderungen an irrelevanten Byte der $challengeFull$ führen dazu, dass der Handshake trotzdem erfolgreich abläuft. Der relevante Teil von $ChallengeFull$ wird bezeichnet als $challenge \in B^{16}$. Es gilt

$$challenge = challengeFull[2:17]$$

Die restlichen vier Byte der $challengeFull$, die nicht mit in die $challenge$ einfließen, werden auch in weiteren Berechnungen nicht weiter verwendet. Nach dieser Nachricht besitzen sowohl Client als auch SPS die 16 Byte lange Bytefolge $challenge$.

4.1.4. Nachricht 3

Die dritte Nachricht, und damit die Antwort auf die Challenge, ist vom Typ Request mit der Funktion “SetMultipleVariables”. Das heißt, es sollen Attribute eines Objektes auf der SPS-Seite verändert werden. Das Zielobjekt dieser Funktion ist das durch CreateObject erstellte Objekt der Klasse “ClassServerSession”. Von diesem Objekt sollen zwei Attribute verändert werden: “ServerSessionClientVersion” und ein Attribut mit der ID 1830, das im Weiteren als “SecurityAttribute” bezeichnet wird. Der dem “SecurityAttribute” entsprechende Wert in der “ValueList” ist eine Struktur vom Typ “StructSecurityKey”. Diese Struktur enthält fünf Elemente, die jeweils durch eine ID bezeichnet werden. Diese IDs können mittels der Zuordnung von IDs zu Strings aus dem TIA-Portal wie folgt bezeichnet werden:

- “SecurityVersion”
- “SecurityKeySecurityLevel”
- “SecurityKeyPublicKeyID”
- “SecurityKeySymmetricKeyID”

- "SecurityKeyEncryptedKey"

"SecurityVersion" und "SecurityKeySecurityLevel" besaßen bei der Analyse dabei stets den Wert 0. Das Element "SecurityKeyPublicKeyID" enthält die ID des öffentlichen Schlüssels K_{pub} der SPS als VLQ, sowie die Werte 65552 und 0. Das Element "SecurityKeySymmetricKeyID" enthält die ID des Session-Keys $K_{session}$ als VLQ, sowie die Werte 65537 und 0. Die Art und Weise der Berechnung der IDs zu den Schlüsseln folgt im Abschnitt "Symmetric-Key ID" bzw. "Public-Key ID". Von besonderem Interesse ist das Element "SecurityKeyEncryptedKey", da darüber der Schlüssel $K_{session}$ ausgetauscht wird, der für die integritätsgeschützte Kommunikation von Nöten ist. Im Wireshark Dissector Plugin wurde das Element als "Encrypted Key" bezeichnet.

"SecurityKeyEncryptedKey" ist vom Typ "Blob" und hat eine Größe von 180 Byte. Diese 180 Byte wurden auf zwei Ziele hin analysiert:

1. Erkennen einer Struktur in den Daten
2. Nachvollziehen der Berechnung der Felder in der Struktur

Das Ergebnis in Bezug auf das erste Ziel, die erkannte Struktur, ist in Abbildung 4.2 exemplarisch dargestellt. Auf die Felder, wie sie in der Darstellung zu sehen sind, soll im Folgenden näher eingegangen werden. Dabei soll im Zuge des zweiten Analyseziels auch dargelegt werden, wie die Felder jeweils berechnet werden.

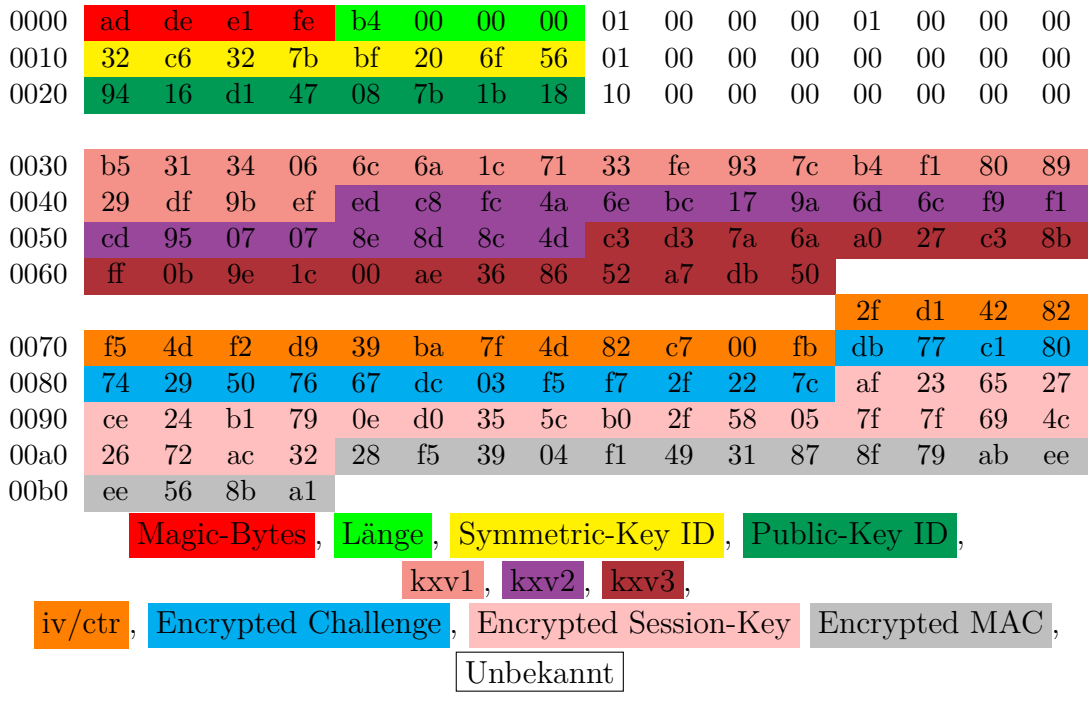

Abbildung 4.2.: Aufbau des "SecurityKeyEncryptedKey" Datenblobs

Magic-Bytes Dieses Feld hat eine Länge von 4 Byte, die Bytereihenfolge ist Little-Endian. Der Wert des Feldes ist statisch: 0xfee1dead.

Länge Dieses Feld hat eine Länge von 4 Byte, die Bytereihenfolge ist Little-Endian. Der Wert des Feldes ist die Länge des Blobs in Byte.

Symmetric-Key ID Dieses Feld ist eine Bytefolge mit einer Länge von 8 Byte. Der Wert des Feldes, $symmKeyID \in B^8$, berechnet sich wie folgt:

$$symmKeyID = \text{SHA-256}(K_{session} \,||\, \text{'DERIVE'})[0:7]$$

Public-Key ID Dieses Feld ist eine Bytefolge mit einer Länge von 8 Byte. Der Wert des Feldes, $pubKeyID \in B^8$, berechnet sich wie folgt:

$$pubKeyID = \text{SHA-256}(K_{pub}[0:23] \,||\, \text{'DERIVE'})[0:7]$$

Key Exchange Vector 1 (kxv1) Dieses Feld ist eine Bytefolge mit einer Länge von 20 Byte. Im Zusammenspiel mit kxv2 tauschen Client und SPS über dieses Feld den Master-Key K_M aus. Eine Beschreibung der Berechnung von kxv1 und kxv2 wird im Abschnitt über kxv3 gegeben.

Key Exchange Vector 2 (kxv2) Dieses Feld ist eine Bytefolge mit einer Länge von 20 Byte. Im Zusammenspiel mit kxv1 tauschen Client und SPS über dieses Feld den Master-Key K_M aus. Eine Beschreibung der Berechnung von kxv1 und kxv2 findet im Abschnitt über kxv3 statt.

Key Exchange Vector 3 (kxv3) Dieses Feld ist eine Bytefolge mit einer Länge von 20 Byte. Das Verfahren, dass zum Austausch des Master-Keys eingesetzt wird, konnte nicht in Gänze nachvollzogen werden, da entsprechende Codepassagen im Assembly des TIA-Portals durch die Siemens AG verschleiert wurden (Code Obfuscation). Das Verfahren besteht aus acht Funktionen, die intern verschleierte Funktionen verwenden. Da einige der Funktionen stets hintereinander ausgeführt werden, lassen sich diese zusammenfassen. Außerdem gibt es einen Parameter, der das Ergebnis der Funktion nicht beeinflusst, sodass dieser weggelassen werden kann. Das Verfahren lässt sich dann beschreiben durch

$$\begin{aligned} f_1 &: B^{40} \times B^{20} \to B^{72} \\ f_2 &: B^{72} \to B^{20} \\ f_3 &: B^{72} \times B^{60} \to B^{20} \end{aligned}$$

$$\begin{aligned} r &= <20> \\ kxv1 &= f_2(f_1(g, r)) \\ kxv2 &= f_3(f_1(K_{pub}, r), K_M) \end{aligned}$$

Dabei ist K_M der im Vorfeld vom Client berechnete Master-Key, von dem die Schlüssel K_{enc}, K_{sbox} und K_{mac} abgeleitet werden. K_{pub} ist mutmaßlich ein öffentlicher, asymmetrischer Schlüssel (siehe A.1). Die Bytefolge $g \in B^{20}$ ist fest (siehe A.2).

Auf Grund der Struktur des Verfahrens, sowie der Länge des öffentlichen Schlüssels, liegt die Vermutung nahe, dass es sich um ECC El Gamal mit einer 160 Bit Kurve mit dem Generator g handelt. Die Vermutung konnte allerdings nicht bestätigt werden. Im Besonderen steht die Frage im Raum, wie ein 60 Byte langer Schlüssel K_M mittels einer 20 Byte langen Bytefolge ausgetauscht werden kann. Eine mögliche Erklärung dafür ist, dass die effektive Länge des Schlüssels K_M 20 Byte beträgt und lediglich auf Grund eines Obfuscation-Layers bei der dynamischen Analyse eine Länge von 60 Byte aufweist. Diese These scheint auch deshalb plausibel zu sein, weil K_M ausschließlich von verschleierten Funktionen verarbeitet wird.

Da der Schlüsselaustausch auch mit im Proxy manipuliertem kxv3 funktioniert, ist kxv3 selbst für den Schlüsselaustausch offensichtlich irrelevant. Berechnet wird kxv3 durch

$$\text{kxv3} = <20>$$

iv/ctr Dieses Feld ist eine Bytefolge mit einer Länge von 16 Byte. Es dient als initialer Counter für die folgende AES-Verschlüsselung im Counter Mode und wird im Weiteren als ctr_{init} bezeichnet.

$$ctr_{init} = <16>$$

Encrypted Challenge Dieses Feld ist eine Bytefolge mit einer Länge von 16 Byte. Berechnet wird sie, indem die Challenge *challenge* aus der zweiten Nachricht mittels AES im CTR-Mode unter Verwendung des Schlüssel K_{enc} und dem initialen Counter ctr_{init} verschlüsselt wird. Die Berechnung ist in dem Abschnitt “Encrypted Session-Key” enthalten.

Encrypted Session-Key Dieses Feld ist eine Bytefolge mit einer Länge von 24 Byte. Berechnet wird sie, indem der Session-Key $K_{session}$ mittels AES im CTR-Mode unter Verwendung des Schlüssels K_{enc} und dem initialen Counter ctr_{init} verschlüsselt wird. Dazu wird berechnet

$$\begin{aligned} data_{plain} &= challenge \mid\mid K_{session} \\ data_{enc} &= \text{AES-CTR}_{SCTR}(data_{plain}, K_{enc}, ctr_{init}) \end{aligned}$$

wobei eine Python-Implementierung von SCTR in A.3 zu finden ist. $data_{enc}$ bildet dann den Wert für die beiden Felder “Encrypted Challenge” und “Encrypted Session-Key”.

Encrypted MAC Dieses Feld ist eine Bytefolge mit einer Länge von 16 Byte. Es dient als MAC über die Felder “iv/ctr”, “Encrypted Challenge” und “Encrypted Session-Key”. Zur Berechnung dieses MACs wird das Verfahren CBC-MAC mit einer Verschlüsselungsfunktion SE benutzt. Dabei handelt es sich bei SE um eine Blockverschlüsselung, die auf einer mit einem Schlüssel initialisierten SBOX arbeitet. Das Verfahren konnte nicht identifiziert werden. Eine Rekonstruktion des Verfahrens war allerdings möglich. So ist in A.4 eine Python-Implementierung des Verschlüsselgunsverfahrens zu finden. Nach Berechnung des MAC wird dieser in einem letzten Schritt mittels AES mit dem Schlüssel K_{mac} verschlüsselt.

$$\begin{aligned} message &= ctr_{init} \mid\mid data_{enc} \mid\mid const \\ mac &= \text{CBC-MAC}_{SE}(message, K_{sbox}) \\ mac_{enc} &= AES(mac, K_{mac}) \end{aligned}$$

Die Bytefolge $const \in B^{16}$ besitzt dabei stets den Wert

$$const_i = \begin{cases} (0,0,1,0,1,0,0,0) \text{ für } i = 12, \\ (0,0,0,0,0,0,0,0) \text{ sonst} \end{cases}$$

mit $i \in \mathbb{N}_0,\ 0 \leq i \leq 15$.

4.1.5. Nachricht 4

Die letzte Nachricht bei einem erfolgreichen Handshake ist die Bestätigung der SPS, dass die Funktion “SetMultipleVariables” erfolgreich ausgeführt wurde. Nach dieser Nachricht ist der Handshake abgeschlossen und die weitere Kommunikation findet mittels des Nachrichtentyps “v3” und damit integritätsgeschützt statt.

4.1.6. Zusammenfassung

Die Kommunikation zwischen Client und SPS im Handshake, sowie die mathematischen Operationen, die dabei durchgeführt werden, ist in Abbildung 4.3 dargestellt. Dabei beschränkt sich die Darstellung der Kommunikation auf die Inhalte der Nachrichten, die für die mathematischen Verfahren nötig sind.

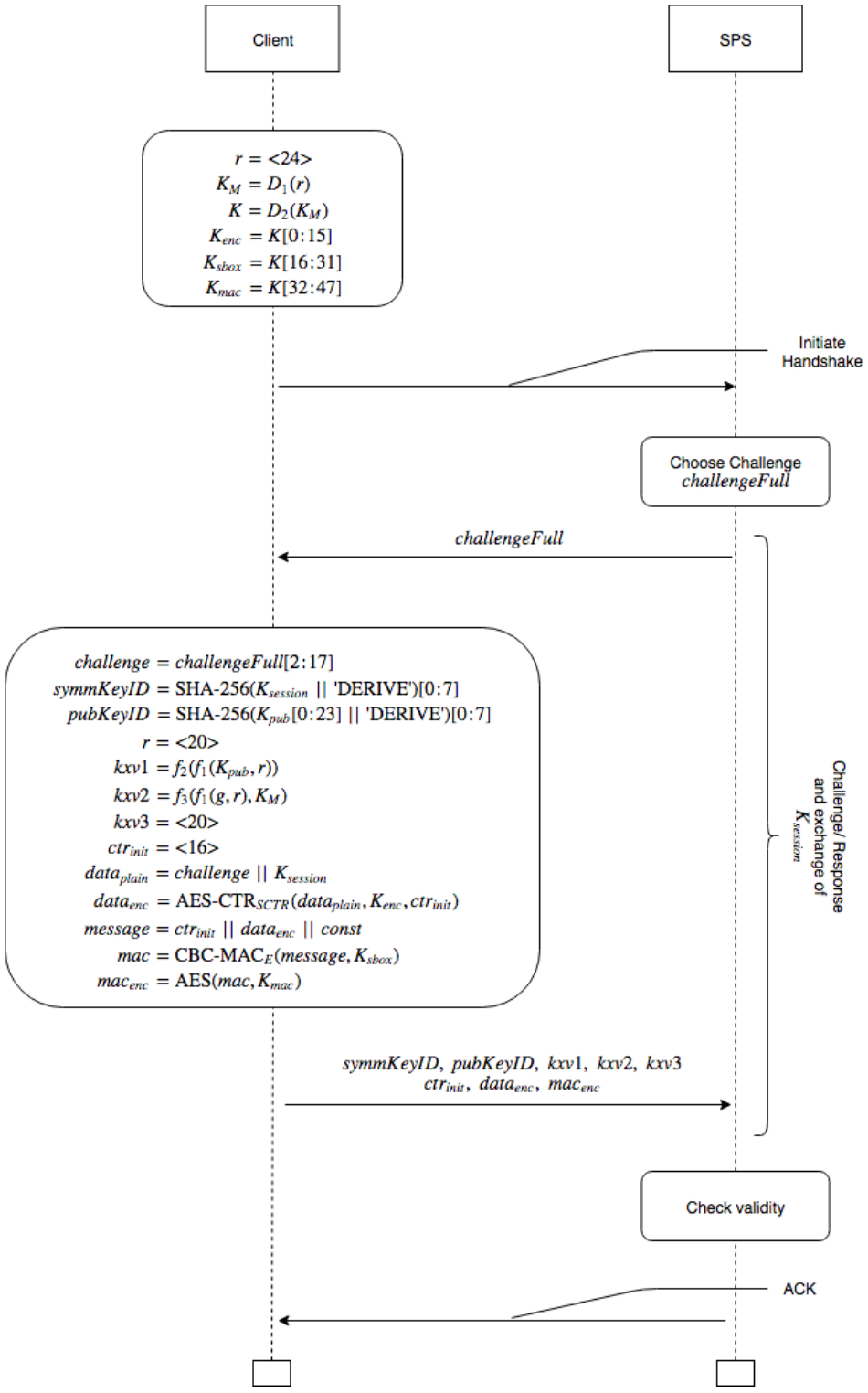

Abbildung 4.3.: Berechnungen und Kommunikation im Handshake

4.2. Integritätsgeschützte Kommunikation

Die Integrität der Kommunikation zwischen Client und PLC ist, nach erfolgreicher Durchführung des Handshakes, durch eine 32 Byte lange kryptographische Prüfsumme $mac \in B^{32}$ geschützt. In diese fließen die gesamten Daten $message \in B^n$ mit $n \in \mathbb{N}$ aus dem Bereich "Data" des S7CommPlus Pakets, sowie dem im Handshake ausgetauschten Session-Key $K_{session}$ und der Challenge *challenge* ein. Dabei wird die Prüfsumme wie folgt berechnet

$$\begin{aligned} data_{challenge} &= \text{Func}(challenge) \\ message_{challenge} &= data_{challenge} \mid\mid challenge \\ K &= \text{HMAC-SHA-256}(message_{challenge}, K_{session})[0:23] \\ mac &= \text{HMAC-SHA-256}(message, K) \end{aligned}$$

mit

$$\text{Func} : B^{16} \rightarrow B^8$$

Die Funktion Func ist verschleiert.

4.3. Client Authentifizierung

Über das Setzen von Zugriffsschutzstufen ist es möglich, den Zugriff auf die SPS zu regulieren. Dabei definiert jede Schutzstufe, welche Aktionen auf der SPS durchgeführt werden dürfen, ohne dass sich der Client zuvor authentifizieren muss. Nur falls eine Authentifizierung zum Durchführen einer Aktion notwendig ist, wird diese Phase durchlaufen. Eine Übersicht über die Zugriffsschutzstufen und die in den Zugriffsschutzstufen erlaubten Aktionen ist in Tabelle 4.1 zu finden. Dabei ist unter "Unauthentifiziertes Lesen" das Lesen von Daten und Variablen ohne vorausgegangene Authentifizierung zu verstehen. "Unauthentifiziertes Schreiben" meint die Möglichkeit des Schreibens von Variablen und Daten ohne vorherige Authentifizierung. Ein Häkchen in einer Spalte bedeutet, dass eine Authentifizierung in der entsprechenden Zugriffsschutzstufe nicht nötig ist. Das eingeklammerte Häkchen meint, dass lesender Zugriff auf die Diagnosedaten ohne Authentifizierung möglich ist, Zugriff auf andere Daten allerdings nicht. Die Standardeinstellung der SPS ist "Vollzugriff". Über das TIA-Portal kann die aktuelle Schutzstufe über das setzen eines Passworts $password \in B^n$, $n \in \mathbb{N}$ frei gewählt werden.

Ist eine Clientauthentifizierung notwending wird diese mittels eines Challenge-Response Verfahrens innerhalb von vier Nachrichten durchgeführt. Auf die einzelnen Nachrichten soll im Folgenden näher eingegangen werden.

ZS	Name	Unauth. Lesen	Unaut. Schreiben
1	Vollzugriff	✓	✓
2	Lesezugriff	✓	✗
3	HMI-Zugriff	(✓)	✗
4	Kein Zugriff	✗	✗

Tabelle 4.1.: PLC Zugriffsschutzstufen

4.3.1. Nachricht 1

Der Client stößt den Authentifizierungsmechanismus an, indem er mittels einer Request-Nachricht mit der Funktion “GetVarSubStreamed” die Variable mit der ID 303 “Server-SessionChallenge” anfordert.

4.3.2. Nachricht 2

Die SPS antwortet mit einer Response-Nachricht mit der Funktion “GetVarSubStreamed” und der 20 Byte langen Challenge $challenge_{auth} \in B^{20}$.

4.3.3. Nachricht 3

Der Client authentifiziert sich dann mit einer Request-Nachricht mit der Funktion “SetVarSubStreamed”, indem er die Variable mit der ID 1846 auf den Wert $data_{auth} \in B^{32}$ setzt. Diesen Wert berechnet er dabei aus der im Handshake verwendeten Challenge $challenge$, der $challenge_{auth}$, dem im Handshake ausgetauschten Session-Key $K_{session}$ und dem Passwort $password$ wie folgt:

$$
\begin{aligned}
message &= \text{SHA-1}(password) \mid\mid challenge_{auth} \\
data_{challenge} &= \text{Func}(challenge) \\
message_{challenge} &= data_{challenge} \mid\mid challenge \\
K &= \text{HMAC-SHA-256}(message_{challenge}, K_{session})[0:23] \\
K_{auth} &= \text{SHA-256}(K \mid\mid \text{'MISTRUST'})[4:27] \\
data_{auth} &= \text{HMAC-SHA-256}(message, K_{auth})
\end{aligned}
$$

Die hier verwendete Funktion Func ist die selbe verschleierte Funktion, wie die die bei der Berechnung der Prüfsumme zur integritätsgeschützten Kommunikation verwendet wird.

$$\text{Func} : B^{16} \rightarrow B^{8}$$

4.3.4. Nachricht 4

Über eine Response-Nachricht mit der Funktion “SetVarSubStreamed” wird der Erfolg oder Misserfolg des Authentifizierungsversuchs signalisiert.

4.3.5. Zusammenfassung

Die Kommunikation zwischen Client und SPS zur Client-Authentifizierung, sowie die mathematischen Operationen, die dabei durchgeführt werden, ist in Abbildung 4.4 dargestellt. Dabei beschränkt sich die Darstellung der Kommunikation auf die Inhalte der Nachrichten, die für die mathematischen Verfahren nötig sind.

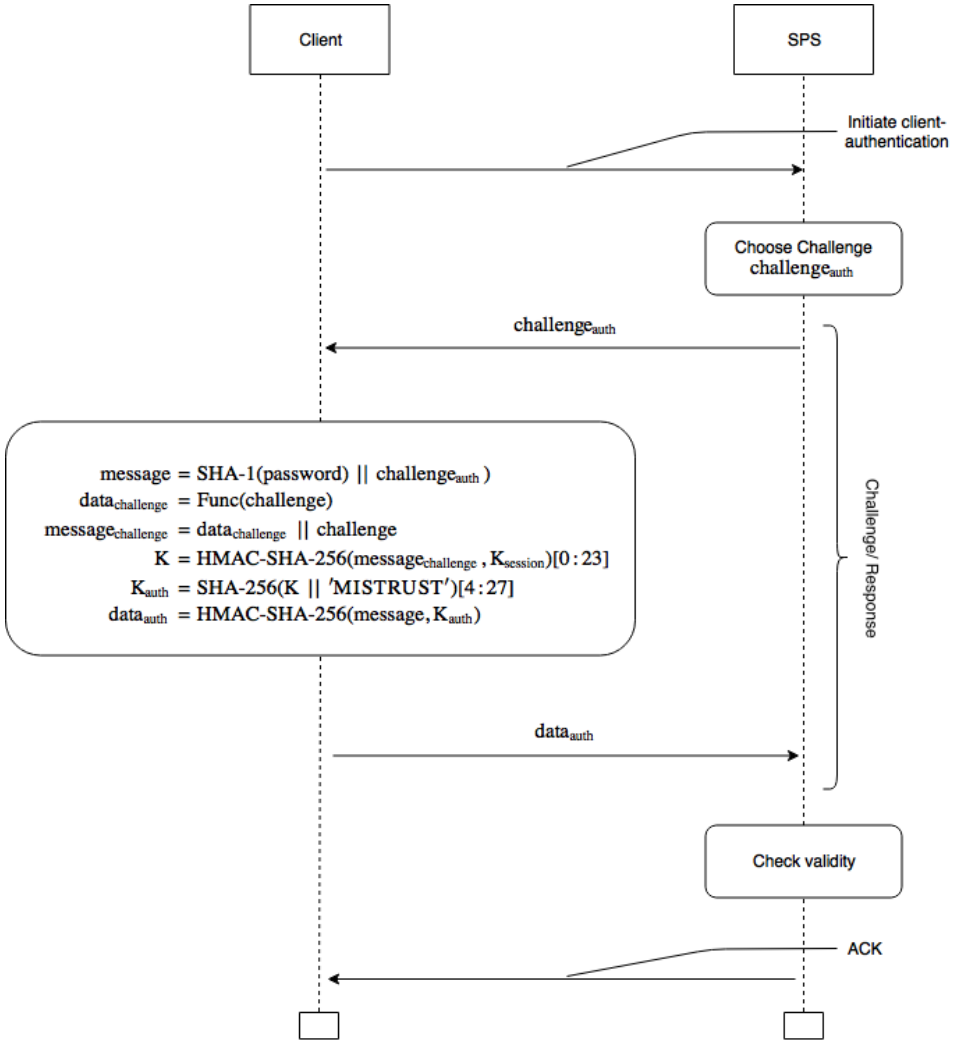

Abbildung 4.4.: Berechnungen und Kommunikation zur Client-Authentifizierung

5. Evaluation

5.1. Client-Authentifizierung

Für das Ausnutzen der im Folgenden beschriebenen Schwachstelle ist es notwendig, dass ein Angreifer aktiver Man in the middle (MITM) in der Verbindung zwischen TIA-Portal und SPS ist. Dies ist nach derzeitigen Kenntnisstand[1] allerdings nicht möglich, wodurch die Schwachstelle zurzeit lediglich theoretischer Natur ist.

Die Client-Authentifizierung im Protokoll S7CommPlus ist ein von der Berechnung der integritätsschützenden Prüfsumme unabhängiger Prozess. Dies führt dazu, dass ein aktiver MITM auch ohne Kenntnis des Passworts die Identität des Clients übernehmen kann. Dazu muss der echte Client einen Authentifizierungsprozess anstoßen, dessen Nachrichten der MITM weiterleitet und die Prüfsummen austauscht. Der MITM ist danach gegenüber der SPS authentifiziert, ohne das Passwort selbst zu kennen. Dies könnte durch eine Kopplung der Authentifizierung an die Prüfsumme verhindert werden. Dazu müsste das Passwort bei entsprechender Zugriffschutzstufe mit in die Berechnung der Prüfsumme einfließen. So würde auch die Authentizität der Nachrichten, bei denen die Prüfsumme auf diese Weise berechnet wird, sichergestellt und der oben genannte Angriffsweg unmöglich gemacht.

5.2. AES Komplexitätreduktion

Die Menge $K = \{D_2(D_1(r))[0:15] \mid r \in B^{24}\}$ aus der der Encryption-Key K_{enc} gewählt wird umfasst maximal 2^{112} Schlüssel. Der Encryption-Key verschlüsselt den Session-Key, der für die integritätsgeschützte Kommunikation, sowie die clientseitige Authentifizierung vonnöten ist. Der Grund dafür, dass die Schlüsselmenge maximal 2^{112} anstelle von 2^{128} Schlüssel enthält ist der Art und Weise geschuldet, wie der Encryption-Key über die Key-Derivation-Funktionen D_1 und D_2 (zu D_1 und D_2 siehe Unterabschnitt 4.1.1) generiert wird.

[1] Stand 2.10.2019

Sei

$$D_{intern} : B^8 \to B^{24}$$

und

$$\begin{aligned} D_2' : B^{60} &\to B^{16} \\ K_M &\mapsto K_{enc} = D_2(K_M)[0\!:\!15] \end{aligned}$$

Die Analyse entsprechender Codestellen im TIA-Portal zeigt, dass gilt

$$D_1(r_1 \mid\mid r_2 \mid\mid r_3) = D_{intern}(r_1) \mid\mid D_{intern}(r_2) \mid\mid D_{intern}(r_3)[0\!:\!11]$$

mit $r_1, r_2, r_3 \in B^8$. Durch eine empirische Analyse von D_2 lässt sich über D_2' folgende Aussage treffen:

$$\text{(I)} \;\; \forall_{a,b \in B^{60}} \; D_2'(a) = D_2'(b) \Leftrightarrow a_i = b_i, \;\; i \in \{0, .., 25, 28, 29, 32, 33\}$$

Diese Erkenntniss wird gewonnen, in dem eine zufällige 60 Byte lange Bytefolge gewählt wird. Von dieser werden 60 weitere Bytefolgen abgeleitet, bei denen jeweils ein einziges, aber immer ein anderes Byte auf einen festen Wert gesetzt wird. Diese 61 Bytefolgen werden sukzessive als Eingabe für D_2' gewählt. Dabei wird verglichen, ob die Ergebnisse der veränderten Bytefolgen von dem Ergebnis der unveränderten Bytefolge abweichen. Ist dies nicht der Fall, so wird angenommen, dass das in der konkret verwendeten Bytefolge veränderte Byte keinen Einfluss auf das Ergebnis hat. Dieser Vorgang wird wiederholt Durchgeführt, um fälschlicherweise als irrelevant angenomme Bytes in der Eingabe dennoch als relevant für die Berechnung der Ausgabe erkennen zu können.

Es gilt

$$\begin{aligned} |K| &= |\{D_2(D_1(r))[0\!:\!15] \mid r \in B^{24}\}| \\ &= |\{D_2'(D_1(r)) \mid r \in B^{24}\}| \\ &= |\{D_2'(D_{intern}(r_0) \mid\mid D_{intern}(r_1) \mid\mid D_{intern}(r_2)[0\!:\!11]) \mid r_0, r_1, r_2 \in B^8\}| \\ &= |\{D_2'(D_{intern}(r_0) \mid\mid (b_0, .., b_{23}) \mid\mid (c_0, .., c_{11})) \mid (b_0, .., b_{23}) = D_{intern}(r_1), \\ &\qquad\qquad (c_0, .., c_{11}) = D_{intern}(r_2)[0\!:\!11]), \\ &\qquad\qquad r_0, r_1, r_2 \in B^8\}| \\ &\overset{\text{(I)}}{=} |\{D_2'(D_{intern}(r_0) \mid\mid (b_0, b_1, 0, 0, b_4, b_5, 0, 0, b_8, b_9, 0, .., 0) \mid\mid (0, 0, .., 0)) \mid \\ &\qquad\qquad (b_0, .., b_{23}) = D_{intern}(r_1), \\ &\qquad\qquad (c_0, .., c_{11}) = D_{intern}(r_2)[0\!:\!11], \\ &\qquad\qquad r_0, r_1, r_2 \in B^8\}| \\ &\leq |\{D_{intern}(r_0) \mid r_0 \in B^8\}| \\ &\quad \cdot |\{(b_0, b_1, 0, 0, b_4, b_5, 0, 0, b_8, b_9, 0, .., 0) \mid (b_0, .., b_{23}) = D_{intern}(r_1), \; r_1 \in B^8\}| \\ &\quad \cdot |\{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}| \\ &\leq (2^8)^8 * (2^8)^6 * 1 = 2^{112} \end{aligned}$$

Die Mächtigkeit der Menge K und damit die Anzahl der möglichen durch dieses Verfahren generierten Schlüssel beträgt damit maximal 2^{112}. Sollte die tatsächliche Mächtigkeit von K nahe diesem Maximalwert von 2^{112} liegen, so wären praktische Angriffe auf die Verschlüsselung auf diesem Weg nicht möglich. Allerdings stellt 2^{112} nur eine obere Schranke dar und die tatsächliche Mächtigkeit von K könnte noch weit darunter liegen.

5.3. TIA-Portal unabhängige Kommunikation

Im Handshake wird der Master-Key K_M ausgetauscht. Dieser Schlüssel dient der Erzeugung dreier weiterer Schlüssel, die für den Austausch des Session-Keys vonnöten sind. Der Austausch des Master-Schlüssels konnte aufgrund der verwendeten Codeobfuscation-Techniken nicht in Gänze nachvollzogen werden. Dies hat jedoch nicht zur Folge, dass eine TIA-Portal-unabhängige Kommunikation mit der SPS über das S7CommPlus Protokoll unmöglich ist. Um trotzdem unabhängig vom TIA-Portal mit der Anlage kommunizieren zu können, müssen zwei Hürden überwunden werden. Erstens müssen für jeden SPS-abhängigen öffentlichen Schlüssel und statischen Wert g einmalig die Key Exchange Vektoren berechnet werden. Dies muss für einen bekannten Master-Key durch das TIA-Portal geschehen. Nach der Berechnung der Werte lassen sich diese wiederverwenden. Die Key Exchange Vektoren für den öffentlichen Schlüssel und Wert g der S7-1500 (siehe A.1 und A.2) und einen Master-Key bestehend aus 60 Null-Bytes sind in 5.1 aufgeführt.

0000	1a	8a	93	fa	2e	14	d7	91	20	45	9d	6f	80	d5	0c	96
0010	84	31	43	bd	b5	78	80	1b	af	46	88	0d	68	0f	99	df
0030	97	c8	ae	d9	76	7b	b1	98	00	00	00	00	00	00	00	00
0040	00	00	00	00	00	00	00	00	00	00	00	00				

kxv1, kxv2, kxv3,

Abbildung 5.1.: KXV für S7-1500 Public Key und g und Masterkey, der aus Null-Bytes besteht

Zweitens muss die Funktion Func aus dem TIA-Portal extrahiert werden, da diese sowohl bei den zum Integritäts- als auch bei den zum Zugriffsschutz eingesetzten Verfahren benötigt wird. Eine Extraktion ist deshalb notwendig, da die Funktion durch Codeobfuscation Techniken nicht identifiziert oder nachimplementiert werden konnte. In einer TIA-Portal-unabhängigen Kommunikation stellt sie daher eine Blackbox-Funktion dar.

Mittels der Verfahren, wie sie im Zuge der Analyse beschrieben wurden, den vorab berechneten Key Exchange Vektoren und der extrahierten verschleierten Funktion "Func" kann nun eine Kommunikation mit der SPS stattfinden. Die Kommunikation ist dabei nicht weiter eingeschränkt.

Angesichts der zwei Hürden, die zu nehmen sind, um ohne das TIA-Portal über das S7CommPlus Protokoll mit einer SPS zu kommunizieren, lässt sich sagen, dass eine vom TIA-Portal gänzlich unabhängige Kommunikation bei derzeitigem Wissensstand nicht möglich ist. Grund dafür sind Codepassagen, deren Analyse durch Codeobfuscation-Techniken erheblich erschwert werden.

6. Weitere Forschungsfragen

Während der Analyse der kryptographischen Verfahren, die im S7CommPlus Protokoll verwendet wurden, sind einige weitere Forschungsfragen aufgekommen. Insbesondere von Interesse sind die folgenden drei.

6.1. Key Exchange Verfahren

Während des Handshakes wird über die Key Exchange Vektoren im "SecurityKey-EncryptedKey"-Blob ein Schlüssel ausgetauscht, von dem der Schlüssel zum Dechiffrieren des Session-Keys abgeleitet werden kann. Dieser Schlüsselaustausch konnte nicht nachvollzogen werden, da ein Großteil des Assembly, der dafür zuständig ist, verschleiert wurde. Hier ist es sicher erstrebenswert Aufwand zu betreiben, um das Verfahren dennoch nachvollziehen zu können. Besonders deshalb, weil die Sicherheit, sowohl der Client-Authentifizierung, als auch der integritätsgeschützten Kommunikation letztendlich von der Sicherheit dieses Schlüsselaustausches abhängt.

6.2. Deobfuscation

Einige sicherheitsrelevante Codepassagen des TIA-Portals wurden durch Codeobfuscation-Techniken verschleiert. Dazu zählen zum Beispiel Funktionen, die bei der Schlüsselerzeugung verwendet werden, genauso wie Funktionen, die beim Schlüsselaustausch zum Einsatz kommen. Das Überführen des Codes in eine für Menschen besser lesbare Form ist sowohl für eine Sicherheitsanalyse des Codes interessant, als auch, um das Protokoll in Gänze zu verstehen.

6.3. Sicherheitsanalyse

Bisher liegen sehr wenige Informationen über die Sicherheit des S7CommPlus Protokolls, sowie dessen Implementierung im TIA-Portal bzw. auf SPS-Seite vor. Diese Wissenslücke durch eine umfassende Sicherheitsanalyse zu schließen ist besonders für Unternehmen,

die entsprechende Anlagen einsetzen, von großem Interesse. Eine solche Analyse sollte neben dem Protokoll auch die Implementierung des Protokolls mit einschließen. Gerade auf Grund der Komplexität des Protokolls ist eine solche Analyse erstrebenswert.

7. Fazit

Bei der Analyse wurden Verfahren zum Integritäts- und zum Zugriffsschutz im S7CommPlus Protokoll identifiziert. Diesen voraus geht der Austausch eines Session-Keys und einer Challenge, die für die Verfahren jeweils benötigt werden. Das Ziel der Analyse bestand nach Identifikation der relevanten Aspekte des Protokolls darin, diese zu verstehen und beschreiben zu können. Größtenteils wurde das Ziel erreicht. So konnten, bis auf eine verschleierte Funktion "Func" die Verfahren zum Schutz der Integrität und zum Schutz vor unautorisiertem Zugriff beschrieben werden. Ebenfalls konnte der Austausch des Session-Keys zu einem Großteil dargelegt werden. Die Aspekte des Protokolls, die nicht in Gänze nachvollzogen werden konnten, die Funktion "Func" und ein Teil des Schlüsselaustausches, wurden empirisch untersucht und so detailliert wie möglich beschrieben. Dass genau diese beiden Teile nicht genauso umfassend beschrieben werden konnten wie die anderen, ist damit begründet, dass die Analyse entsprechender Codepassagen durch Codeobfuscation-Techniken erheblich erschwert wurde.

Der Einsatz von Codeobfuscation-Techniken ist mit Blick auf die Sicherheit des Protokolls durchaus fragwürdig. Die Wirksamkeit der integritäts- und authentizitätsschützenden Maßnahmen, die im S7CommPlus Protokoll ergriffen wurden, hängen letztendlich alle von der Sicherheit des Schlüsselaustausches ab. Da wesentliche Passagen des Codes, die dafür zuständig sind, allerdings verschleiert wurden (Code Obfuscation), ist es schwer nachzuvollziehen, wie dieser genau abläuft. Insbesondere wird dadurch auch eine umfassende Sicherheitsanalyse des Schlüsselaustausches deutlich erschwert. Dabei gibt es lediglich zwei mögliche Szenarien. Im ersten ist die Sicherheit des Schlüsselaustausches hoch, damit wären die Codeobfuscation-Techniken überflüssig, weil sie die Sicherheit nicht noch weiter erhöhen. Im zweiten Szenario ist die Sicherheit des Schlüsselaustausches niedrig, womit die gesamte Sicherheit, die das Protokoll bietet, von den Codeobfuscation-Techniken abhängen würde. Damit würden die Codeobfuscation-Techniken entweder keinen Vorteil oder sogar einen Nachteil in Bezug auf die Sicherheit des Protokolls darstellen.

Literaturverzeichnis

[1] Dipl.-Betriebswirtin (FH) Michaela Rothhöft. Marktstudie sps-systeme. http://www.marktstudien.org/pdf/ergebnisauszug_sps.pdf, 2014. Abgerufen am 08.01.2018.

[2] Reinhold Schäfer. Sps bleibt für die automatisierung unverzichtbar. https://www.maschinenmarkt.vogel.de/sps-bleibt-fuer-die-automatisierung-unverzichtbar-a-402440/index3.html, 2013. Abgerufen am 08.01.2018.

[3] Maik Brüggemann. ICShell. https://code.opensource-security.de/brueggemann/icshell, 2016. Abgerufen am 05.03.2018.

[4] Cheng Lei, Li Donghong, and Ma Liang. The spear to break the security wall of s7commplus. 2017.

[5] Thomas Wiens. S7comm wireshark dissector plugin. https://sourceforge.net/projects/s7commwireshark/. Aberufen am 08.01.2018.

[6] Siemens AG. Unser schnellster Controller für die Automatisierung. https://www.siemens.com/global/de/home/produkte/automatisierung/systeme/industrie/sps/simatic-s7-1500.html. Abgerufen am 14.02.2018.

[7] Connection Oriented Transport Protocol. ISO Standard 8073, International Organization for Standardization, 1988.

[8] Marshall T. Rose and Dwight E. Cass. ISO Transport Service on top of the TCP. RFC 1006, May 1987.

[9] Wikipedia contributors. Variable-Length Quantity. https://en.wikipedia.org/wiki/Variable-length_quantity, 2017. Abgerufen am 26.02.2018.

[10] 3rd D. Eastlake and T. Hansen. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234, Mai 2011.

[11] Wireshark Foundation. Network protocol analyzer. https://www.wireshark.org/#. Abgerufen am 20.02.2018.

[12] dnSpy Project. .NET debugger and assembly editor. https://github.com/0xd4d/dnSpy. Abgerufen am 20.02.2018.

[13] x64dbg Project. An open-source x64/x32 debugger for windows. `https://x64dbg.com/#start`. Abgerufen am 20.02.2018.

A. Anhang

A.1. S7-1500 Public Connection Key

0000	84	56	A2	69	96	12	22	16	C9	21	C5	71	FF	11	E0	BE
0010	FA	FD	B1	D7	0B	5D	4B	C8	39	0F	5B	0C	C2	73	EC	14
0020	2A	03	F2	A0	4E	6F	15	93								

A.2. S7-1500 g

0000	8E	0C	9D	A3	07	17	47	85	32	61	7A	4A	D7	58	6E	84
0010	E2	D8	2E	7D	3B	67	8C	CD	13	F1	BC	88	F7	E8	7F	6C
0020	FE	4C	69	9D	E9	35	A7	57								

A.3. Siemens-AES-Counter-Inkrementierung (Python)

```
def SCTR(self, ctr):
    endianness = 'little'
    v0  = int.from_bytes(ctr[0:4], endianness)
    v1  = int.from_bytes(ctr[4:8], endianness)
    v2  = int.from_bytes(ctr[8:12], endianness)
    v3  = int.from_bytes(ctr[12:16], endianness)

    v0_out  = ((v1 << 31) % 0x100000000) | v0 >> 1
    v1_out  = ((v2 << 31) % 0x100000000) | v1 >> 1
    v2_out  = ((v3 << 31) % 0x100000000) | v2 >> 1
    t0      = v0 & 0x00000001
    t1      = 0xffffffff
    if t0 == 0:
        t1  = 0
    t2        = v3
    t1       &= 0xe1000000
    t2       >>= 1
    t2       ^= t1
    v3_out    = t2

    ctr_next    = (v0_out).to_bytes(4, endianness) + \
                  (v1_out).to_bytes(4, endianness) + \
                  (v2_out).to_bytes(4, endianness) + \
                  (v3_out).to_bytes(4, endianness)

    return ctr_next
```

A.4. Auf einer SBOX basierende Verschlüsselungsfunktion (Python)

```
def val(barr):
    """Bytes to int value."""
    val = 0
    for b in barr:
        val <<= 8
        val |= b
    return val

def tobytes(val):
    """Int value to byte."""
    ret = b''
    for i in range(0, 4):
        ret = bytes([val & 0xff]) + ret
        val >>= 8
    return ret

def reverse(string):
    return string[::-1]

def ls32(val, bits):
    """bits wide left shift of 32bit integer."""
    val = struct.unpack("<I", struct.pack(">I", val))[0]
    val >>= bits
    val = struct.unpack("<I", struct.pack(">I", val))[0]
    return val

def sr32(val, bits):
    """bits wide right shift of 32bit integer."""
    val = struct.unpack("<I", struct.pack(">I", val))[0]
    val <<= bits
    val &= 0xffffffff
    val = struct.unpack("<I", struct.pack(">I", val))[0]
    return val

def create_sbox_byte(sbox_key):
    """Keyed creation of the sbox needed for the encryption alg."""
    sbox = bytearray([0] * 4096)
    sbox[0:16] = b'\x00' * 16
    sbox[16:32] = sbox_key
    e = 'little'

    i = 2
    while True:
        ind_0   = i*16
        ind_1   = ind_0 + 12
        ind_3   = (i >> 1) << 1 # setzte letztes bit 0
```

```
ind_4   = ind_3 * 8 + 12
j       = 3
ind_5   = ind_4

while True:
    val_0_0     = int.from_bytes(sbox[ind_4-4:ind_4], e) >> 31 % 0x100000000
    val_0_1     = int.from_bytes(sbox[ind_4:ind_4+4], e) * 2 % 0x100000000
    val_0       = val_0_0 | val_0_1
    ind_6       = ind_1
    j           -= 1
    ind_1       -= 4
    ind_4       -= 4
    sbox[ind_6:ind_6+4] = (val_0).to_bytes(4, e)
    if j <= 0:
        break

val_1_0 = int.from_bytes(sbox[ind_3*8:ind_3*8+4], e)
val_1   = (2 * val_1_0) % 0x100000000
sbox[ind_0:ind_0+4] = (val_1).to_bytes(4, e)

val_t0  = int.from_bytes(sbox[ind_5:ind_5+4], e)
if val_t0 & 0x80000000 != 0:
    val_t1  = int.from_bytes(sbox[ind_0+4:ind_0+8], e)
    val_t2  = int.from_bytes(sbox[ind_0:ind_0+4], e)
    val_t1 ^= 0x01
    val_t2 ^= 0x8005
    sbox[ind_0+4:ind_0+8]   = (val_t1).to_bytes(4, e)
    sbox[ind_0:ind_0+4]     = (val_t2).to_bytes(4, e)

if i > 1:
    ind_7   = 24
    ind_8   = ind_0 + 24
    k       = i - 1
    while True:
        val_2_0 = int.from_bytes(sbox[ind_7-8:ind_7-4], e)
        val_2_1 = int.from_bytes(sbox[ind_0:ind_0+4], e)
        val_2   = val_2_0 ^ val_2_1
        sbox[ind_8-8:ind_8-4] = (val_2).to_bytes(4, e)

        val_3_0 = int.from_bytes(sbox[ind_7-4:ind_7], e)
        val_3_1 = int.from_bytes(sbox[ind_0+4:ind_0+8], e)
        val_3   = val_3_0 ^ val_3_1
        sbox[ind_8-4:ind_8] = (val_3).to_bytes(4, e)


        val_4_0 = int.from_bytes(sbox[ind_0+8:ind_0+12], e)
        val_4_1 = int.from_bytes(sbox[ind_7:ind_7+4], e)
        val_4   = val_4_0 ^ val_4_1
        sbox[ind_8:ind_8+4] = (val_4).to_bytes(4, e)
```

```
                ind_7   += 16

                val_5_0 = int.from_bytes(sbox[ind_7-12:ind_7-8], e)
                val_5_1 = int.from_bytes(sbox[ind_0+12:ind_0+16], e)
                val_5   = val_5_0 ^ val_5_1

                ind_8   += 16
                k -= 1

                sbox[ind_8-12:ind_8-8] = (val_5).to_bytes(4 ,e)

                if k == 0:
                    break

            i *= 2
            if i >= 0x81:
                break

    return sbox

def btov(bsbox):
    """Bytes to vector."""
    vsbox = []
    for i in range(0, len(bsbox) // 16):
        vector = bytearray()
        for j in range(0, 16):
            b = bsbox[i*16+j]
            vector.append(b)
        vsbox.append(vector)
    return vsbox

def create_sbox(sbox_key):
    """Create sbox."""
    byte_sbox = create_sbox_byte(sbox_key)
    vector_sbox = btov(byte_sbox)
    return vector_sbox

def encrypt_unknown(plain, sbox_key):
    """Encrypt plain data using a keyed sbox."""
    sbox = create_sbox(sbox_key)

    # 16 byte vector -> 4 * uint32
    v0 = val(reverse(plain[0:4]))
    v1 = val(reverse(plain[4:8]))
    v2 = val(reverse(plain[8:12]))
    v3 = val(reverse(plain[12:16]))

    t0 = 0
    t1 = 0
    t2 = 0
```

```
t3 = 0

h0 = 0
h1 = 0
h2 = 0
h3 = 0

g0 = 0
g1 = 0
g2 = 0
g3 = 0

e0 = 0
e1 = 0
e2 = 0
e3 = 0

k0 = 0
k1 = 0
k2 = 0
k3 = 0

for i in [24, 16, 8]:
    l0 = (v0 >> i) & 0xff
    l1 = (v1 >> i) & 0xff
    l2 = (v2 >> i) & 0xff
    l3 = (v3 >> i) & 0xff

    t0 = h0 ^ val(sbox[l0][0:4])
    t1 = h1 ^ val(sbox[l0][4:8])
    t2 = h2 ^ val(sbox[l0][8:12])
    t3 = h3 ^ val(sbox[l0][12:16])

    g0 = t1 ^ val(sbox[l1][0:4])
    g1 = t2 ^ val(sbox[l1][4:8]) ^ val(sbox[l2][0:4])
    g2 = t3 ^ val(sbox[l1][8:12])
    g3 = k3 ^ val(sbox[l1][12:16])

    e0 = k1 ^ val(sbox[l3][12:16])
    e1 = g2 ^ val(sbox[l2][4:8]) ^ val(sbox[l3][0:4])
    e2 = g3 ^ val(sbox[l2][8:12]) ^ val(sbox[l3][4:8])
    e3 = k2 ^ val(sbox[l2][12:16]) ^ val(sbox[l3][8:12])

    k0 = ls32(e0, 24) | k0 >> 8
    k1 = ls32(e3, 24) | e0 >> 8
    k2 = ls32(e2, 24) | e3 >> 8
    k3 = ls32(e1, 24) | e2 >> 8

    h0 = t0 >> 8
    h1 = ls32(t0, 24) | g0 >> 8
```

```
    h2 = ls32(g0, 24) | g1 >> 8
    h3 = ls32(g1, 24) | e1 >> 8

l0 = v0 & 0xff
l1 = v1 & 0xff
l2 = v2 & 0xff
l3 = v3 & 0xff

d0 = h0 ^ val(sbox[l0][0:4])
d1 = h1 ^ val(sbox[l0][4:8])
d2 = h2 ^ val(sbox[l0][8:12])
d3 = h3 ^ val(sbox[l0][12:16])

f0 = d1 ^ val(sbox[l1][0:4])
f1 = d2 ^ val(sbox[l1][4:8])
f2 = d3 ^ val(sbox[l1][8:12])
f3 = k3 ^ val(sbox[l1][12:16])

m0 = f1 ^ val(sbox[l2][0:4])
m1 = f2 ^ val(sbox[l2][4:8]) ^ val(sbox[l3][0:4])
m2 = f3 ^ val(sbox[l2][8:12]) ^ val(sbox[l3][4:8])
m3 = k2 ^ val(sbox[l2][12:16]) ^ val(sbox[l3][8:12])

u = k1 ^ val(sbox[l3][12:16])
v = ls32((ls32(k0, 13) ^ k0), 17) ^ m2 ^ k0

out0 = sr32((sr32(v, 13) ^ v), 2) ^ d0 ^ v
out1 = ls32((ls32(v, 13 ) ^ v), 17) ^ sr32((sr32(m3, 13) ^ m3), 2) ^ f0 ^ v ^ m3
out2 = ls32((ls32(m3, 13) ^ m3), 17) ^ sr32((sr32(u, 13) ^ u), 2) ^ m0 ^ m3 ^ u
out3 = ls32((ls32(u, 13) ^ u), 17) ^ sr32((sr32(k0, 13) ^ k0), 2) ^ m1 ^ u ^ k0

encrypted = tobytes(out0) + tobytes(out1) + tobytes(out2) + tobytes(out3)
return encrypted
```