You Are What You Attack: Breaking the Cryptographically Protected S7 Protocol

Preprint · January 2023

DOI: 10.13140/RG.2.2.25549.10721/1

2 authors:

Wael Alsabbagh
IHP
18 PUBLICATIONS **57** CITATIONS
SEE PROFILE

Peter Langendoerfer
IHP
290 PUBLICATIONS **1,678** CITATIONS
SEE PROFILE

You Are What You Attack: Breaking the Cryptographically Protected S7 Protocol

Wael Alsabbagh[1,2] and Peter Langendörfer[1,2]
[1]*IHP – Leibniz-Institut für innovative Mikroelektronik, Frankfurt (Oder), Germany*
[2]*Brandenburg University of Technology Cottbus-Senftenberg, Cottbus, Germany*
E-mail: (Alsabbagh, Langendoerfer)@ihp-microelectronics.com

***Abstract*—S7 protocol defines an appropriate format for exchanging messages between SIMATIC S7 PLCs and their corresponding engineering software i.e., TIA Portal. Recently, Siemens has provided its newer PLC models and their proprietary S7 protocols with a very developed and sophisticated integrity check mechanism to protect them from various exploits e.g., replay attacks. This paper addresses exactly this point, and investigates the security of the most developed integrity check mechanism that the newest S7CommPlus protocol version implements. Our results showed that the latest S7 PLC models as well as their related protocols are still vulnerable. We found that adversaries can manipulate two hashes that play a significant role in generating keys and bytes for the encryption processes implemented in the S7CommPlus protocol. This allows to reproduce S7 packets and conduct several attacks that eventually impact the operation of the target PLC and the entire physical process it controls. To validate our findings, we test all the attack scenarios presented in this work on a cryptographically protected S7 PLC from the 1500 family which uses the S7CommPlusV3 protocol.**

***Index Terms*—S7 Protocol, SIMATIC PLCs, Cyber Security, Industrial Control Systems; Replay Attacks;**

I. Introduction

The S7 Protocol serves the exchange of critical information between Programmable Logic Controllers (PLCs) and their Total Integrated Automation (TIA) Portal engineering software. The exchanged messages include network configuration, critical data e.g., the control logic program, diagnostic information, set-point values, etc. between the connected parties. Its core communication follows a "client-server" pattern. Meaning that, the TIA Portal device (client) initiates transactions, and the connected PLC (server) responds by either supplying the requested data or executing certain actions. The newer generations of S7 PLCs i.e., S7-1200 and S7-1500 are provided with a cryptographically protected S7 protocol, called S7CommPlus. This protocol is identified by a unique protocol ID (*0x72*) and has three sub-versions [3] as follows:

- S7CommPlusV1: it is used by the older versions of TIA Portal and only in S7-1200 PLCs. This protocol does not include any integrity protection.

- S7CommPlusV2: it is used in the TIA Portal up to V12 and its PLCs S7-1500 firmware up to 1.5. This protocol is integrity protected and has security features against replay attacks (e.g., Hashed-based Message Authentication Code-Secure Hash Algorithm-256 (HMAC-SHA-256)).

978-1-7281-9023-5/21/$31.00 ©2021 IEEE

- S7CommPlusV3: it is used in the newer versions of TIA Portal i.e., from V13 on, and in the newer S7-1500 PLC firmwares e.g., V1.8, 2.0, etc. This protocol requires that both the TIA Portal and the PLCs to support the features of this protocol. Here, Siemens has improved the integrity mechanism by providing their S7CommPlus protocol with very complex encryption processes. Therefore, it is considered the most secure protocol among the other versions i.e., S7CommPlusV1 and S7CommPlusV2, and it is the focus of our work.

The integrity mechanism implemented in the newest PLC models and the S7CommPlusV3 protocol consists of three main parts : 1) a *Challenge* packet sent from the PLC to the TIA Portal, 2) a *Response* packet sent from the TIA Portal to the PLC, and finally 3) an *Integrity Part* in each S7 function packet sent from the TIA Portal to the PLC and vice versa [3]–[6]. In this work, we show how an attacker could get sufficient information of the integrity mechanism, and disclose vulnerabilities that he can exploit to maliciously craft S7 *Function* packets that could be potentially used in replay attacks. To validate our findings, we perform several attack scenarios on a real hardware SIMATIC S7-1512SP PLC with a firmware V2.9.2, and a TIA Portal software with a firmware V16. The attacks presented in this work include a simple start/stop, unauthorized software and hardware changes to the PLC program, removing the PLC program and Denial of Service (DoS) attacks. All our attack scenarios are network-based and designed with the help of pre-recorded packets that can be used after calculating specific encryption bytes correctly as shown later.

The rest of the paper is organized as follows. We compare our work with related ones in section II. Section III gives an overview of the S7CommPlusV3 protocol while its communication process is illustrated in IV. Section V presents our attack approach, and we conclude this paper in Section VI.

II. Related Work

There are only a few researches that discussed vulnerabilities and security gaps in the S7CommPlus protocol. The first ever work was published in 2016 when Spenneberg et al. [8] introduced a worm that found new vulnerabilities in the S7CommPlus protocol, precisely the S7CommPlusV2. The authors demonstrated a malicious code designed with the help of TIA Portal. The code was first injected into an S7-1200 PLC. After patching the PLC, the worm automatically scanned

the network and connected to other PLCs trying to download the rough code into them. The downside of Spenneberg approach was that the infected PLC is always rebooted during the patch which could eventually pay the legitimate user attention. In 2017, Cheng et al. [6] investigated the new encryption method used in the S7CommPlus protocol. The authors used a reverse engineering technique to analyze the communication process between the PLC and TIA Portal. Afterwards, they presented a spear that can break the security wall of the S7CommPlus protocol that Siemens S7-1200 PLCs utilize. The authors performed two attacks. First they pushed crafted packets into the network to start and stop the PLC remotely. In the second attack scenario, the authors manipulated the input and output values of the victim causing a serious damage for the physical process. However, the work of Cheng lacks detailed information of how the integrity check mechanism works, how the encryption processes are calculated and which bytes can be manipulated by adversaries to conduct successful replay attacks. A research group in 2018, showed that a network based attack e.g., Address Resolution Protocol (ARP) poisoning is feasible in S7-1200 PLCs and their S7CommPlus protocol [9]. The authors found specific 7 bytes, known as "*S7-ACK*", can be exploited for a session stealing and DoS attacks. Biham et al. [3] investigated the security features of both S7CommPlusV2 and S7CommPlusV3 protocols used in both S7-1200 and S7-1500 PLCs. The authors disclosed serious security gaps in the S7CommPlus protocol and performed different exploits against cryptographically protected PLCs. In 2022, Alsabbagh et al. in [4], [5] investigated also the security of S7CommPlusV3 protocol and revealed a few security gaps in the design of this protocol e.g., one-way authentication method between the PLC and TIA Portal software allows attackers to connect to the victim PLC using a TIA Portal software without any efforts. Furthermore, the authors found that the PLC did not check the integrity of all packet attributes. Based on their findings, the authors designed an injection tool, namely *PLCinjector* that allows attackers to establish a connection with a remote PLC and alter the program running in the device. All the aforementioned works [3]–[6], [9] required from adversaries to have already a TIA Portal software installed on the attacker machines. In opposite, we in this work overcome this point by introducing typical replay attack scenarios based on pre-captured packets from old S7 sessions without the need to have a TIA Portal installed on the attacker's machine. Furthermore, our attacks are valid for all S7-1500 PLCs from the same firmware i.e., V2.9.2.

III. S7CommPlusV3 Protocol Background

The S7CommPlusV3 protocol supports various operations that are performed by the TIA Portal software such as start/stop the PLC, download/upload a control program to/from the PLC, read/write values of a control variable, etc. All these operations are translated first by the TIA Portal software to S7CommPlus packets, precisely S7 *Function* packets before they are transmitted to the PLC. Then, the PLC acts upon the messages it receives, executes the control operations, and responds back to the TIA Portal accordingly. The messages are transmitted in context of a session, each has a session ID (chosen by the PLC). Since the S7CommPlus is a cryptographically-protected protocol, each new S7 session established using this protocol begins with a four-message handshake to select the cryptographic attributes of the session, including the protocol version and encryption keys [4]. After the handshake, all messages are integrity protected using a very complex integrity mechanism.

A. *Protocol Structure*

S7CommPlusV3 is a "request-response" protocol. Each message consists of a protocol *Header*, *Data* and a *Trailer* as shown in figure 1.

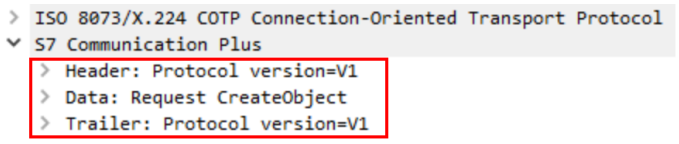

Fig. 1: The Structure of S7CommPlus Protocol

The *Header* and *Trailer* have always the same structure including the following components: 1-byte *Protocol version*, 1-byte *Protocol ID*, , and 2-byte *DATA Length* as shown in figure 2.

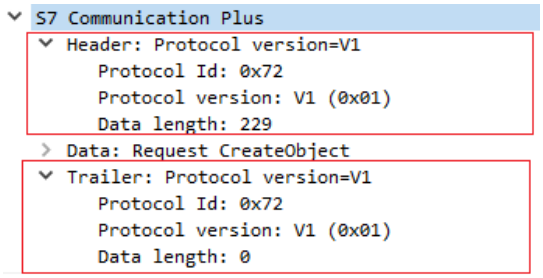

Fig. 2: S7CommPlus Protocol: *Header* and *Trailer* have the same Structure

The Protocol Data Unit (PDU) type determines the S7CommPlus protocol version i.e., V1, V2 or V3 for the value 0x01, 0x02, or 0x03 respectively. In case the PDU type has the value 0x01 or 0x02, this means there is no *Integrity Part* in the *Data* field. In the opposite, an additional *Integrity Part* (see the red block in figure 3) is padded with the *Data* field in case the PDU type is donated with 0x03 as shown in figure 3. In this work, we are only interested in studying the latest S7CommPlus version. Therefore from this point on, all the information provided throughout this paper is related to the S7CommPlusV3 protocol.

The *Data* block is comprised of 14 bytes (see the green block in figure 3). Starting from the top, we see a 1-byte *Opcode* which identifies the purpose of the S7CommPlus packet e.g., *0x31* if the packet is a request, *0x32* if the packet is response or *0x33* if the packet is a notification. After the *Opcode*, we see a 2-byte field that has a fixed value of *0x0000*.

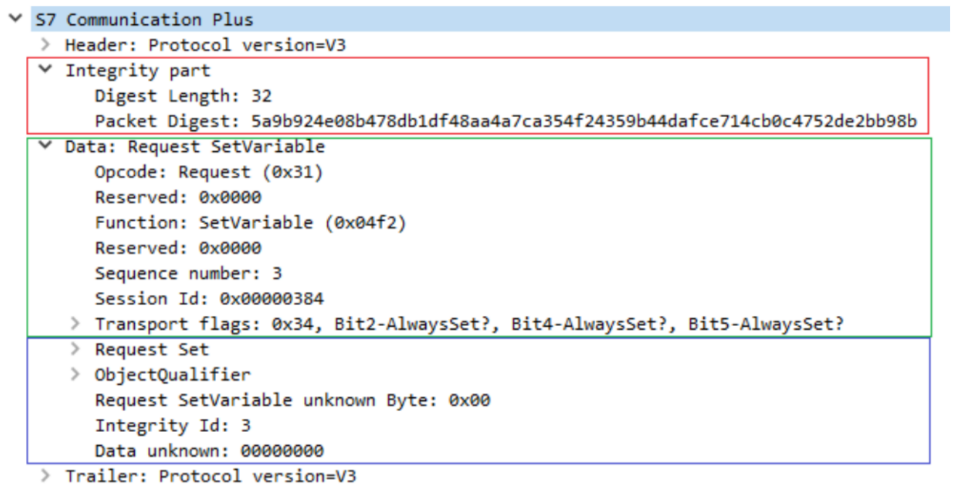

Fig. 3: S7CommPlusV3 protocol - *Data* field Components

Then, there is a 2-byte field, called *Function*, which determines the functionality of the packet e.g., *0x04ca* for *CreateObject*, *0x0542* for *SetMultiVariable*, *0x04f2* for *SetVariable*, etc. In the next field we find again a fixed value of *0x0000* followed by a 2-byte field representing the sequence number of the packet. The *Session ID* length is 4-byte and has always the format of *0x000003xx*. The *xx* in the *Session ID* is a combination of *ObjectID* and *0x80*. Finally, we have the transport flag that is a 1-byte field generated randomly without any use neither in the encryption nor in the authentication methods.

The structure and content of the *Set* block (see the blue block in figure 3) are related to the PDU type and *Opcode*. This block has many diverse types and is quite complex. For more details please see the S7comm Wireshark dissector plugin project [7].

B. Communication Process

The TIA Portal and PLCs exchange four kinds of packets: S7 *Request*, *Challenge*, *Response*, and *Function* packets see figure 4.

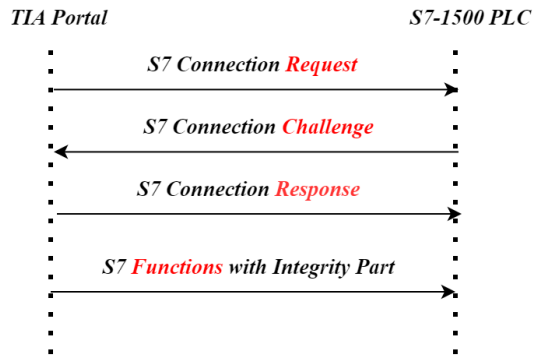

Fig. 4: S7CommPlus Communication Process

As can be seen, at the beginning of each new communication session, the TIA Portal sends an S7 *Request* to establish a connection with the PLC. After the PLC receives the S7 *Request*, it sends a 20-byte array, namely *Challenge*, that significantly differs from a session to another. Those 20 bytes are generated by a hash or pesudo random function. After the TIA Portal receives the *Challenge*, it generates a *Response* that contains among many bytes three interesting blocks: 9-byte, 8-byte and 132-byte blocks respectively see figure 7. These three blocks are checked by the PLC and a Transmission Control Protocol (TCP) packet with a reset flag will be immediately sent from the PLC if the bytes of the blocks are not what the PLC expected, otherwise the communication continues by sending an "*OK*" packet to the TIA Portal. Those bytes are generated by specific algorithms illustrated in Section IV. After the TIA Portal *Response* packet, if the connection is approved by the PLC, all the following packets exchanged between the TIA Portal and PLC are protected with an *Integrity Part* related to the functions provided by the TIA Portal. In the next section, we investigate this communication process in more detail.

IV. Investigating the Communication Process

In order to understand the encryption algorithms used in the S7CommPlusV3 protocol and explore possible exploits, we need first to analyze the communication process between the TIA Portal and PLC. To this end, A manual analysis was conducted using helpful tools such as Scapy[1] and WinDbg[2], and a number of different communication sessions. First, we open the TIA Portal and press on the "*go online*" button then capture all the packets and save them in a pcap file for a further analysis. To support our study, we use the WinDbg software that allows us to make several breakpoints during the communication session that is comprised of four packets (see figure 4). In the following we present our analysis results for each packet in detail.

[1]https://scapy.net/
[2]https://windbg.org/

A. S7 Request Packet

The first message is a *Request* message sent from the TIA Portal to the PLC in order to initialize a new session. This packet has no encryption bytes, therefore an attacker can re-use this packet "as-is" without any appropriate adjustments.

B. S7 Challenge Packet

After the PLC receives the S7 *Request* from the TIA Portal, it responds by sending an S7 packet (we call it S7 *Challenge*). Our investigation showed that this packet has a 20-byte array that varies significantly every time the TIA Portal sends a new *Request* i.e., every time the user presses on the "go online" button. This 20-byte array is called *ServerSessionChallenge* and located always in the 26th byte position of any S7 *Challenge* (as shown in figure 5).

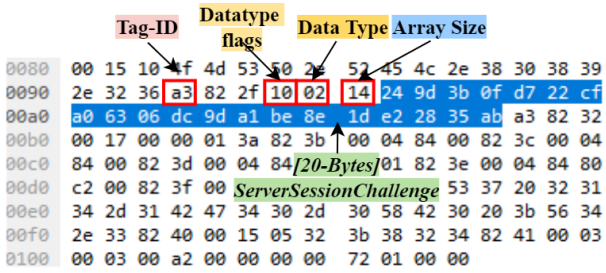

Fig. 5: S7 *Challenge* Packet - *ServerSessionChallenge* Array

By further investigation and inserting several breakpoints at the memory address where this array is located, we found that only 16 bytes, precisely from the 3rd to 18th byte of the 20-byte array were copied and stored in another address. This final 16 bytes block or as called *challenge* in [3] plays an important role in generating certain encryption bytes for the following S7 *Response* and the *Function* packets as illustrated later in the next Subsections (IV-C and IV-D).

C. S7 Response Packet

The *Response* packet is sent from the TIA Portal to the PLC as a response to the *Challenge* packet. It is quite complex and can be divided into several parts as shown in figure 7.

1) ***Encryption Bytes can be manipulated****:* Our investigations to this packet showed that the Secure Hash Algorithm 256 (SHA-256) is utilized two times to generate two hashes. The inputs of the SHA-256 algorithm are generated randomly using the Application Programming Interface (API) cryptography functions, precisely the "*CryptGenRandom*" function see figure 6. The two resulting hashes are then used as a part in generating specific encryption bytes in the S7 *Response* packet. Figure 7 shows these bytes which are as follows:

- **9-byte block** located between the byte 91 and 99.
- **8-byte block** located between the byte 136 and 143.
- **132-byte block** located between the byte 168 and 299. This block is also divided into sub-blocks as follows: the first 76-byte block of the 132-byte block located between the byte 168 and 243, the "First Encryption" (16-byte) between the byte 244 and 259, the "Second Encryption" (16-byte) between the byte 284 and 299.

Since the three encryption blocks are generated based on the two hashes the SHA-256 introduces as outputs [3], adversaries can maliciously manipulate those blocks by manipulating the generation process of the two hashes. To this end, an attacker can use the WinDbg software to feed constant inputs to the hash function of the SHA-256 algorithm that will eventually result fixed hashes rather than random ones when the "*CryptGenRandom*" function was used. For instance, when we feed the hash function with "0" values as inputs, the bytes representing the hashes generated by the SHA-256 algorithms remained constant in every session. This is a very serious vulnerability as an attacker can subsequently generate the three encryption blocks, craft the entire S7 *Response* packet and send it finally to the PLC without the need to have a TIA Portal software installed on his machine as [3]–[5] assumed.

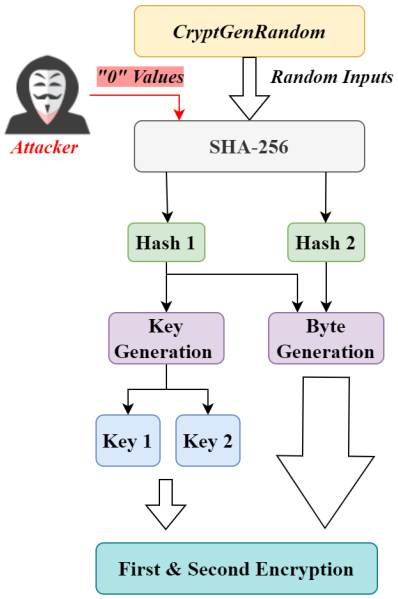

Fig. 6: Generating Keys and Bytes for the First and Second Encryption

One of the two hashes, precisely the "*Hash 1*", is used as a part of a process to generate two 16-byte keys, called "*Key 1*" and "*Key 2*". The resulting keys are then utilized in two symmetric-key encryption processes, precisely the Advanced Encryption Standard (AES)-128 algorithm, referred to as "First Encryption" and "Second Encryption" [6] as depicted in figure 8 and 9 respectively. Since the inputs of the two hashes can be manipulated, the two keys can be also manipulated by attackers. Consequently, the both encryption processes in the S7 *Response* packet could be manipulated. In the following we explain in detail how an attacker can manipulate the "First" and "Second Encryption" processes.

2) ***First Encryption Process****:* Figure 8 depicts the "First Encryption" algorithm that the *Response* packet implements. As can be seen, the output of this encryption is a 16-byte value located always between the 77th and the 93rd bytes of the 132-

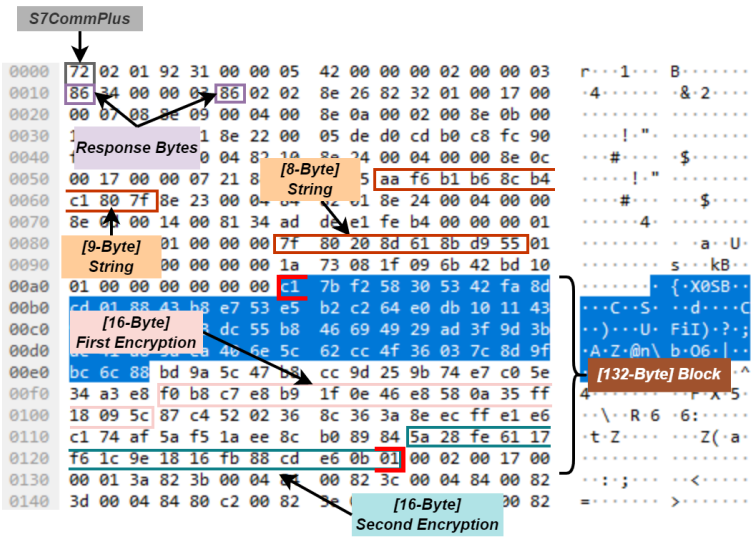

Fig. 7: S7CommPlus Response Packet from TIA Portal to the PLC

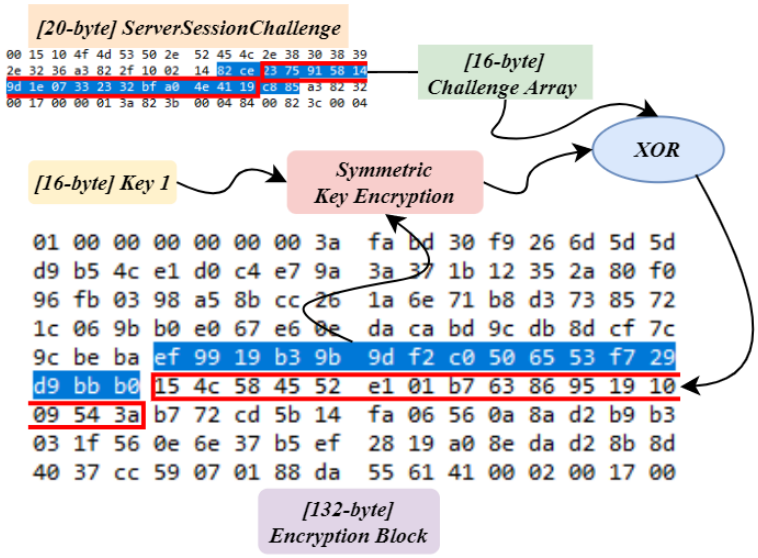

Fig. 8: First Encryption in the S7CommPlus *Response* packet

byte block identified in the S7 *Response* packet (see figure 7). Our analysis showed that the "First Encryption" process uses two inputs: the 61st to 76th byte of the 132-byte block as a plaintext, and the 16-byte "*Key 1*" as an encryption key. Afterwards, the output of the symmetric-key encryption, which is 16-byte block, will be XOR-ed with the 16-byte *challenge* array. The resulting output of the XOR operation is finally stored in a certain address before it is sent to the PLC. For all that, we can conclude that the "First Encryption" process is a simple XOR operation of a constant 16-byte with the *challenge* array. Therefore, to manipulate this encryption, an attacker only needs to manipulate the hashes used to generate the "*Key 1*" as mentioned earlier in IV-C1.

*3) **Second Encryption Process**:* The "Second Encryption" algorithm uses the same encryption algorithm as the "First Encryption" i.e., AES-128, but the plaintext here is generated by a much more sophisticated algorithm which uses the 16-byte output of the "First Encryption" as a part of the inputs in the "Second Encryption". Figure 9 depicts the complete algorithm. As can be seen, there are five different inputs in the plaintext generation process for the "Second Encryption". The five inputs are: 1) a 16-byte value, 2) the result of the first encryption, 3) a 16-byte ciphertext, 4) a 8-byte ciphertext value diminished from another ciphertext and 5) a 4-byte counter value padded with "0" values. Our investigations showed that all of these inputs, except the 16-byte result of the "First Encryption" are constant relative to the two hashes we already identified in IV-C1. Each step of the plaintext generation process is an XOR operation of two inputs and the result of each is fed as input to the next plaintext generation as depicted in figure 9.

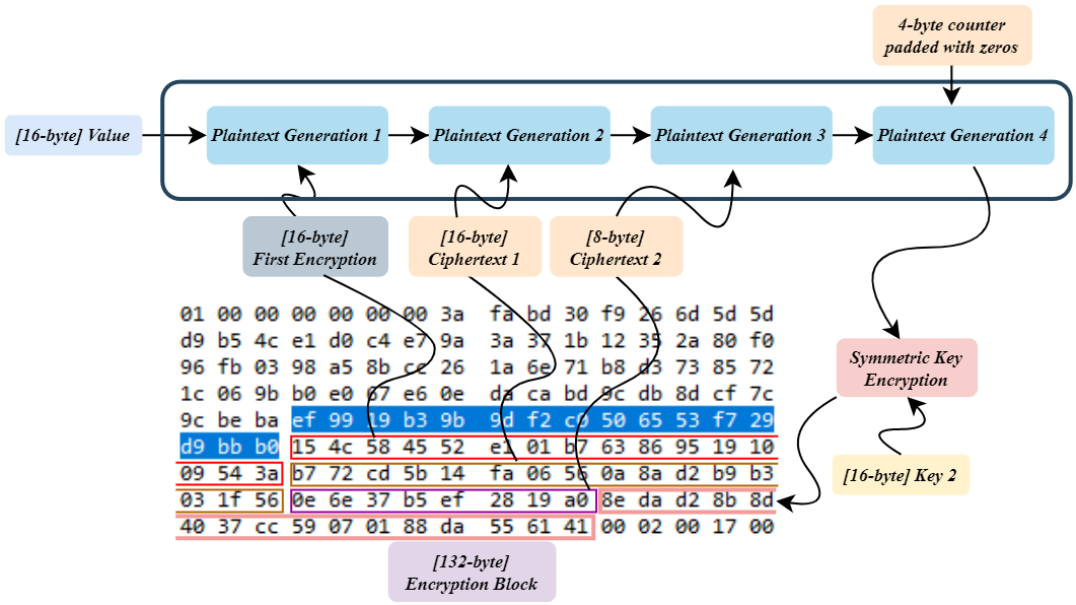

Fig. 9: Second Encryption in the S7CommPlus *Response* Packet

After the final plaintext is generated, it is encrypted using the "*Key 2*" in the same symmetric-key encryption algorithm as the "First Encryption". The output of the encryption algorithm will be used as the last 16-byte of the 132-byte block in the S7 *Response* packet see figure 7.

D. S7 Function Packet

After the S7 *Challenge* and *Response* packets are exchanged between the TIA Portal and PLC, S7 *Function* packets containing required data/operations will be then sent from TIA Portal (see Section III). Figure 10 shows one of these packets which contains control information.

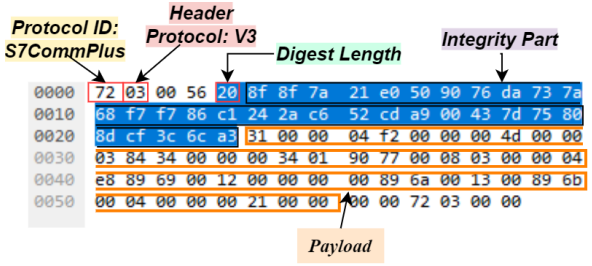

Fig. 10: S7 *Function* Packet from the TIA Portal

Each *Function* packet contains a 32-byte encryption block called *Integrity Part* (as named in [6]) before the payload. Our analysis to the *Integrity Part* shows that this 32-byte block is in fact an Hash-based Message Authentication Code (HMAC) integrity check for the *Function* packet. This integrity check aims at ensuring that the payload has not been maliciously modified, and authenticating the TIA Portal as the encryption keys used in the HMACs are only known by the parties involved in the current connection. To generate the 32-byte *Integrity Part* block, two HMAC algorithms are called. The first one is used to generate a key for all subsequent HMACs, while the second one is used to digitally sign all the following *Function* packets. These two HMACs are designed based on the same hashing algorithm as follows.

1) ***First HMAC****:* The first HMAC is called prior to sending the S7 *Response* packet from the TIA Portal to the PLC. The plaintext here consists of an 8-byte value (generated by a very specific algorithm that is critical to the S7 integrity check [3]) and the 16-byte *challenge* array see figure 11. The combined 24-byte value is then digitally signed using a 24-byte key generated using the two hashes identified earlier in IV-C1. The output of the first HMAC is a 32-byte value but it is diminished to only a 24-byte value which is eventually stored and used as an essential key in the second HMAC computation.

2) ***Second HMAC****:* The second HMAC is used to generate the actual 32-byte *Integrity Part*. Please note that the length of the S7 *Function* packet varies significantly based on the purpose of the packet. However, the 32-byte HMAC output starts always at the 5th byte of any *Function* packet see figure 11. The input of the second HMAC includes all the bytes following the 32-byte *Integrity Part* i.e., starting from the 38th byte, excluding the footer of the packet which is usually the last six bytes (for instant, in the packet shown in figure 11 these six bytes are "*00 00 72 03 00 00*" at the end of the packet). Since the length of each *Function* packet and its payload can vary, the footer also varies from a *Function* packet to another. However, since attackers are familiar with the key generation process, a simple trail-error method could easily determine which bytes are used as input in the second HMAC computation.

V. Attack Scenarios

In Section IV, we analyzed the packets exchanged over the S7CommPlusV3 protocol and found that manipulating

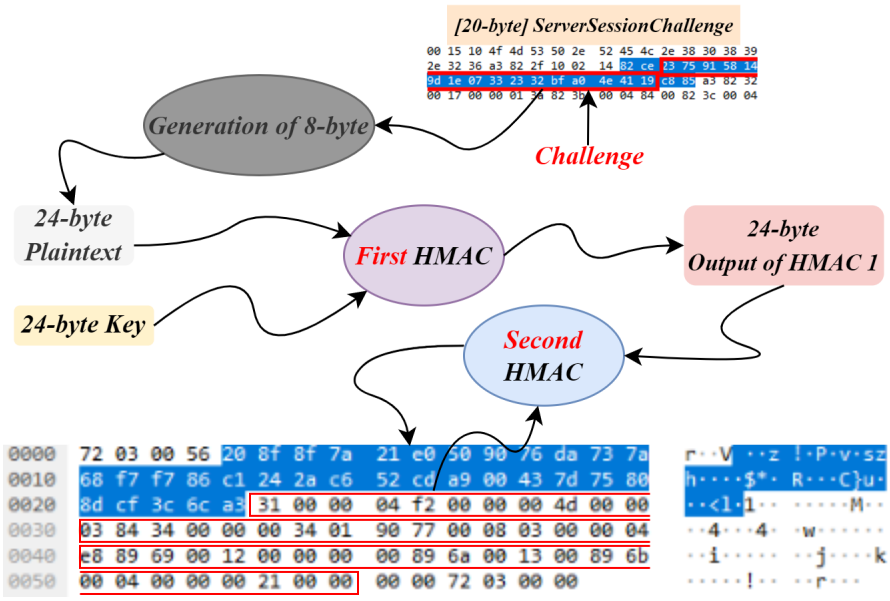

Fig. 11: Integrity Part Encryption Process

the inputs of the SHA-256 will result constant hashes which eventually allows attackers to generate the three encryption blocks in the *Response* packet, as well as the *Integrity Part* in the *Function* packet. This vulnerability is quite severe and exploiting cryptographically-protected PLCs is feasible. In the following, we present potential attacks can be conducted based on crafting S7 *Function* packets. All the crafted packets used to conduct our attacks were built and sent from the attacker machine to the PLC using the Scapy library. Please note that establishing an S7 session with the PLC is out of the scope of this paper and already illustrated in our former paper [4].

A. *Unauthorized Start/Stop Attack*

To start/stop a PLC, The TIA Portal sends an S7 *Function* packet that implements a "*SetVariable*" attribute on the "*NativeObject.theCPUexeUnit_Rid*" object see figure 12.

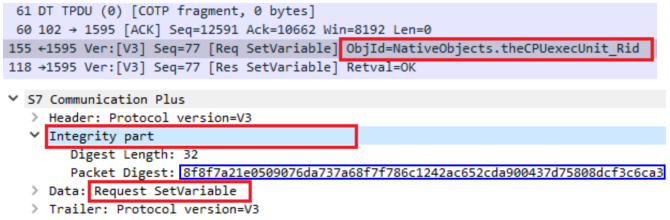

Fig. 12: Start/Stop S7CommPlus Function Packet

This packet contains a 32-byte *Integrity Part* ("Packet Digest" in figure 12) that is checked by the PLC, and only if this block is computed correctly, the PLC verifies the packet and then executes the Start/Stop command. For all this, the attacker needs to craft this S7 *Function* packet by maliciously generating the "Key 1 & Key 2", and computing the HMACs implemented in this block (as explained in Section IV-C and IV-D). It is worth mentioning that even for different sessions, all the encryption bytes can be generated for any spoofed packet based on the same encryption keys and required bytes (see figure 6) that will remain constant as long as the attacker manipulates the two hashes resulted from the SHA-256 algorithm.

B. *Unauthorized Software Changes to the PLC Program*

The user chooses the "Software Changes" option in the TIA Portal to update (download) the PLC program into the PLC. Our experiments showed that it is possible to craft the S7 *Function* packet containing the control logic program sent to the PLC. To this end, we open the TIA Portal software and create a completely new project. Then we program the main Organization Block (OB1) with a malicious program on our will. Afterwards we download this program to the PLC and capture the S7 *Function* packets that are in charge of the download process. By extracting the payload parts of the captured S7 *Function* packets i.e., the bytes after the *Integrity Part* until the last six bytes of the packet "*00 00 72 03 00 00*", and using this payload in our crafted S7 *Function* packet, we can successfully update the program running in the PLC that will only check the *Integrity Part* block not the malicious payload.

C. *Removing the PLC Program*

This attack scenario is quite similar to the former one (V-B), but here the attacker aims at deleting the program (OB1 block) from the PLC's memory. An interesting fact is that the TIA Portal does not provide a "Delete" operation to the user. Therefore, the only way to delete the current program running in the target PLC is to replace the OB1 with an empty one. To do so, we create a new project in the TIA Portal, and leave the OB1 empty without any instructions. Then, we download the empty OB1 (using "Software Changes" operation) to the

remote PLC. Meanwhile, we open the Wireshark and capture all the network packets exchanged between the parties which eventually contain the certain S7 *Function* packets the TIA Portal sends to download the empty OB1. Similar to the former attack scenario (V-B), By extracting the payload from the captured *Function* packets, and using this payload in our crafted S7 *Function* packet (after calculating a new *Integrity Part* and adjusting the sequence number as well as the response bytes appropriately), we can send the malicious S7 packets to the PLC which will replace the OB1 with an empty one, and the infected PLC will have no more instructions to execute.

D. *Unauthorized Hardware Changes to the PLC Program*

Attackers can maliciously alter the hardware configuration causing a non-configured PLC state (unidentified station), and the TIA Portal software will not be able to establish a connection with the infected PLC. To conduct such an attack scenario, we first create a new project on the TIA Portal, inserting a new device (PLC) to the project, and configure the settings of this device different than the settings of the target PLC e.g., a different PLC firmware or IP address. After that, we save the project and download the new configuration to the connected PLC by using the option "Hardware and Software (only changes)". Meanwhile, we capture the packets transferred between the stations and extract the S7 *Function* packets sent from the TIA Portal to make the required changes on the remote PLC. By using the payload of the captured packets in new crafted packets, an attacker can make unauthorized hardware configuration to the PLC that will change its settings causing abnormal behaviors e.g., the PLC cannot be set to the run state since it has a false configuration, or it has a new IP address so it will close all the connections with other devices and shows a synchronizing program message before the user attempts to download his own program i.e., there are differences between the configurations in the control logic in the PLC and the user project in the TIA Portal.

E. *Denial of Service*

Knowing the encryption algorithms (i.e., "First Encryption", "Second Encryption" and HMACs) allows attackers to not only manipulate the operations provided via TIA Portal software but also to perform a DoS attack and to deprive the user of accessing the PLC. To this end, attackers can send crafted packets aiming at only establishing a new S7 session with the PLC, keeping this session alive "forever" by sending "*S7-ACK*" packets regularly as introduced in [4], [9]. This attack scenario is feasible due to two facts. First, all S7 PLCs do not allow a new session to be initiated if there is already an alive session. Secondly, the "*S7-ACK*" packet, "*03 00 00 07 02 F0 00*", lacks the 32-byte *Integrity Part*, and can be utilized "as-is". Assuming that there is no current session between the PLC and the legitimate TIA Portal, an attacker placed on the same network with the PLC can initiate a TCP connection to the PLC via the usual handshake and exchange Connection Oriented Transport Protocol (COTP) packets. After the PLC sends the *Challenge* packet, an attacker can respond with a crafted *Response* packet containing the appropriate encryption bytes, and right after that he follows his *Response* packet with endless-loop of "*S7-ACK*" packets. This will prevent any connection from the legitimate TIA Portal, and the attacker session will remain alive without the need of doing any further actions. Please note that the PLC will keep running the currently existing program, and it is not possible to make any software or hardware changes in the PLC. Therefore, a manual reboot is required to close the attacker session.

VI. Conclusion and Future Work

This paper shows that the encryption processes used by the S7CommPlus protocol to protect the communication process are vulnerable. Motivated adversaries can maliciously introduce constant inputs e.g., "0" values to the SHA-256 algorithm that generates randomly the keys and bytes needed to calculate the three encryption blocks in S7 *Response* packets, and HMAC computations in S7 *Function* packets. Our experiments proved that attackers can craft their own S7 *Function* packets and use them in replay attacks to cause several impacts on target PLCs. Based on our findings, we managed successfully to perform a series of attacks against a real hardware cryptograpgically-protected PLC from 1500 family to validate our results.

This work is done on an assumption that the target PLC is not password protected. In the future, we will try to solve this assumption by investigating the S7 authentication protocol that cryptograpgically-protected PLCs use. We believe that such a protocol has also anti-replay mechanism and the password itself is also somehow encrypted. Thus, a further investigation is needed to cover this topic.

References

[1] [Online]. Available: https://ipcsautomation.com/blog-post/market-share-of-different-plcs/

[2] [Online]. Available: https://roboticsandautomationnews.com/2020/07/15/top-20-programmable-logic-controller-manufacturers/33153/

[3] E. Biham, S. Bitan, A. Carmel, A. Dankner, U. Malin, and A. Wool, "Rogue7: Rogue engineering-station attacks on S7 simatic PLCs," in Black Hat USA, 2019, [Online]. Available: https://i.blackhat.com/USA-19/Thursday/us-19-Bitan-Rogue7-Rogue-Engineering-Station-Attacks-On-S7-Simatic-PLCs-wp.pdf.

[4] W. Alsabbagh and P. Langendöerfer, "A New Injection Threat on S7-1500 PLCs - Disrupting the Physical Process Offline," in IEEE Open Journal of the Industrial Electronics Society, vol. 3, pp. 146-162, 2022, doi: 10.1109/OJIES.2022.3151528.

[5] W. Alsabbagh and P. Langendörfer, "No Need to be Online to Attack - Exploiting S7-1500 PLCs by Time-Of-Day Block," 2022 XXVIII International Conference on Information, Communication and Automation Technologies (ICAT), 2022, pp. 1-8, doi: 10.1109/ICAT54566.2022.9811147.

[6] C. Lei, L. Donghong, and M. Liang, "The spear to break the security wall of S7CommPlus," in Black Hat USA, 2017, [Online], available: https://www.blackhat.com/docs/eu-17/materials/eu-17-Lei-The-Spear-To-Break%20-The-Security-Wall-Of-S7CommPlus-wp.pdf

[7] https://sourceforge.net/projects/s7commwireshark/

[8] A. Spenneberg, M. Brüggemann, and H. Schwartke, "PLC-blaster: A worm living solely in the PLC," in Black Hat Asia Marina Bay Sands, 2016, pp. 1–16.

[9] H. Hui, K. McLaughlin, "Investigating current plc security issues regarding siemens s7 communications and TIA portal," in: 5th International Symposium for ICS & SCADA Cyber Security Research, 2018, pp. 67–73.