Developing a CCSDS compliant platform to reliably secure current and future space data links

Louis Masson
CYSEC
Lausanne, Switzerland
louis.masson@cysec.com

Mickael Bonjour
CYSEC
Lausanne, Switzerland
mickael.bonjour@cysec.com

Laurent Thoeny
CYSEC
Lausanne, Switzerland
laurent.thoeny@cysec.com

Sylvain Willy
CYSEC
Lausanne, Switzerland
sylvain.willy@cysec.com

***Abstract*—The space sector is expanding at an increasing rate thanks to the momentum provided by the NewSpace industry. With this rise of new satellite services and facilities, the stakes of this economy have reached new heights. Ensuring the security of satellites is more essential than ever to preserve the critical infrastructure in space. An identified weakness of satellite platforms is their communication link with Earth-based ground stations. With access to radio equipment and knowledge of standard data link layer protocols, ill-intended actors are able to access sensitive and confidential data downlinked by satellite. Reverse engineering of telecommands and seizing control of the satellite from their operators is only a step away. Industry standardization efforts have aimed to address this, with a 2015 data link layer security protocol by Consultative Committee for Space Data Systems (CCSDS). The Space Data Link Security (SDLS) protocol, complemented with the SDLS Extended Procedures (SDLS-EP) in 2020, allows to secure a satellite's communication link at the data link layer. The authors propose a portable, easy to use, and hard to misuse implementation of the SDLS protocol in an effort to homogenize the industry's security best practices. To support the development of this implementation and other inter-operable implementations of the industry, a development platform is presented. It is composed of two test environments, one to test the security of the core SDLS procedures, another to test high level use cases surrounding the SDLS-EP. The versatility potential of these tools for improving the security of satellite communication is discussed, and the initial results of the security implementation to a variety of test scenarios are detailed.**

***Index Terms*—CCSDS, data link, SDLS, SDLS-EP, extended procedure, secure satellite communications**

I. Introduction

The growth of the civil and commercial space industry has been steadily gaining momentum in the past decade through the proliferation of NewSpace companies. The increase in satellite services brought by this growth spurt provides undeniable value to humankind's economic and scientific interests. Inexorably, this value has been shadowed by legitimate concerns regarding the sustainability and continued safe use of space. Concerns have been raised against the impact that mega-constellations such as Starlink have been having on astronomy activities on Earth. Others have been raised against the ever increasing risk of collisions with debris. The main concern that shall be addressed in this paper regards the security aspects that come with such uncontrolled growth.

As a case study, the recent attack on Viasat modems that occurred on February 24th 2022 has shown how such security events have real and dire consequences. A demonstration of a satellite being taken over by a hostile entity has been demonstrated in an academic exercise presented by Thales at the CYSAT 2023 conference, in Paris. The threat of an intrusion in existing satellites is compounded by the increasing size of satellite constellations. Combined with expansive ground station and terminal networks, these infrastructures provide a wide and vulnerable attack surface for cybersecurity threats.

Future events with nefarious impact to government services, commercial activities, and the sustainability of Earth's access to space are an inevitability [1]. Indeed, targeted attacks on spacecraft orbiting the Earth pose a risk of rendering them inoperable, non-compliant, and – at the worst – dangerous when equipped with propulsion systems. One attack vector that presents itself as sensitive is the communication link from satellite to their ground segment, as well as Inter-Satellite Links (ISL) to a certain extent. Hostile actors with access to radio equipment, with knowledge of a spacecraft's used frequency bands, and with knowledge of frame encoding / decoding processes are well capable of two things: accessing potentially sensitive and confidential data that is transmitted by the satellite, and sending commands (and potentially software updates) with the aim of seizing control of the satellite from its rightful owner and operator.

In light of traditional approaches to spacecraft and communication systems design, security considerations and solutions in the space industry have frequently been dissociated from the design of data link protocols. Adding security as an afterthought is a definite weakness for a system's overall security, and security considerations should be taken into account through all stages of space systems development. Additionally, some commercial space actors have wrongfully relied on security through obscurity by expecting to be protected by the secrecy of their communication protocols.

Recent standardization efforts have aimed to address this. The Consultative Committee for Space Data Systems (CCSDS) published a security standard in 2015, the Space Data Link Security (SDLS) protocol [2]. SDLS provides the means to integrate basic security features – i.e., encryption and authentication – to the CCSDS' widely adopted Space Data Link Protocols (SDLP), e.g. Telecommands (TC) and Telemetry (TM). This standard has been complemented in 2020 by Extended Procedures [3], increasing the security capabilities of the protocol with a Key Management Service (KMS), a Security Association Management Service (SAMS), and a Monitoring & Control Service (MCS).

However, the industry's slow adoption of the standard and the late development of compliant market products are insufficient when faced with the urgency posed by cybersecurity threats to satellite services. This paper aims to present a platform for the development of security solutions with ease of deployment and security best practices in mind. To provide a solution that can be widely used and swiftly integrated into already deployed systems, compliance with the SDLS protocol has been targeted. With this aim in mind, a test platform has been developed by the authors to simulate the link and instrument the libraries being tested between a spacecraft and a ground station. The platform consists of test environments following two goals: to test the resiliency of an implemented secure link, and to validate the correct execution of Extended procedures.

In the following sections, an overview of the SDLS protocol shall be provided first. This shall be followed by a presentation of SATLINK, the SDLS compliant solution that has been developed by CYSEC. Subsequently, the two test environments that make up the presented development platform shall be presented, followed by a discussion on the results that have been obtained by these platforms.

II. Overview of Space Data Link Security

The Space Data Link Security (SDLS) protocol has been published by the Consultative Committee for Space Data Systems (CCSDS) in 2015 [2]. A summary of concepts and rationale of the base SDLS protocol and its extensions has been published in a CCSDS Green Book [4] [5] [6]. With this protocol, the CCSDS has opened up the possibility of adding security features to protocols that reside in the data link layer, i.e. Layer 2, of the Open System Interconnection (OSI) model.

Addressing the security of the link between a satellite and the ground segment at the data link layer comes with the advantage of early mitigation of attacks. This is due to security being applied at the first data level of the communication stack. Data link layer frames carry packets that need to be routed through the satellite bus to all subsystems, sensors, actuators, and payloads that a satellite is comprised of. Detecting issues when a frame is decoded and before routing packets to sensitive systems makes for an overall better reactivity in the events of attacks.

The implementation of authentication mechanisms at the data link layer also reduces the overhead of security on the bit rate of the communication. By authenticating a group of packets transported in a single frame instead of authenticating each packet individually, a single Message Authentication Code (MAC) is needed. This reduces the impact of security on effective transfer rates, resulting in a more efficient space link in terms of data throughput.

The SDLS protocol aims to solve two concerns that are typical in communication protocols between remote systems, satellites included:

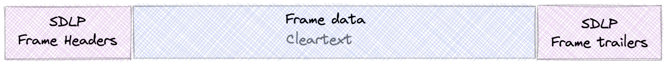


Fig. 1. Structure of an unsecured CCSDS frame

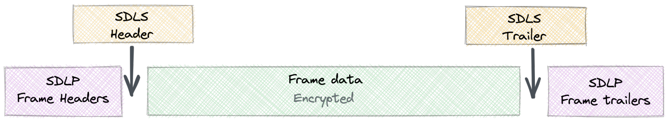


Fig. 2. Insertion of security features into a frame

- *authenticity* of the communication is guaranteed, meaning that the receiver shall be able to verify that incoming communications have been created by a legitimate and authenticated endpoint,
- *confidentiality* of the transferred data is guaranteed, meaning that any actor intercepting the data shall not be able to interpret it.

A. Principle of operation

The basic security mechanisms allowing the SDLS protocol's services to address these two concerns are twofold, as illustrated in Fig. 1 and Fig. 2.

On the one hand, encryption of a frame's data field through a cryptographic encryption algorithm associated with a matching key ensures confidentiality of the data carried by the packet. The resulting cipher text cannot be decrypted by an actor which does not have access to the key used in the cryptographic process. Intended recipients of the frame which hold the key shall be able to decrypt the content of the frame's data field.

On the other hand, a frame can be authenticated by computing a MAC through a cryptographic authentication algorithm associated with a matching key. The computed code is then appended to the end of the frame, encapsulated in a Security Trailer (see Fig. 2). A receiver will be able to run the frame through the same algorithm while using the same key and compare the resulting MAC with the code that has been appended. When asserting that the calculated and appended codes match, the recipient shall be able to verify that the frame has been created by an actor who holds the same key.

Both security mechanisms may be combined at once through separate cryptographic algorithms for authentication and encryption using their respective keys, or by using Authenticated Encryption with Additional Data (AEAD) algorithms such as AES-GCM with a single key [7].

The security procedures supported by the SDLS protocol are based on symmetrical cryptographic algorithms, implying that both endpoints (i.e. satellite and ground station, or satellite and satellite) hold a shared secret (i.e. identical pair of private keys).

B. Concepts of the base protocol

SDLS is a protocol that is described through procedures. Two procedures form the minimal and baseline service that

SDLS-compatible systems must provide: *ApplySecurity* is initiated by a sending endpoint in order to insert security features into an outgoing frame (i.e. authentication and/or encryption); *ProcessSecurity* is initiated by the receiving endpoint in order to unpack an incoming frame from its security features (i.e. decryption and/or authentication).

In support of these procedures, the standard defines the concept of Security Associations (SA). These data constructs essentially configure and manage the secure link between two endpoints, and are instanced on and maintained by both endpoints. An SA configures a secure link by defining the algorithms to be used, the keys that must be used with these algorithms, and manages a rolling Anti-Replay Sequence Number (ARSN). The SA concept has been borrowed and adapted from the IPSec VPN protocol for SDLS.

The latter, used in security schemes relying on authentication, tracks the most recent frame that has been successfully authenticated by the protocol. It is an added layer of security which serves to thwart replay attacks: frames that are of an older sequence number than the ARSN value currently maintained by an SA shall immediately be rejected.

The correct end-to-end execution of *ApplySecurity* and *ProcessSecurity* relies on parameters that must be passed from one endpoint to the other. This is achieved through Protocol Data Units (PDU) that are inserted into the frame by the *ApplySecurity* procedure as illustrated in Fig. 2. The optional Security Trailer merely carries a MAC for secure links that require authentication. The mandatory Security Header provides necessary arguments for the receiving endpoint's execution of *ProcessSecurity*:

- a Security Parameter Index (SPI) identifying which SA should be used for the procedure,
- (optionally) a Sequence Number (SN) if authentication is used,
- (optionally) an Initialization Vector (IV) if encryption is used,
- (optionally) a padding length parameter when it is required for a given algorithm.

C. Extended Procedures

The SDLS protocol published by the CCSDS in 2015 provides a minimum viable approach to the issue of satellite communication security. The proposed approach comes with the assumption that satellites are deployed with a static set of keys and SA through which the mission would be able to cycle through if needed. The simplicity of this implementation unfortunately comes at the cost of the system's overall security. Should all keys be used in response to security events through the course of the mission, the satellite operator will not be able to guarantee the confidentiality and authenticity of communications anymore.

In response to this inherent weakness of the base SDLS protocol, the CCSDS has issued in 2020 the Space Data Link Security Extended Procedures (SDLS-EP) [3] [8]. The procedures defined in this extension of the SDLS protocol expand on the protocol user's control of keys and SA through a multitude of services: a Security Association Management Service (SAMS), a Key Management Service (KMS), and a Monitoring & Control Service (MCS).

Each service provides a set of procedures which may be initiated by any endpoint of a secure link. These procedures are executed in the form of PDU transported in data link layer frames which carry commands to and responses from either endpoint. These procedures enable Over The Air Rekeying (OTAR) of the secure link of live & operational missions through the KMS. In addition to these key rotation capabilities, the procedures also enable the re-configuration and creation of new SA for all communication channels of the secure link through the SAMS. Finally, the MCS provides monitoring capabilities of the secure link, enabling the mission operator to monitor any security events that may have occurred on the secure link.

D. State of the protocol in the industry

The importance of this security standard being published by the CCSDS has industry wide implications. The Telecommand (TC) [9] and Telemetry (TM) [10] Space Data Link Protocols (SDLP), among other SDLP [11], are issued by the CCSDS.

The implementation of these data link layer protocols and their use in the civil industry is common ground. The built-in compatibility of the SDLS protocols with these SDLP facilitates the rapid adoption of this security standard across the aerospace industry. Additionally, a lightweight implementation can be applied to already deployed satellite missions. This concerns satellites supporting software update capabilities, paving the way to secure the communication links for currently unsecured satellite communication links.

Governmental agencies are known to have their own implementation of the SDLS protocol for their missions, with inter-operability tests between implementations have been conducted between the European Space Agency (ESA), the Centre National d'Études Spatiales (CNES), and the National Aeronautics and Space Administration (NASA) [12] [13].

However, the recent publication of the SDLS protocol means that the industry adoption rate is not universal after this short time frame. This is made apparent by the absence of market-ready solutions that exist at the time of publication of this paper. Publications that approach the topic of the SDLS protocol use it as a case study rather than through an implementation angle. The exception being the inter-operability testing published by governmental agencies [12].

III. SATLINK: SDLS COMPLIANT LIBRARY

There are no SDLS compliant solutions publicly available to the civil aerospace industry at the time of publication of this present paper. ARCA SATLINK, presented in the following paragraphs, has been developed by CYSEC in order to address this current gap in the industry. As it shall be discussed, ARCA SATLINK is developed on two separate streams: one focusing on an implementation targeted at ground segments, another focusing on an implementation targeted at space segments (i.e. satellite platforms). The development of the ground endpoint will be mainly discussed, as it has been the main actor of the tests environments described in the following sections. The space endpoint is briefly touched upon in the following paragraphs, but the specifics of this implementation will be addressed in a followup publication.

A. *Design assumptions*

In an effort to pursue industry homogenization through the application of standards, ARCA SATLINK has been designed to be compliant with the SDLS protocol (see Section II). Since SDLS is grafted onto CCSDS data link protocols such as TC and TM, the library would need to be usable within satellite operators existing workflows without replacing them. Hence the form of a library exposing a convenient interface has been chosen for ARCA SATLINK. Illustrations of how ARCA SATLINK would be deployed on either ground or space segments are respectively shown in Fig. 4 and Fig. 5.

A primary assumption for the implementation of this library stipulates that it would need to be compatible with a wide variety of configurations. This is because each satellite integrator's implementation may vary in terms of used hardware and network configurations, which is all the more pertinent for the space segment. Therefore, the library shall not rely on specific backend to complete its operations. In the following paragraphs, a backend is defined as a layer of software that interfaces the library's internal modules for storage or cryptographic processes with external software or hardware.

As ARCA SATLINK is intended as a library with built-in security best practices, another important design assumption is that it shall be hard to misuse. As an example as to what this entails, any user choice that may lead to a security risk – e.g., using encryption only mode without protecting the authenticity of the data – must be explicitly enabled by the library's before being usable.

Finally, one final assumption for the design of ARCA SATLINK is that the user's data link implementation between the various endpoints is reliable. Since the library does not replace the communication link workflow of the user but is instead used to enhance it with added security, it is dependent on the user to transport frames to and from the other endpoints.

B. *Software architecture*

An overview of the architecture is schematized in Fig. 3. All the colored blocks are considered as modular components of ARCA SATLINK. It is important to consider that, typically, the ground variant of the library will act as the initiator of SDLS-EP procedures. Therefore, the ground segment is not focusing on being lightweight but on being robust and covering all the features that are needed.

As a stated goal for the library, as much flexibility as possible must be offered to the user. One example is the customization of the library through the many mission-specific parameters that can differentiate a mission from another. Additionally, the library accepts different approaches to some mission configurations. The best example of this may be found

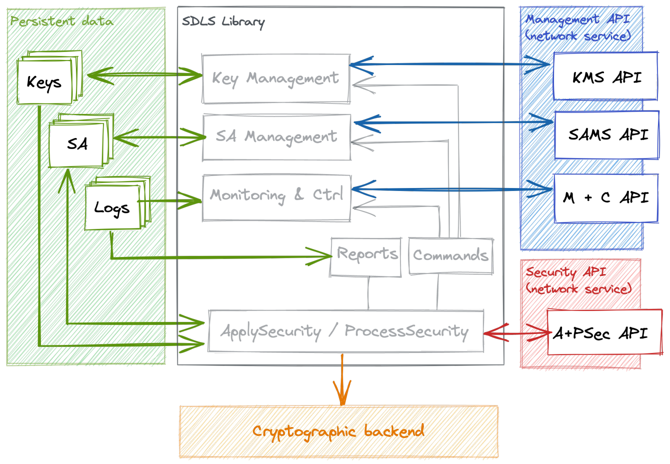


Fig. 3. Structure of the SATLINK ground library

with the use cases of the SDLS Extended Procedure (SDLS-EP): the library may be used either as a proxy, in which it generates packets that the user wraps in their used data link protocols; or it may be used transparently, where a dedicated virtual channel of the satellite is used to transport SDLS-EP Protocol Data Units (PDU).

The library has been made to be independent of specific software or hardware through its modular design. Firstly, it supports different cryptographic libraries as the backend for its security operations, and can be extended to interface with existing solutions should a specific user need require it. Secondly, the same principle applies to SA and keys: their storage may be handled by an existing database or secrets management implementation – e.g., Amazon Web Services (AWS) S3, Hashicorp Vault, regular filesystem – or they may be tailored to match specific needs. Finally, the input and output streams, as well as the logs generated by the library, come with a generic implementation. It is up to the user to decide how the library will be used depending on their needs.

C. *Development approach*

The development process of SATLINK has followed Test Driven Development (TDD) principles. Due to the safety critical nature of the project, a strong focus on quality and security for both the ground and space variants of the library has been enforced.

The quality of the software has been built from the ground up since the early stages of development steps with the elaboration of extensive tests at several levels: unit tests to verify the behavior of individual components, integration tests to validate the behavior of multiple components integrated together, and high level functional tests which validate the use cases of the library. Integration tests have been used to ensure the quality of the modular aspects of ARCA SATLINK, as combinations of backends must be verified to be compatible with one another. The test environments presented in the next sections of this paper pertain to the high level testing of the library. Continuous Integration and Delivery (CI/CD) practices

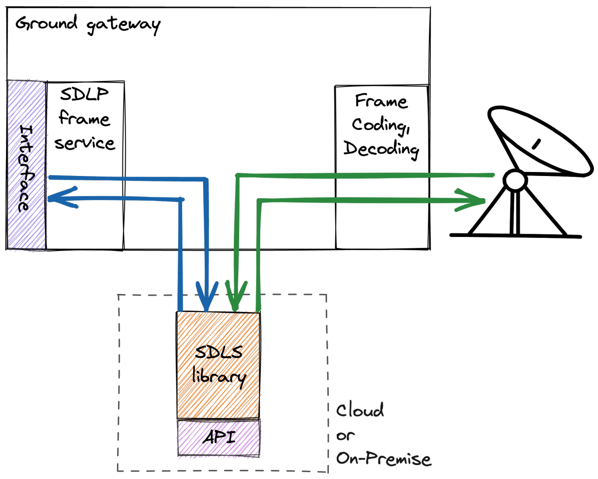


Fig. 4. Example of a SATLINK ground integration

have been followed, with reviews and regression testing being applied to every project increment.

Due to the safety critical nature of the project, the library would need to be crash free and able to fail safely in case of failure. With this in mind, the Rust programming language has been selected as the programming language for the ground variant of the library due to its strong memory safety and concurrency guarantees [14]. The static analysis of the Rust compiler alongside tools from the Cargo ecosystem have been instrumental in maintaining high quality standards.

D. Ground deployment

The deployment of ARCA SATLINK on the ground segment follows the workflow presented in Fig. 4. It is either implemented in its library form as an on-premise solution, or it is hosted on the cloud as a service hosted on the Amazon Web Services (AWS) platform. The two approaches are discussed in the following paragraphs.

For on-premise deployments, the satellite integrator integrates ARCA SATLINK to their workflow by directly using the Application Programming Interface (API) calls of the library. After library initialization, frames are passed through *ApplySecurity* and *ProcessSecurity* calls in order to be encrypted, decrypted, or authenticated.

The cloud deployment of ARCA SATLINK aims to be integrated with the AWS ecosystem. The ARCA SATLINK library is hosted on an AWS instance, and its API calls are exposed externally to the user. This storage, logging, and cryptographic processes are all fulfilled by AWS live services. To encrypt, decrypt, or authenticate frames, the user must push frames to be processed and pull processed frames to and from AWS First-In-First-Out (FIFO) queues. The architecture of SATLINK on AWS respects the one introduced earlier, with the whole API being provided in a cloud alternative.

E. Space deployment

The same concepts and design principles have been applied to the space variant of ARCA SATLINK. The space endpoint's

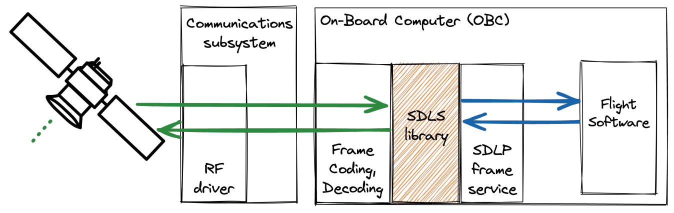


Fig. 5. Overview of SATLINK space deployment

library has been developed with a focus on flexibility – due to less providers being natively supported – and with the goal of not constraining the user. This is all the more relevant in space applications as satellite architectures vary considerably from on satellite integrator to the next.

Unlike its ground variant, the space-bound library has been made to be portable and lightweight. The portability aspect stems from the aforementioned particularity of satellite platforms being integrator specific. As for the software being lightweight, this has been identified as essential to make the integration of this software with bare metal applications that are sometimes found in the embedded systems used to drive satellites.

Following these requirements, the space variant has been developed using the C99 programming language. Due to the safety critical aspects of the deployment on a satellite and the more stringent memory safety requirements, Motor Industry Software Reliability Association (MISRA) 2012 guidelines for C99 have been followed. In the interest of qualifying the software for re-use by satellite integrators, European Cooperation for Space Standardization (ECSS) software development and quality processes have been followed. The same TDD principles have been followed for this project as well, combined with the same CI/CD processes. These guidelines coupled with the use of static analysis tools such as SonarQube [15] and Valgrind [16] have provided optimal guarantees regarding the safety and security of the software.

The space-bound form of ARCA SATLINK is only intended to be deployed in its library form by the developer of a satellite's flight software. Fig. 5 illustrates an example of a simplified satellite architecture using SATLINK space library. The user must invoke function calls exposed by the library in order to encrypt, decrypt, and authenticate incoming and outgoing frames before broadcasting or receiving them.

IV. End-to-end test platform

The functional tests, apart of the TDD methodology described in the previous section, are run as part of a ensemble of automated regression tests that are performed in the CI/CD workflow. To facilitate the instrumentation of these tests, two test environments have been developed. The first one of these, the End-to-End (E2E) test environment, shall be presented in this section.

The purpose of E2E tests are to assess that the SDLS implementation deployed on two endpoints is performing as

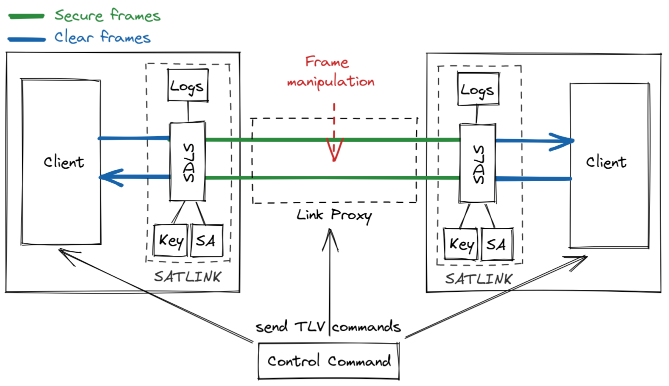


Fig. 6. Architecture of the E2E SDLS test environment

described by the baseline version of the standard (without Extended Procedures). In other words, they allow to ensure that frames that have gone through the *ApplySecurity* procedure will successfully be decrypted and authenticated when going through the *ProcessSecurity* procedure. Additionally, the E2E tests serve the purpose of verifying the resilience of the SDLS implementation to common attacks that could be perpetrated on the space link. This E2E test environment has orchestrated the verification latter by providing a backdoor which simulates attacks on the physical channel.

It is worthy of note that, although this E2E test environment has been developed for the tests of the SDLS implementation presented in this paper, it is capable of being interfaced with other implementations. In particular, ARCA SATLINK may be tested with another SDLS implementation, or any other two implementations may be tested together. This paves the way for inter-operability tests between SDLS implementations, and for the verification of the mutual compliance to the CCSDS standard across the industry.

The architecture of the environment presented in Fig. 6 has been used for the demonstration of the ground variant of ARCA SATLINK at the CYSAT 2023 conference in Paris. For this demonstration, the two endpoints were run in environments which approximate their real deployment targets: an endpoint has been deployed on an embedded STM32 board to simulate the space endpoint; another endpoint has been deployed on AWS to simulate the ground segment.

The *Client* shown in Fig. 6 serves as the means to instrument the SDLS implementation to be tested, and to provide an interface between the library and the rest of the environment. Both endpoints run a client so that frames are processed by the SDLS implementations' *ApplySecurity* and *ProcessSecurity* procedures. Outgoing frames on which *ApplySecurity* has been applied are then transferred to the other endpoint through the next component.

The *Link Proxy* shown in Fig. 6, as aptly suggested by the name, is a proxy through which all transactions of frames between both endpoints transit. In essence, it simulates the physical layer – i.e., Radio Frequency (RF) bands – of the communication stack. Conveniently, it also allows the user of the E2E test environment to eavesdrop on the frame traffic between both endpoints, be it secured or not. Not only does it serve to monitor the communications, but it also provides means to interfere with the communication link through frame manipulation. Such a mechanisms allows to simulate cases where a third party intercepts or injects frames into the space link in an attempt to seize control of the satellite from its operator. As such, this software component fulfills the environment's imperative to verify that the SDLS implementation is resilient to common attack patterns.

The frame manipulation capabilities of the *Link Proxy* are based on identified potential attack vectors. As such, in addition to providing a window into what is transiting through the link, this component of the test environment is capable of performing:

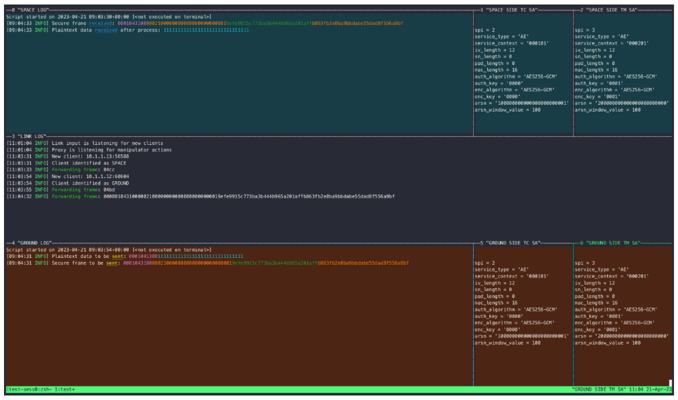
- **Frame replay**, to test the anti-replay sequence number mechanism;
- **Frame tampering**, to assess the authentication mechanism;
- **Frame permutation**, to provide insight into the system's behavior in cases of frames arriving in disorder;
- **Frame forwarding**, for nominal frame transfers without intervention.

Finally, the *Control Command* shown in Fig. 6 is the tool providing control over the whole test environment to the user. Additionally, it can be used to automate testing which is essential in making CI/CD processes efficient. This component has the form of a command line software that is executed from a terminal. It provides control over the other components of the test environment, any *Client* or the *Link Proxy*, by sending Tag-Length-Value (TLV) packets that describe various operations:

- (*Client*) Send an amount of N TM frames from the space endpoint,
- (*Client*) Send an amount of N TC frames from the ground endpoint,
- (*Link Proxy*) Replay the next N frames from either endpoints,
- (*Link Proxy*) Tamper the next N frames from either endpoints,
- (*Link Proxy*) Perform random permutation of the next N frames from either endpoints.

Same as for the ground variant of ARCA SATLINK, all of the previously presented components of the E2E test environment have been developed in the Rust programming language. This has been convenient for developing cross-compatible, memory safe, small-scale, and easy to maintain software to implement these components.

Another goal of the E2E test environment has been to visualize the logs coming from the *Client* endpoints and the *Link Proxy* to demonstrate that ARCA SATLINK, or any other SDLS implementation, is behaving as intended. The logs produced by each component are mounted into the operator of the test bench and showed in an easily readable output as shown in the Fig. 7. This is achieved using a *tmux* terminal in order to display information side-by-side in organized panels.



```
0 "SPACE LOG"
Script started on 2023-04-21 09:03:30+00:00 [<not executed on terminal>]
[09:04:33 INFO] Secure frame received: 0001043100000210000000000000000000000019efe9915c773ba3b444b965a201affb863fb2e0ba9bbdabe55dad8f556a9bf
[09:04:33 INFO] Plaintext data received after process: 111111111111111111111111111111
```

```
1 "SPACE SIDE TC SA"
spi = 2
service_type = 'AE'
service_context = '000101'
iv_length = 12
sn_length = 0
pad_length = 0
mac_length = 16
auth_algorithm = 'AES256-GCM'
auth_key = '0000'
enc_algorithm = 'AES256-GCM'
enc_key = '0000'
arsn = '1000000000000000000000001'
arsn_window_value = 100
```

```
2 "SPACE SIDE TM SA"
spi = 3
service_type = 'AE'
service_context = '000201'
iv_length = 12
sn_length = 0
pad_length = 0
mac_length = 16
auth_algorithm = 'AES256-GCM'
auth_key = '0001'
enc_algorithm = 'AES256-GCM'
enc_key = '0001'
arsn = '2000000000000000000000000'
arsn_window_value = 100
```

```
3 "LINK LOG"
[11:01:04 INFO] Link input is listening for new clients
[11:01:04 INFO] Proxy is listening for manipulator actions
[11:03:31 INFO] New client: 10.1.1.13:56588
[11:03:31 INFO] Client identified as SPACE
[11:03:33 INFO] Forwarding frame: 04cc
[11:03:54 INFO] New client: 10.1.1.12:60604
[11:03:54 INFO] Client identified as GROUND
[11:03:55 INFO] Forwarding frame: 04bd
[11:04:32 INFO] Forwarding frame: 00000104310000021000000000000000000000019efe9915c773ba3b444b965a201affb863fb2e0ba9bbdabe55dad8f556a9bf
```

```
4 "GROUND LOG"
Script started on 2023-04-21 09:03:54+00:00 [<not executed on terminal>]
[09:04:31 INFO] Plaintext data to be sent: 0001041300111111111111111111111111111111
[09:04:31 INFO] Secure frame to be sent: 0001043100000210000000000000000000000019efe9915c773ba3b444b965a201affb863fb2e0ba9bbdabe55dad8f556a9bf
```

```
5 "GROUND SIDE TC SA"
spi = 2
service_type = 'AE'
service_context = '000101'
iv_length = 12
sn_length = 0
pad_length = 0
mac_length = 16
auth_algorithm = 'AES256-GCM'
auth_key = '0000'
enc_algorithm = 'AES256-GCM'
enc_key = '0000'
arsn = '1000000000000000000000001'
arsn_window_value = 100
```

```
6 "GROUND SIDE TM SA"
spi = 3
service_type = 'AE'
service_context = '000201'
iv_length = 12
sn_length = 0
pad_length = 0
mac_length = 16
auth_algorithm = 'AES256-GCM'
auth_key = '0001'
enc_algorithm = 'AES256-GCM'
enc_key = '0001'
arsn = '2000000000000000000000000'
arsn_window_value = 100
```

```
[test-sess0:zsh- 1:test*                                    "GROUND SIDE TM SA" 11:04 21-Apr-23
```

Fig. 7. E2E test environment visualization of logs:
top left is the space endpoint log; center is the *Link Proxy* log; bottom left is the ground endpoint log; top right displays the space endpoint's SA; bottom right display the ground endpoint's SA.

Coloring is applied to panels to easily situate which endpoint they concern, i.e. blue for space and red for ground.

V. SDLS-EP INTER-OPERABILITY TESTING

As the SDLS protocol is a standard, it is naturally implied that it will come in several implementations from various authors. To uphold the standard as a means to homogenize industry practices, each implementation should be inter-operable with one another. Erroneous interpretation of the standard is a possibility, thus testing implementations in this manner becomes a necessity. In order to reduce this risk, the ARCA SATLINK solution is tested with itself as well as with other available SDLS implementations. With this CCSDS standard, the inter-operability is not guaranteed as there are several parameters that are mission-specific. ARCA SATLINK must then be configured with the same algorithm and parameters on both implementations to be tested.

As the Extended Procedures are required be transported by frames secured through SDLS procedures, testing the inter-operability of the SDLS-EP also allows to test of the baseline procedures, *ApplySecurity* and *ProcessSecurity*. At the time of writing, the authors test ARCA SATLINK against NASA's CryptoLib[1] as it is open source.

The inter-operability tests may be executed either on individual SDLS-EP procedures, or on more complex real-world use cases. One such use case that is useful to evaluate is the rotation of keys for a currently operational SA. For this use case, which is framed as an *operation* by the authors, the following sequence of seven SDLS-EP are executed:

1) *OTAR* (Over-the-Air-Rekeying): a new set of keys are transferred to the recipient,
2) *Key Activation*: the state of a set of keys is set to Active on both the initiator and the recipient,
3) *Stop SA*: the state of an SA is set to Keyed on both the initiator and the recipient,
4) *Expire SA*: the state of an SA is set to Unkeyed on both the initiator and the recipient,
5) *Set ARSN*: the anti-replay sequence number of the SA is set to zero,
6) *Rekey SA*: the keys of an SA are configured for both the initiator and the recipient,
7) *Start SA*: the state of an SA is set to Operational on both the initiator and the recipient.

In order to test the SDLS-EP and various use cases transcribed as *operations*, different test cases have been automated. This has been achieved through a suite of python scripts allowing to programmatically launch and control the different software components. The different test instructions implemented in these scripts are the following:

- Start and stop an SDLS implementation instance, i.e. ARCA SATLINK or NASA's CryptoLib;
- Setup the databases, to simulate a specific situation or scenario;
- Inspect the databases and logs, to assert their states and content;
- Instrument an SDLS implementation, to execute a specified SDLS-EP;
- Instrument the *Link Proxy*, to simulate issues on the communication link.

This approach has the advantage to offer a simple way to manipulate the different entities interfaced by the tests. The tests can then be scripted in python or any other compatible framework allowing to easily create a great number of complex test scenarios replicating real-life *operations* in various situations and starting conditions.

To control the tests, the authors use behave[23], a python Behavior Driven Development (BDD) testing library. In order to be controllable from within the tests, each library is integrated into a dedicated test client that listens for commands on Transmission Control Protocol (TCP) links. This enables the instrumentation of the tests from any language and framework. The authors chose python with behave for its simplicity and the support of Gherkin language, which allows to write powerful automated tests in plain English.

In this example, the feature corresponds to the name of the test suite and each test is a scenario. Each scenario is composed of steps beginning with either *Given*, *When* or *Then* keywords to express a different type of instruction within the test. The steps are implemented as string patterns, associated with a python function, which can contain identifiers to capture values as function parameters. The following is an example of steps that can be implemented for a test case:

```
 Then sending a command Key Inventory (1 to 10) to "A"
 Then the node "A" should have logged "Received EP
notification: KeyInventory([(KeyID([0, 1]), Active)])"
```

Although Gherkin hides the technical specificities of the tests, it opens up the writing and maintenance of test to more

[1]NASA Cryptolib repository: https://github.com/nasa/CryptoLib

[2]Behave repository: https://github.com/behave/behave
[3]Behave documentation: https://behave.readthedocs.io

people. In particular, people who have functional expertise on the tested implementation's behavior but don't need to know the details of the implementation. As these types of end-to-end tests are very high-level, the details are far less relevant than for unit or integration test. In this context, the advantages overweigh the disadvantages of the obfuscation of a test's technical implementation.

VI. Conclusion

The looming cybersecurity threat to satellites by way of their sensitive communication link to Earth has been well established. This threat vector would allow ill-intentioned actors to access potentially sensitive data broadcasted by the satellite and, in the worst case, seize control from its operator. The reliance on security through obscurity leads into a false sense of security. Furthermore, the decoupling of security aspects from other aspects of a mission's design introduce vulnerabilities to the system.

To address these concerns, a solution following the standardization efforts of the CCSDS through the SDLS protocol is proposed in this paper. The development of an easy to use, hard to misuse, and secure solution, the ARCA SATLINK software library, has been presented. A defining characteristic of this tool is its versatility in terms of supported cryptographic backends and secrets (i.e. keys and Security Associations) storage. Additionally, it comes with built-in security best practices which have been design and reviewed in collaboration with security professionals. Two different variants are discussed: a ground variant, developed in Rust and intended for deployment in ground stations; and a space variant, developed in C following ECSS standards for flight software development intended for deployment in spacecraft. The libraries aim to be inter-operable with other implementations following the same standard, hence use of the two variants in conjunction is not necessary.

A platform for the testing and validation of software implementing the SDLS protocol, whether it be ARCA SATLINK or any other, is then presented through two test environments.

On the one hand, an end-to-end test environment instrumenting SDLS implementations and provides tools to test the resilience to typical attacks. The tools allow the implementer to test the reaction of the library to replay attacks, frame tampering, and frame permutations. This particular test environment mostly concerns itself with the base SDLS protocol and precludes any key rotations and Security Association reconfigurations. Two instances of the ground variant of ARCA SATLINK have been successfully interfaced using this environment.

On the other hand, a test environment for SDLS Extendend Procedures tests the inter-operability of various implementations on the procedures. An example of tests between the ground variant of ARCA SATLINK and the NASA CryptoLib is provided, and shows that inter-operability can be achieved and validated. This test environment is still a work in progress and will keep evolving in the future to support all of the SDLS Extended Procedures.

These developments form stepping stones in an effort to consolidate the industry's practices in securing satellite communications. Considerable effort has been poured into understanding the standard, which have served to successfully design the architecture of ARCA SATLINK, a compliant implementation. The presented work has been instrumental in formulating one of the first SDLS compliant products to be presented publicly. The next steps to be followed for this endeavor include the testing of the space variant of ARCA SATLINK in both test environments. Additionally, the space variant of ARCA SATLINK is to be flown on the OPS-SAT cubesat operated by ESA with the objective of attaining flight heritage.

References

[1] P. Tedeschi, S. Sciancalepore, and R. Di Pietro, "Satellite-based communications security: A survey of threats, solutions, and research challenges," *Computer Networks*, vol. 216, p. 109246, Oct. 2022.

[2] Consultative Committee for Space Data Systems, "Space Data Link Security Protocol (CCSDS 355.0-B-2)," 2022.

[3] Consultative Committee for Space Data Systems, "Space Data Link Security Protocol—Extended Procedures (CCSDS 355.1-B-1)," 2020.

[4] Consultative Committee for Space Data Systems, "Space Data Link Security Protocol—Summary of Concept and Rationale (CCSDS 350.5-G-1)," 2018.

[5] I. A. Sanchez, G. Moury, and H. Weiss, "The CCSDS Space Data Link Security protocol," in *2010 - MILCOM 2010 MILITARY COMMUNICATIONS CONFERENCE*, (San Jose, CA, USA), pp. 219–224, IEEE, Oct. 2010.

[6] I. A. Sanchez, G. Moury, C. Biggerstaff, B. Saba, D. Fischer, and H. Weiss, "Towards completion of the CCSDS space data link security protocol," in *2012 IEEE Aerospace Conference*, (Big Sky, MT), pp. 1–18, IEEE, Mar. 2012.

[7] D. McGrew and K. Igoe, "AES-GCM Authenticated Encryption in the Secure Real-time Transport Protocol (SRTP)." RFC 7714, Dec. 2015.

[8] D. Fischer, I. Aguilar Sanchez, D. Koisser, B. Saba, G. Moury, B. Bailey, C. Biggerstaff, H. Weiss, and D. Richter, "Making space-link security work: Auxiliary services to enable the CCSDS Space Data-Link Security Protocol," in *AIAA SPACE 2016*, (Long Beach, California), American Institute of Aeronautics and Astronautics, Sept. 2016.

[9] Consultative Committee for Space Data Systems, "TC Space Data Link Protocol (CCSDS 232.0-B-4)," 2021.

[10] Consultative Committee for Space Data Systems, "TM Space Data Link Protocol (CCSDS 132.0-B-3)," 2021.

[11] G. Kazz and E. Greenberg, "CCSDS Next Generation Space Link Protocol (NGSLP)," in *SpaceOps 2014 Conference*, (Pasadena, CA), American Institute of Aeronautics and Astronautics, May 2014.

[12] D. Fischer, I. Aguilar Sanchez, B. Saba, G. Moury, B. Bailey, C. Biggerstaff, H. Weiss, M. Pilgram, and D. Richter, "Finalizing the CCSDS Space-Data Link Layer Security Protocol: Setup and Execution of the Interoperability Testing," in *AIAA SPACE 2015 Conference and Exposition*, (Pasadena, California), American Institute of Aeronautics and Astronautics, Aug. 2015.

[13] D. Fischer, M. Spada, and D. Koisser, "SpaceSecLab: A Representative, Modular Environment for Prototyping and Testing Space-Link Security Protocols End to End," *Springer International Publishing*, pp. 375–394, 2017.

[14] S. Klabnik and C. Nichols, *The Rust Programming Language, 2nd Edition*. No Starch Press, 2023.

[15] G. A. Campbell and P. P. Papapetrou, *SonarQube in action*. Manning Publications Co., 2013.

[16] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," *ACM Sigplan notices*, vol. 42, no. 6, pp. 89–100, 2007.