Project Report

Access control and securing of the NUTS uplink

Vilius Visockas

Stig Frode Mjølsnes, Academic Supervisor

Roger Birkeland, Project Supervisor

2011-09 / 2011-12

Abstract

My project is access control and securing student satellite uplink. The constrain of only considering uplink is external. Due to international agreements downlink is not allowed to be encrypted for CubeSats.

My goal was to take systematic approach and analyze security risks and estimate its importance. Taking into account hardware and other limitations, I aimed to design security mechanism to ensure communication message privacy and integrity. Part of research was to investigate approach to security in other CubeSat standard projects.

First I analyzed the biggest security risks. It turned out that message integrity is key aspect for uplink. Secure protocol should ensure that commands sent from the ground station are from authorize entity. However, generally risk of attack is low, mainly because of equipment needed and lack of possible pay-off for attacker.

Then I gave overview of other CubeSat projects closely related to one designed at NTNU. Among several which had well documented design, only few revealed details of actual communication protocol. It turned out that security and integrity issues are not approach at all – it only provides cyclic redundancy checksums to check for transfer errors. Therefore it could be said that security was obtained by obscurity.

After initial setup, there was need to start developing communication protocol. Communication sub-team of project considered either implementing such libraries from scratch, either looking for off-the-shelf solution. The latter was chosen, because software part of project is already lagging in time. We chose open source library CSP (CubeSat Space Protocol) as good candidate for our project. It provides such features as packet routing, sockets, reliable and unreliable packet transportation. In addition, it already had built in security mechanisms: message integrity and privacy protection. That was good material for research and analysis of given design.

The next step was to read and understand the code of CSP library. It appeared that actual security primitives were borrowed from another libraries and not written by authors themselves. Authors used code from “LibTom” cryptography suite. XTEA algorithm was used for encryption and HMAC-SHA1 for integrity protection. There was no big need to check whether implementation of primitives was correct, because library was quite stable and well known. However, I focused on how primitives were used and how it can be improved.

One potential risk found was that integrity check value (HMAC) did not include packet header values, just packet payload. In addition, HMAC does not include sequence number or fresh data, therefore it is vulnerable to replay-attack.

As for encryption, XTEA was trade off between efficiency and security. At least as alternative, other state of art block chipper (like AES) could be implemented.

Last, but not least, it was not addressed what happens if preshared key used for encryption and/or authentication is compromised.

After having analyzed CSP protocol regarding to security, I came up with suggestions for design upgrade. First issue was to have abstraction layer on security and possibility to choose another method of encryption. For this, I have done research for cryptographic suites and looked for simple and reliable AES implementation. In my code prototype I used code snippets from “OpenSSL” library. The best mode of operation looked CCM as with WPA2. First, we add integrity check value and then encrypt using counter mode. As for addressing secure exchange part, I wrote snippet which used Diffie Hellman for secure new key agreement.

To conclude, analysis of CSP led of insights what design could meet security requirements for NUTS project. In addition, it seems that NUTS is one of the first projects considering this issue.

Table of Contents

Acronyms

AES	Advanced Encryption Standard
ADCS	Attitude Determination and Control System
AUS	Authentication Center
CAN (bus)	Controller Area Network
CCMP	Counter Mode with Cipher Block Chaining
CFB	Cipher FeedBack mode of encryption
CRC32	Cyclic Redundancy Check Algorithm (32 bits)
CSP	CubeSat Space Protocol
CTR	Counter mode of encryption
CubeSat	Cube Satellite
DES	Data Encryption Standard
ECB	Electronic CodeBook Mode for encryption
GSS	Ground Station Server
HMAC	Hash-based Message Authentication Code
IV	Initialization Vector
MCC	Mission Control Center
NAROM	Nasjonalt senter for romrelatert opplæring (Norwegian)
NUTS	NTNU Test Satellite Project
OFB	Output FeedBack mode of encryption
RC4	Encryption algorithm by Ron Rivest
SHA-1	Secure Hash Algorithm
TLS	Transport Layer Security

Introduction

In the introduction I will describe NUTS student satellite project from broad perspective. I will also briefly discuss internal organization of work in project team. Then I state the goal of my assignment and give stricter definition of scope.

NUTS project background

NUTS is student satellite project in Trondheim, part of national student satellite project organized by NAROM. Main goal of the project is to educate students, get them interested in space technology. Desirable goal is build fully functional prototype. The satellite should have two way radio link, and has infrared camera as payload. For basic technical specifications, see informational poster [ROG11] and refer to *Appendix B. Technical Specifications*.

Project team consists of approximately 15 master students working on different areas of satellite design. In the following section I will briefly describe how work was organized.

Team was split into smaller manageable sub-teams, such as payload, ADCS, communication team. In addition to individual work, some work was related to cooperation with other members in sub-team. It was important to have cooperation in higher level as well, therefore regular weekly meetings were held of all project members. Therefore approximately 20 hours during semester were spent on discussions ant status updates with other project members. Additionally, we organized midterm presentations in order to have feedback about intermediate results. Once in a while, guest lectures (such as from previous project members) were organized.

Goals

My goal was to investigate uplink security of satellite project. My analysis should include reasoning for security requirements, then suggest basic design to meet them. As consequence overview of possible implementations can be provided.

Scope of work

My main goal of research is focus on satellite uplink security. However because security has to built on some communication protocol, some work interleaves with design decisions of communication protocol.

It is important to note that I mainly concentrate on theoretical analysis of security. The reason that software basis is not solid enough yet to provide full implementation details. However, in majority of this work for specific topics it was assumed that CuebSat space protocol will be used.

Uplink security analysis and design

To design security mechanisms for satellite project, I decided to borrow systematic approach suggested in [NN03] used for designing GSM security system. The following are required steps:

1. **Threat Analysis** The aim of this step is to predict all possible threats. It is not important to suggest actions and countermeasures to avoid them.
2. **Risk Analysis** This step measures probability of each threat. It can be expressed as absolute probability, or at least relative probability according to other threats.
3. **Requirements Capture** Based on results from obtained from previous steps, requirements are described.
4. **Design** In this phase, real security mechanisms are designed to meet security requirements. Possibly existing security primitives can be used, or in case needed, new protocols and standards can be developed. However it is a good practice to reuse solutions which worked in the past, because new security primitives slowly gain trust. It might happen that it is not possible to meet all requirements, in this case one should use iterative process and potentially redefine requirements of reevaluate security threats.
5. **Security analysis** It corresponds to testing of suggested security mechanism design. It can cover both automated tools and more creative techniques. This step is used to test whether all main security threats are resolved by suggested design and it does not introduce new ones in implementation.
6. **Reaction** It is not possible to oversee all possible threats and attacks in advance, therefore design hardly be finished at some point of time. After security breach is found after security mechanism is deployed, additional analysis should be carried out and improvements considered.

My scope of analysis will cover steps from 1 (Threat analysis) to step 5 (Security Analysis). Obviously, step 6 (reaction) can be discussed yet, because currently satellite is on design mode. In the following subsections I will cover each of these steps in more detail.

The following sections will go in more detail into each of these steps.

Threat analysis

First, An important external assumption should be mentioned that we only can have encryption on uplink, not downlink.

The risks related to satellite communication link are similar as for any other wireless network communication. Radio signal is insecure channel by itself.

In order to understand what are the risks, it is important to understand types of messages communicated. We can make distinction of two major message types: command data and payload data. Command data includes satellite control commands, exchange of house keeping data. Payload is basically data captured on satellite with scientific value.

First likely risk is that intruder could want to control satellite on its behalf. Therefore control center should include message authentication code which would guarantee that command was issued from authorized operator.

In addition, intruder could read packets if they were not encrypted (privacy issue). It is not an issue with payload data, because it is only sent downlink direction. As for command data there is not big benefit for intruder to know what commands are being sent to satellite, but certainly privacy is preferred.

We should also consider denial of service attack. One example could be attempt to jam satellite with bad packets and disable it from receiving commands from ground station. As another example, one could send commands to turn off all antennas of satellite which would lead to fatal error and end of communication.

Because of only two major entities are involved on original design (ground station and satellite), natural choice is to use preshared secret approach instead of public key infrastructure. This ultimately raises issue about key being compromised, lost or damaged.

To conclude, given the educational flavor of project, biggest threat of the ones mentioned above would be intercepting control of satellite and sending wrong commands.

Risk analysis

Absolute risks Important aspect of security mechanism if satellite project is scale of deployment. It can be assumed that same security mechanisms will be deployed on one instance of satellite. Even open source communication protocol is reused, the order of real systems will be low, up to 10 distinct satellites. This means it is completely different deployment scale as WiFi, for example. Therefore one could apply proportionality principles and put lower security requirements. They could argue that it is expensive and not cost efficient to analyze and to crack satellite which is not commercial. Here is summary of factors which contribute to risk of possible active attacks:

- **- Risk.** NUTS is not commercial, but educational purpose project
- **- Risk.** It is costly to have required equipment for attack.
- **- Risk.** Little payload data and which could be used only for scientific purposes
- **- Risk** Short and unknown time frame of actual satellite operation in orbit.
- **+ Risk.** Students work will be public, therefore protocol will be open to analysis
- **+ Risk.** More appealing than for example cracking WiLAN.

Having all these considerations in mind, we can deduce that is really low probability that active attacks will be executed to satellite.

Requirements

The identified risks and technical specifications of satellite brings requirements for security system. They are listed in order of importance, from highest to lower.

- **Reliable (High priority).** As satellite project itself is complex system and there are many potential points of failure, security mechanisms should be absolutely reliable, correctly implemented and tested. One way to achieve this is to use best practices for other wireless communication protocols and acknowledged security primitives.
- **Highly efficient (High priority).** Communication consists of ground station and satellite. As ground station is capable to handle big amount of information, satellite is based of micro-controllers with less computing power. Despite bottleneck probably will be physical link and big radio signal latency, it is not satisfactory to have slow encryption and authentication. Furthermore, it should be applied only when it is really necessary.
- **Ensuring message integrity (Medium priority).** It is important that control access of satellite can be gained only by authorized entities. Therefore at least

command messages from ground station should be sent together with message integrity code.

- **Ensuring message encryption (Lower priority).** Despite encryption is allowed on uplink only (due to international treaties) and satellite communication data has mainly scientific value, it is still desirable to be able to encrypt at least subset of communication messages.
- **Ability to change keys (Low priority).** As preshared key approach shall be taken, it is desirable need that corrupted key can be changed with new one. It is clear that new key should be exchanged such way that eaves-dropper cannot obtain it.

Overview of security in satellite communication

I looked for other similar projects. Main search criteria was that communication protocol is documented and available in public. The problem is that communication protocol is that subpart of already complex communication subsystem (antennas, transceivers), therefore is usually purely documented.

NUTS project is part of norwegian student satellite program (ANSAT) run by NAROM [AH08]. Therefore there are several other similar projects developed in parallel, namely:

- CubeStar (University Of Oslo Students)
- HinCube (Narvik University College)

Besides projects in ANSAT, there are several other interesting projects [SAT11]

- SwissCube (EPFL Swiss project)
- AUSAT (University of Adelaide)
- OUFTI-1 (University of Liege)
- Goliat (University of Bucharest)
- PW-Sat (Warsaw university of technology)
- Robusta (University of Montpellier)
- XaTCobeo (Vigo University)
- CP6 (Cal Poly University)
- HawkSat-1 (Hawk Institute for Space Science)
- AeroCube 3 (The Aerospace Corporation)
- Libertad 1 (Universidad Sergio Arboleda)

In the following section, I will give interview of security of communication protocol for each of these projects.

OUFTI-1. As for lowest level protocol AX.25 is used. For higher level PUS protocol is used [DIJ10]. Its frames consist of primary header, secondary header, data and error control. Though this protocol provides reliable transfer through ACK's and sequence numbers, it does not address security and integrity.

Goliat. Communication protocol not documented. However there is hint about physical security: “two independent transceivers architecture commanded by two different processors for redundancy” [GOL10].

Robusta. Communication over AX.25 protocol. Commands are sent through simple custom protocol [STE09], which does not define even checksums. Therefore security issues are not addressed.

The SwissCube project was initiated in 2005 by the Space Center EPFL and complies with CubeSat standard. Unlike majority other protocols, its communication protocol is open and described in detail.

Swisscube uses AX.25 protocol as a base and CCSDS protocol on top of that [CG09]. The fields of packet is described in the *Appendix C. Formats of protocol packets*. Packet header describes packet id and sequence control data (there are several channels defined by APID and each of them keeps separate sequence counter). In addition there is packet length field.

Packet data field is of variable size. It contains telemetry data field header, which defines type, subtype of service and timestamp of the packet. It also contains several unused bits. Finally, there is error control field, implemented as checksum.

As we can see communication protocol used by SwissCube provides neither security, nor authenticity. Only CRC checksum is provided which is used for error correction.

Conclusions of overview

As we can see existing projects either do not document communication protocol, either security issue is not addressed. However, if packets are defined as variable length, security always could be added on highest (application) level.

CubeSat Space Protocol

To start, there should be distinction between application level protocol and transport level protocol. Application level protocol should focus on command format and how to transfer payload, not taking into account reliability and security. The current draft of application is not scope of this project, but current draft which was created together with communication team can be found on *Appendix A. Command set format (Draft)*. My project focused to transport layer protocol, because it is the one which should provide security.

At current stage of project it is still little work done regarding communication protocol, most effort have been put to engineering and mechanical parts. To speed up development process communication team looked for alternatives to find free off-the-shelf solution for transport protocol. Main concern was to find libraries which would help to do packet routing, addressing and reliable packet transfer. Security features were desirable, but not the main goal. After some research we found Cube Space Protocol, which suited our needs. Though final decision is not made yet, major part of this report assumes that this protocol will be used on satellite project and analyze its security.

Cube Space Protocol is network layer protocol designed for satellite systems. Idea was started by students from Aalborg university in 2008 and was supposed to be used in mission on 2011. Originally it was designed for CAN bus, but later support for other interfaces was provided (I2C and RS232). The main features of protocols includes the following [CSP11]

- Router core with static routes. Supports transparent forwarding of packets over e.g. space link.
- Simple API similar to Berkeley sockets.
- Support for both connectionless operation (similar to UDP), and connection oriented operation (RFC 908 and 1151).
- Service handler that implements ICMP-like requests such as ping and buffer status.
- Support for loopback traffic. This can e.g. be used for Inter-process communication between subsystem tasks.
- Optional Support for broadcast traffic if supported by the physical interface.
- Optional support for promiscuous mode if supported by the physical interface.
- Optional support for encrypted packets with XTEA in CTR mode.
- Optional support for RFC 2104 authenticated packets with truncated HMAC-SHA1.

Protocol packet structure

Current protocol packet header is structured as depicted in the following table [WCS11].

Bit offset	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
0	Priority		Source					Destination					Destination port						Source port						Reserved				HMAC	XTEA	RDP	CRC
32	Data (0 – 65535 bytes)																															

Table 1: CSP protocol packet header structure

As we can see from the table 1, packets consists of 32 bit length header and data itself. Priority defines importance of packets and internally identifies packet queue. Source and destination defines end nodes (for example, satellite or ground station). Ports define service the packet is directed to. Ports 0 – 7 are reserved for general services as ping and other ports can be used for subsystem specific tasks. Four flags indicated whether packet is encrypted (XTEA), includes integrity check value (HMAC), includes cyclic redundancy check value (CRC). All payload, potentially bigger in size of encryption/integrity check is stored just right after 32bit header. AX.25 is used as lower level protocol to send actual frames.

Architecture of CSP protocol

CSP protocol is written for GCC compiler. It is mainly designed for POSIX systems, primarily because it uses pthreads for enabling threading. The following table gives summary of components in the library.

Module	**Purpose**
arch	System dependent module implementations (mutex, queues, threads, time)
crypto	Modules ensuring privacy and authenticity
interfaces	Link layer interfaces libraries
transport	Transport level implementations (UDP and RDP)
csp_buffer	Lock Synchronized library for managing memory
csp_conn	Initializing, opening and closing connection between nodes
csp_crc32	Helper module for calculating CRC32 checksums
csp_debug	Helper module for debugging
csp_endian	Wrappers to obtain required endianess in message formats
csp_io	Sending and encrypting packets, usage of sockets
csp_route	Main routing task usage to forward packets to particular interfaces
csp_service_handler	Wrapper interface for all services
csp_services	Implementation of each service (ping, reset, etc)

Table 2: Main components of CSP library

Security model in CSP

Security functions are applied done just before sending out the packet. Security is not applied when the packet is targeted to the same node itself. Depending what is enabled on compiled CSP library, procedure of encryption and authentication takes the following steps (any of them may be skipped).

With Obtained packet to be encrypted:

1. Append HMAC
2. Append CRC32 checksum
3. Encrypt packet using key and nonce which is appended to the message

This follows usual approach used in most established security standards. First it calculates messages digest using sender's key – it ensures that message is sent from node who knows the key. Then CRC checksum can be calculated which helps to prevent transfer errors. Finally message is encrypted using symmetrical key. To prevent replay-attack a random nonce is added to the message which is used as initialization vector (IV). However there is no requirement that it should be unique, therefore it is no guarantee agains replay protection.

The main issue is how the secret it shared. Basically secret key for signature and encryption calculated is symmetrical and hardcoded in library. Is it correct approach? Is it enough for this project security requirements?

Symmetrical shared secret has strong advantage that it is simple to manage keeping in mind that there probably going to be just two main nodes (space segment and ground segment). So far just one ground station is planning to get access to satellite, but things may get more complicated if it was more. It would be difficult to ensure that satellite speaks to correct controller.

Security analysis in CSP

First I will look into security primitives of CSP, namely HMAC_SHA1 and XTEA.

HMAC is used for integrity protection in CSP. It is constructed using SHA-1 algorithm. It has been published early in 1995 and has been security standard. However, in 2005, security flaws were identified in SHA-1, namely that a mathematical weakness might exist, indicating that a stronger hash function would be desirable [SCH95].

Important threat is that authentication code is calculated only on application level data, but not contains CSP protocol header fields (priority, source port, destination port, source and destination). Though it is difficult to send fake command, because it is on application level and put as data payload, it is still possible to think of scenarios for denial of service attacks.

As for encryption, CSP uses XTEA (Extended Tiny Encryption Algorithm) algorithm [XTE11]. It is 64 bit block cipher, using 128bit key and Feistel network with 64 rounds approach. As main advantage, it is extremely easy to implement. Cryptanalysts have been studying algorithm and there are several findings. Y. Ko described related key differential attack on 27 rounds of XTEA [KHL04]. J. Lu pointed out related-key rectangle attack on 36 rounds of the XTEA block cipher [JIQ09]. These attacks are mainly theoretical. As privacy (encryption) was identified as lower importance goal than integrity, this can be acceptable.

To conclude, both security primitives used are quite out date in fast changing security world. It is desirable to have improved algorithms or at least an option to do so.

Improving CSP security

As previously indicated, security primitives used in the CSP are not the most secure ones. I decided to approach encryption security first.

Despite CSP has security provided with XTEA protocol, it may be enhanced by using AES protocol for encryption. The reason is that AES is considering really strong and been de facto standard for various security sensitive tasks. However we should keep in mind that it is more computationally expensive. Therefore we should provide an option of choosing encryption algorithm. Obviously both segments (ground station and satellite) should use the same encryption method. Because satellite system is already quite complex, the decision about security mechanism should be made in advance and built in in the code. Having protocol to negotiate encryption method, may lead to unwanted complexity.

To add AES support in library I had aimed to reuse working snippet. Therefore I did research what open source cryptographic suites are available and what encryption modes they support.

Overview of cryptography libraries

CSP satellite protocol already has some security, in particular HMAC and XTEA. The implementation of these two algorithms were borrowed from *libtom* library [LBT11]. LibTom Projects are open source libraries written in portable C under WTFPL license. Despite minor advantage that it has been a while while library was actively supported (last change in 2007), it has fairly good and simple implementations of security algorithms in C. This project is basically initiative of one person – Tom St Denis.

Another possible alternative is crypto++ suite [CRY11]. Crypto++ Library is a free C++ class library of cryptographic schemes. It covers many HMAC, cipher, public key infrastructure algorithms. However the library modules are written in C++ therefore not suitable for reusing in CSP implementation.

Finally, there is well known project OpenSSL which is collection of cryptographic modules. It is full featured, and provides implementations for TLS and SSL too. Important part of suite is command line tools which are actually used very often to generate and sign various keys. There are some well structured books, which describe major functionalities of OpenSSL, for example the one written by John Viega, Matt Messie and Pravir Chandra [VMC02]. Having this said, it makes good candidate of reusing some code for extra security features.

AES implementation in OpenSSL

Because OpenSSL is quite big library and covers many features, AES implementation initially is extensive: supports many encryption modes, have some convenience methods. The aim should be to have as much functionality as actually needed. Therefore, instead of adding support of all encryption modes, we should use the one which fits best project needs.

Overview of implemented modes. In the table summary of encryption modes is provided by OpenSSL [BCM11]. As we can see, ECB mode is not even implemented as consider not secure enough.

Mode name	Behavior	Comment
CBC	In the *cipher-block chaining* (CBC) mode, each block of plaintext is XOR'ed with the previous cipher-text block before being encrypted	Popular mode
CFB	The *cipher feedback* (CFB) mode, a close relative of CBC, makes a block cipher into a self-synchronizing stream cipher.	Similar to CBC
OFB	The *output feedback* (OFB) mode makes a block cipher into a synchronous stream cipher It generates keystream blocks, which are then XOR'ed with the plaintext blocks to get the ciphertext	Encryption and decryption are the same; blocks may be processed in parallel
CTR	Like OFB, *counter mode* (CTR) makes block cipher into synchronous stream cipher It generates next keystream block by encrypting successive values of “counter”.	

Table 3: Supported encryption modes in OpenSSL

Stream cipher vs Block cipher

In this section a short introduction of stream cipher and block cipher will be given. After that, given constrains of satellite project, advantages and disadvantages of both approaches will be considered.

In block cipher, plain text is divided into blocks of bits (for example of size 128 bits = 16 bytes). It last block is not complete, it is then padded. As basic concept, the encoding of each block runs independently. However, there exists some encryption modes which adds result of previous block in order to prevent replay attacks. Examples of block cipher includes DES and AES.

In contrast, when stream cipher is used, plain text bits are combined with pseudo random bit stream. This pseudo random stream is generated using some initial seed value. We can look into this stream as one time pad. Examples of stream cipher include RC4 (which was used in WEP). There are two kind of stream ciphers *synchronous* and *self-synchronizing*. In synchronous cipher key stream is generated independently and then combined with plain text, usually using exclusive OR operation bit by bit.

Which approach would be preferred for NUTS project encryption? NUTS wireless link is not much different from other wireless networks such as 802.11. WEP uses RC4 stream, for example, while WPA2 uses AES cipher in CCMP mode (which is essentially counter mode). The reason why essentially stream cipher is used, is that it does not require to pad data to be exactly of block size. For small packets, which will be probably the case for NUTS communication, it saves lot of potentially extra payload. Other advantage is that one could precalculate bit stream and applying it with plaintext is really efficient. However, this hardly will be needed and implemented on NUTS project, where biggest efficiency bottleneck is signal latency rather than encryption.

To conclude, block cipher such as AES operating in counter mode is good option for our case. Of course, it is important that initial counter value is random. AES combined in counter mode is essentially the same as using stream cipher. OFB mode also could be good substitute for counter mode to simulate stream cipher.

Implementation. Having all this said, it seemed that reusing OpenSSL code for AES in CTR or OFB mode to add another option for encryption is good approach. I created prototype code for extra feature. Because originally protocol was designed for using XTEA, there was need for minor code refactoring. There should be abstraction level in the code, which would allow to chose implementation in a flexible way. Sketch code can be found in *Appendix E. Adding AES support*. All other available related files can be found on the GitHub [VV11].

Efficiency. Efficiency for running security related functions was one of the highest priority requirements. After adapting AES for CSP, I decided how efficient it is compared to previously available XTEA. For the test, I simply encrypted 1.000.000 packets on my Intel Core2 Duo CPU T7500 2.20GHz machine running Ubuntu. It did not include packet routing (it will definitely take major part in actual deployment), but only compared actual encryption time instead.

	XTEA	AES
32 byte packets	1.90 seconds	2.41 seconds
128 byte packets	1.91 seconds	2.36 seconds

Table 4: Speed comparison of encryption between AES and XTEA

Important conclusion can be drawn here that adding more advanced encryption algorithm may be not considerably slower. However it is future task to actually try whether micro-controllers support OpenSSL implementation and what is added time when packets being routed (even to the loopback interface).

Additional satellite security topics

This section is dedicated to advanced topics in satellite communication security. First I will introduce Genso project. Then I cover some details how new key can be exchanged in secure way.

Ground station cooperation

One of limitation for satellite is small time window when commands and payload can be exchanged with ground station. There is ongoing community effort to tackle this issue, namely *Genso* project.

Genso is open project which should allow get and sent data to satellites using distributed ground station networks. The project announced to release code in near future in the time of writing. Having ground stations distributed geographically enables to exploit satellite more efficiently. However, this can cause security issues. This section will take deeper look at architecture of *Genso* project based on [GEN11] and some security considerations.

Architecture. Each operator of ground segment which participate in *Genso* project runs Ground Station Server (GSS). Operators who control satellite run Mission Control Client (MCC). It contacts GSS for exchange data/control. Additional entity called Authentication Server (AUS) has goal of authenticating GSS and MCC.

Authentication Server (AUS). Despite its name, it does not only provide authenticity. AUS also serves as database of events – it is possible to query and trace when some satellite passed some ground station. It also contains housekeeping data (status, for example) about each ground station and sends it when requested. On preliminary design therese is only centralized AUS server.

Ground station server (GSS). As mentioned before, it can be used to get downlink data or send commands. In order to send commands, it needs permissions for ground station itself, satellite operator and local laws. There exists scheduling possibility.

Mission Control Client (MCC). The client is used to send and receive data for particular satellite. Given MCC is authenticated with AUS, data is automatically transferred from GSS when available. In addition it is also possible to send control commands to trusted GSS.

Advantages and disadvantages of participating of Genso. Using *Genso* has many good outcomes. Naturally, more data can be transferred using more ground stations. It is possible to reach satellite quickly in emergency situation. It is also possible to provide orbit

estimate. Benchmarking of communications becomes possible.

However there can be also some dangers also. The project is on early stage of development, and additional management level can add extra complexity and possibility to fail mission. Centralized system also infers that AUS is single point of failure. Having distributed systems requires to act on common protocols. However technical details are not available yet, therefore it is difficult to infer more about security of system.

Despite that it can be inferred where security will be concern. It is important how authentication server recognizes clients. Reasonable guess that it is being handled by using public key infrastructure. In addition, only one authentication center (taking into account its other responsibilities) is highly risky and one must ensure that it does not leak information and can cope with active denial of service attack.

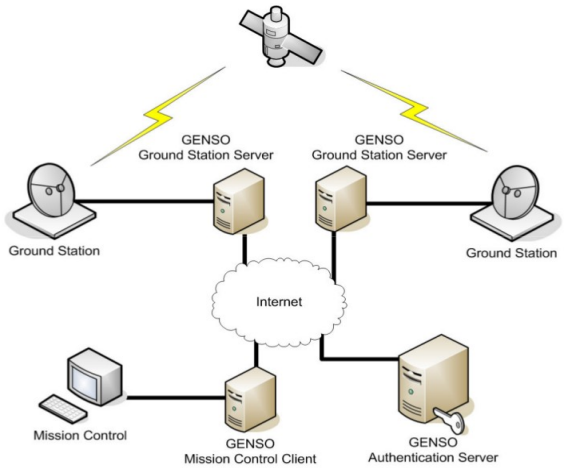

Illustration 1: Genso project infrastructure. Genso ground servers and mission control clients communicate through Internet.

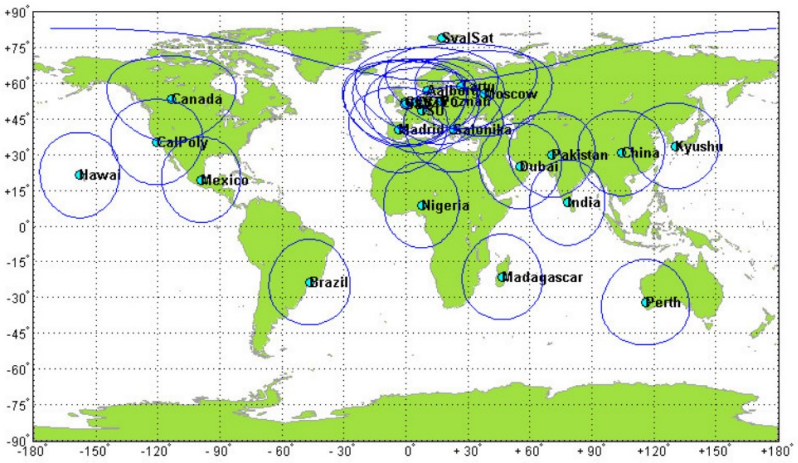

Illustration 2: Extension of control area through network of ground stations

Establishing new key using Diffie-Hellman

Currently CSP has no secure way of changing encryption or integrity keys after satellite has been launched, because keys are hardcoded into CSP protocol library. The idea is adding one of secure key exchange algorithm to CSP as a service.

Diffie Hellman algorithm is secure and often solution for this problem. According to it, A and B agrees on some prime number p and generator number g. A choses its secret a and sends B information about his secret encoded in A. B does the same. They both use extra information from other side to calculate common shared secret s. The process is summarized in the table.

A				**B**		
Secret	*Public*	*Calculates*	*Sends*	*Calculates*	*Public*	*Secret*
a	p, g		> p, g			b
a	p, g, A	$G^a \bmod p = A$	> A		p, g	b
a	p, g, A		< B	$G^b \bmod p = A$	p, g, A, B	b
a, s	p, g, A, B	$B^a \bmod p = s$		$A^b \bmod p = s$	p, g, A, B	B, s

Table 5: Secure key exchange using Diffie Hellman algorithm. Based on [DH11]

Implementation. CSP has implemented service model in abstraction level and therefore it is possible add your own system services. Service handler API is in the service handler module. If the given packet is a service-request (that is uses one of the CSP service ports), it will be handled according to the CSP service handler. In order to listen to CSP service ports, one has to bind listeners to the CSP_ANY port.

I have written small proof-of-concept example snippet to test Diffie Hellman idea (see *Appendix D. Secure key exchange)*. To actually implement it in CSP, it is needed to extend it over CSP service model as additional system service. My version generates 64 bits length secret. This is due to int64 data type is used. If longer key is needed, either arbitrary precision arithmetics module should be used, either key exchange repeated several times two combine into the bigger secret.

Discussion

Investigations during this project raised some questions which had no clear answer.

First, It raises the issue how secure protocols should be designed. CSP was designed choosing explicit implementations of security algorithms: HMAC and XTEA. However best practices allow abstraction level on integrity/privacy protection. For example, access control in RSN is managed through EAP. One can chose any algorithm which complies with EAP. This gives great flexibility. In contrast, CSP was designed without possibility to chose another security primitive. My intent was to add this simple level of abstraction so that one could chose encryption algorithm at least at compile time.

This project revealed always existing trade-off among security and efficiency. That may be main reason why many similar projects do not secure their communication protocol at all. They might argue that this way we loose potential bandwidth to add security features (HMAC, IV payload).

Another interesting idea is how one implements security primitives. Creators of CSP protocol chose using existing libraries approach. One possible limitation of this choice is license type with which it is bundled. For example, it may be the case that usage of library will require whole project to be be open source. Another important thing to consider is how well library is supported. This issue is relevant to this project, because libtom cryptosuite library (used in this project) has been not updated recently.

Conclusions

After reviewing other CubeSats it became clear that security Is not prioritized issue. One of the reasons might be non commercial nature of these satellites. Another possible is reasoning that it takes too much effort to crack the security protocol. Finally student satellite projects may lack resources, in particular people looking into this problem.

General analysis of satellite communication suggested required security design. As I tried to investigate security risks in systematic way, it appeared that message integrity is the most important aspect for this project. Fact that communication is run between two entities (ground station and satellite) makes preshared key scheme reasonable choice. To make it more manageable, I suggested Diffie Hellman secure key exchange mechanism. Stream cipher or block cipher which can simulate one time pad appears to be best choice for encryption in high latency radio link.

More detail research of CSP protocol revealed that it can be improved. It uses security primitives (SHA1 and XTEA) which might be considered out of day currently and are reported to be broken. My suggestion is to add AES support and additional abstraction level which would allow to chose security primitives. In addition, CSP header does not contain any sequence number except initialization vector which is not defined to be unique. As consequence, replay attack is possible. On the other side, it makes protocol stateless which is advantage to be run in embedded systems.

As side effects of analysis, I developed sketch code snippets for adding AES and trying out Diffie Hellman. In addition, I had opportunity to work in team and to learn how satellite work in general.

To conclude, CubeSat space protocol is one of the first attempts to create open source protocol for satellite communication. With gaining interest in space technology, CSP has potential to be choice for many new projects. Therefore this work can be used as guideline to continuously improve protocol.

Further work

This work identified main security risks for satellite project in general and analysis of CSP protocol was given. In addition, it has been considered to be extended o meet required security needs. However there is still need for further work.

It is important that NUTS project team would agree on communication protocol that will be used. In case CSP is not used, all security requirements presented here should be ported. On the other side, if CSP is to be used, then security mechanisms already present and improvements suggested here should be tested. It is important, that microprocessors are capable of performing of more advanced security tasks.

Despite mentioned in this work, I did not focus on denial of service attacks. It would be interesting to have protocol in action, and try to disrupt service by sending junk signals. It can be black box testing, where truly random data sent on same frequency, or signals which comply to communication protocol.

Another interesting area is feasibility of joining *Genso* network. On the time of writing it is just concept and actual code is not released yet. Current team acknowledged advantages of being member of ground station network and therefore it should be considered. Security and integration can be broad topic for itself.

Coding tasks can be also extended. Idea of Diffie Hellman can be implemented in CSP as service (like PING or other already existing services). SHA-1 used in HMAC calculation was identified as getting out of date, therefore a safer implementation can be adapted. All code provided is not completely finished and tested.

References

AH08: Jøran Antonsen, Torbjørn Houge, The Norwegian Student Satellite Program, ANSAT, 2008.

BCM11: Wikipedia, Block Chiper Modes of Operations, 2011,http://en.wikipedia.org/wiki/Block_cipher_modes.

CG09: Benoit Cosandier, Florian George, Telemetry Packet Definitions for SwissCube, 2009.

CRY11: CryptoCpp, CryptoCPP library and documentation, 2011,http://www.cryptopp.com/.

CSP11: CSP team, Cubesat Space Protocol Webpage, 2011,http://code.google.com/p/cubesat-space-protocol.

DH11: Wikipedia, Diffie Hellman key excange, 2011,http://en.wikipedia.org/wiki/Diffie–Hellman_key_exchange.

DIJ10: Sebastien De Dijcker, Implementation of telecommands in nanosatellite OUTFI-1, 2010

GEN11: Wikipedia, Genso Project Website, 2011,http://www.genso.org.

GOL10: Team from University of Bucharest, GOLIAT, Romanian CubeSat Project,

JIQ09: Lu Jiqiang, Related-key rectangle attack on 36 rounds of the XTEA block cipher", .

KHL04: Youngdai Ko, Seokhie Hong, Wonil Lee, Sangjin Lee, Jongin Lim, Related key differential attacks on 27 rounds of XTEA, 2004.

LBT11: Tom Denis, LibTom Cryprolibrary Website, 2011,http://www.libtom.org.

NN03: Valtteri Niemi, Kaisa Nyberg, UMTS Security, 2003

ROG11: R. Birkerland, NUTS poster,

SAT11: Wikipedia, List Of Cubesats, 2011,http://en.wikipedia.org/wiki/List_of_CubeSats.

SCH95: B. Schneier, Cryptanalysis of SHA-1, 1995.

STE09: Perez Stephanie, Robusta Interface Specifications, 2009.

VMC02: John Viega, Matt Messier, Pravir Chandra, Network Security with OpenSSL, 2002

VV11: Vilius Visockas, CSP patches sources, ,https://github.com/visockas/CSP.

WCS11: Wikipedia, Cube Space Protocol article, 2011,http://en.wikipedia.org/wiki/Cubesat_Space_Proto.

XTE11: Wikipedia, XTEA encryption algorithm, 2011,http://en.wikipedia.org/wiki/XTEA.

Appendix A. Command set format (Draft)

The following section is result of group work of communication team in NUTS project. The following document is draft of application level communication protocol and has been discussed and developed in several meetings together with Dan Erik and project coordinator Roger Birkeland. It was important to have such draft at the beginning of project work to allow further design decisions.

This document describes the command format for the modules within NUTS. This document does not describe the means to deliver those commands, but the commands are expected to be delivered unchanged and in-order. It is not required that commands are idempotent.

Command overview. These are the basic ideas of commands:

- Two types of commands: Requests and responses
- Request commands are fixed length
- Response commands are variable length
- Some, but not all requests, are paired with a response

The rationale for having fixed length request commands, are that most of the requests does not involve the transfer of huge amounts of data. In that regard, the relationship between the initiator and the responder is typically asymmetrical. There are some exceptions to this rule: For instance, some requests involve a variable amount of data with a size exceeding the possible payload for a single request. In that case, it is requirement to use multiple requests. This implies that the recipient has to implement some kind of state machine.

Byte endianess. All command fields are integers stored in big endian format. The exception to this, is the payload field.

Request format. The following table describes request format.

16 bits	**48 bits**
Command type	Command Argument

The table below illustrates commands by set of examples in draft form.

Request Type	Description	Parameter
CAM_TAKE_PIC	take picture	
CAM_SET_EXP	Exposure time	exposure time in milliseconds
CAM_SET_RES	Resolution	
OBC_TAKE_PIC	Schedule CAM_TAKE_PIC	When the CAM_TAKE_PIC should be issued to the camera, in seconds relative to now
GEN_GET_STATUS	Get sensor status	
ADCS_DETUMBLE		
ADCS_EARTH_ORIENT		
ADCS_ROT_X		Target rad/sec
ADCS_ROT_Y		Target rad/sec
ADCS_ROT_Z		Target rad/sec
GEN_SYSTEM_RESET		
GEN_RESET_SUBSYS		Module number
GEN_PWR_ON_SUBSYS		Module number
GEN_PWR_OFF_SUBSYS		Module number
RADIO_BEACON_ON		
RADIO_BEACON_OFF		
RADIO_BEACON_START		
RADIO_BEACON_END		
RADIO_BEACON_ON		
RADIO_BEACON_OFF		

Response format. On successful receival and handling of a request, the success bit will be set high.

Length	1	7	1	7	24	Up to 2^{24}
Purpose	Success	Command Type	Last Response	Options	Length	Payload

Exceptions. Not all entities connected to the bus, are able to interpret this command set. Notable exceptions are sensors for temperature, current and voltage.

Interrupts and faults. In the event that a transaction is interrupted, the effects of the completed parts of the transactions will be lost. There is no requirement that nodes are able to resume transactions in the event of e.g. a power-down event.

Issues. The communication library has to be able to handle link-layer specific constraints, like timing constraints on the I2C-bus.
Multiple masters may want access to the bus at the same time, both of them with the purpose of transferring huge amounts of data. How are we going to handle bus arbitration with no central arbiter or arbitration hardware?

How to enforce fairness on the bus? (introduce wait stages)

Appendix B. Technical Specifications

Our satellite is a double CubeSat. We have our own design, using a backplane instead of stacked PC-104 cards.

OBC Features

AVR32UC3A3256 32-bit MCU
SRAM, NAND-flash, OTP “master image storage”
Provide storage and processing for payload and other sub-systems
Act as backplane master; can control everything

COMM Features

145 MHz transceiver for downlink
437 MHz transceiver for uplink from Analog Devices
Act as backplane master; can control everything

Ground Station Features

X-ed Yagi antennae for VHF and UHF
Working on Helix-antenna for UHF
IC-9100 radio
Yaesu 5500 antenna rotor

Appendix C. Formats of protocol packets

Format of communication message header in *SwissCube* satellite project.

Packet Header (48 bits)							Packet Data Field (Variable)		
Packet ID				Packet Sequence Control		Packet Length	Telemetry Data Field Header	Source Data	Packet Error Control
Version	Type	Data Field Header Flag	APID	Grouping Flags	Source Sequence Count				
3	1	1	11	2	14				
16				16		16	64	Variable	16

Appendix D. Secure key exchange

```c
#include <stdio.h>
#include <stdint.h>
#include "dh.h"
#include <time.h>

// a^b mod c
int64_t dh_pow_mod(int64_t a, int64_t b, int64_t c) {
    int64_t tmp = 1;
    int counter;
    for (counter = 0; counter < b; counter++) {
        tmp = (tmp * a) % c;
    }
    return tmp;
}

void db_randomize() {
    unsigned int iseed = (unsigned int)time(NULL);
    srand (iseed);
}

void dh_create_private(DH* keys) {
    // Token A = g^private mod p
    keys->private = rand() % MAX_PRIVATE_KEY;
    keys->token = dh_pow_mod(keys->g, keys->private, keys->p);
}

void dh_create_secret(DH* keys, int64_t token) {
    // secret = token^private mod p
    keys->key = dh_pow_mod(token, keys->private, keys->p);
}

void dh_init(DH* keys) {
    keys->g = 5;
    keys->p = 9223372036854775783;
    keys->key = 0;
    keys->private = 0;
    keys->token = 0;
    keys->public = 0;
}

void main() {
  db_randomize();
  DH* h1 = (DH*) malloc(sizeof(DH));
  DH* h2 = (DH*) malloc(sizeof(DH));
  dh_init(h1);
  dh_init(h2);
  dh_create_private(h1);
  dh_create_private(h2);
  dh_create_secret(h1, h2->token);
  dh_create_secret(h2, h1->token);

  /* Should match */
  printf("Private key: %lld, %lld\n", h1->private, h1->key);
  printf("Private key: %lld, %lld\n", h2->private, h2->key);
}
```

Appendix E. Adding AES support

This code adds abstraction level in compiled source

```
#include "csp_aes.h"
#include "csp_xtea.h"

//#define CSP_CHIPER_XTEA   1
#define CSP_CHIPER_AES          1

// XTEA chiper implementation
#ifdef CSP_CHIPER_XTEA

int csp_encrypt(uint8_t* plain, const uint32_t len, uint8_t * buffer) {
    uint32_t iv[2];
    iv[0] = (uint32_t) *buffer;
    iv[1] = (uint32_t) *(buffer + 4);
    return csp_xtea_encrypt(plain, len, iv);
}

int csp_decrypt(uint8_t * cipher, const uint32_t len, uint8_t * buffer) {
    uint32_t iv[2];
    iv[0] = (uint32_t) *buffer;
    iv[1] = (uint32_t) *(buffer + 4);
    return csp_xtea_decrypt(cipher, len, iv);
}
#endif

// AES Chiper implementation
#ifdef CSP_CHIPER_AES

int csp_encrypt(unsigned char * plain, const uint32_t len, uint8_t * buffer) {
    return csp_aes_encrypt(plain, len, (unsigned char*) buffer);
}

int csp_decrypt(unsigned char * cipher, const uint32_t len, uint8_t * buffer) {
    return csp_aes_decrypt(cipher, len, (unsigned char*) buffer);

}
#endif
```

```
#include <stdint.h>
#include <string.h>

/* CSP includes */
#include "minimized.h"
#include "aes/aes.h"

#define CSP_ENABLE_AES      1


#if CSP_ENABLE_AES

#define AES_BLOCKSIZE       8
#define AES_KEY_LENGTH      16
#define AES_KEY_BITS        128
#define AES_BUFFER_SIZE     1024
#define AES_IV_SIZE         20
#define AES_IV_ACTUAL       8


unsigned char ckey[] =  "thiskeyisverybad"; // It is 128bits though..
AES_KEY encrypt_key;
AES_KEY decrypt_key;


int csp_aes_set_key(char * k, uint32_t keylen) {
    if (AES_set_encrypt_key(ckey, AES_KEY_BITS, &encrypt_key)) {
            return -1;
    }

    if (AES_set_decrypt_key(ckey, AES_KEY_BITS, &decrypt_key)) {
            return -1;
    }

    return 0;
}

int csp_aes_encrypt(unsigned char * plain,  uint32_t len, unsigned char* iv) {
    unsigned char iv2[AES_IV_SIZE];
    unsigned char buffer[AES_BUFFER_SIZE];

    // Construct IV from the byte array passed
    int ii;
    for (ii = 0; ii < AES_IV_SIZE; ii++) {
            iv2[ii] = iv[ii % AES_IV_ACTUAL];
    }

```

```
48:     //AES_cbc_encrypt((unsigned char*) plain, &buffer, rlen, &encrypt_key, iv2, AES_ENCRYPT);
49:     int num = 0;
50:     AES_ofb128_encrypt((unsigned char*) plain, &buffer, len, &encrypt_key, iv2, &num);
51:
52:     // Copy result of encryption and append IV at the end
53:     memcpy(plain + 0, &buffer, len);
54:     memcpy(plain + len, &iv2, AES_IV_SIZE);
55:
56:     // Added IV
57:     return len + AES_IV_SIZE;
58: }
59:
60: int csp_aes_decrypt(uint8_t * cipher, const uint32_t len, unsigned char* iv) {
61:     unsigned char iv2[AES_IV_SIZE] = {1};
62:     unsigned char buffer[AES_BUFFER_SIZE];
63:
64:     //AES_cbc_encrypt((unsigned char*) cipher, buffer, len, &decrypt_key, iv2, AES_DECRYPT);
65:
66:     // Create IV from the byte array passed
67:     int ii;
68:     for (ii = 0; ii < AES_IV_SIZE; ii++) {
69:         iv2[ii] = iv[ii % AES_IV_ACTUAL];
70:     }
71:
72:     int actual_length = len - AES_IV_SIZE;
73:
74:     int num = 0;
75:     AES_ofb128_encrypt((unsigned char*) cipher, buffer, actual_length, &encrypt_key, iv2, &num);
76:
77:     memcpy(cipher, buffer, actual_length);
78:     return 0;
79: }
80:
81: #endif // CSP_ENABLE_AES
82:
```