LiteCrypto

Team AES - Andrew Wang, Sam (Jiewen) Wu, and Elton Yang

CPE 458: Current Topics in Computer Systems (Cryptographic Engineering)

Dr. Zachary Peterson

December 12, 2014

Abstract

As satellite communication becomes more accessible with the introduction of CubeSats, the need for authenticated and encrypted communication become necessary to protect against attackers. We have developed a lightweight, open-source cryptographic library suitable for satellites, specifically CubeSats, based on the NaCl and TweetNaCl libraries. The library, LiteCrypto, includes authentication, encryption, and authenticated encryption for IPv4 packets under symmetric keys.

Contents

1 Introduction

1.1 CubeSat

CubeSat is a international collaborative project of over 40 educational systems and private firms. A CubeSat is a satellite that is 10 cm^3 and with a mass of 1.33 kg. [3]

1.2 PolySat

PolySat is Cal Polys branch of CubeSat, and was established since 1999. Dr. Bellardo is the current advisor for PolySat. They have eight launched missions and three are currently in development. [7]

2 Motivation and Background

Our group decided on creating a CubeSat Crypto Library for our Crypto Engineering Final Project.

2.1 Background

Dr. Bellardo has discussed with our team that the future of CubeSat is to allow them to move in space. However, their current communications from home base to the satellite is unencrypted and unauthorized. Therefore, an adversary who wishes to harm the CubeSat, or harm another object using the CubeSat, may sniff a command packet to the satellite and control it. Dr. Bellardo has given us a set of requirements that the library should have.

2.2 Requirements

2.2.1 Binary Size

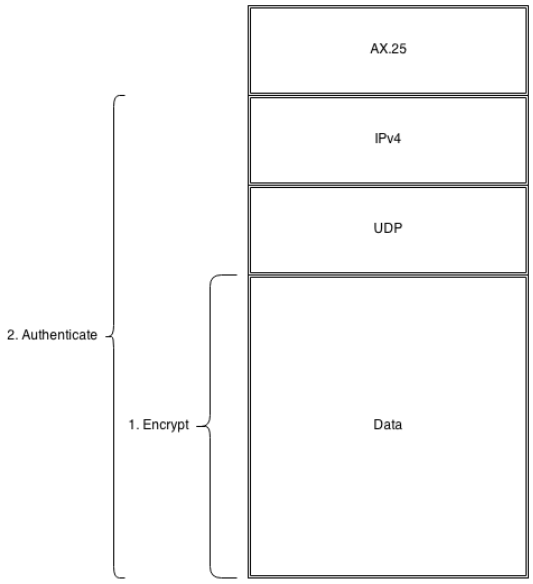
Due to the small size of the CubeSat and the large amount of things it does, there is not a lot of storage space for the library. Therefore, the binary size needed to be around 50 kilobytes. Because the library would be part of its main functions, it will also need to be small for a quick boot time for the operating system.

2.2.2 Data Transfer

Our library can not transfer large amount of data between home base and the satellite. AX.25 speed limits on data are rarely higher than 9,600 bits/s and are usually 1,200 bits/s. [2] The satellite also does not return an acknowledgement packet after successful retrieving a packet. Therefore, we are restricted on which scheme to use for our crypto library. Another issue is that the connection time with the CubeSat is at most 15 minutes an hour due to the satellite orbiting the Earth. Therefore, we cannot use schemes that require a constant connection with the satellite.

2.2.3 Encryption and Authentication

Our library should allow any user to have the option to encrypt or authenticate their packets. It should have 3 functions: No security, one way authentication, and authenticated encryption. The diagram on the next page is how Dr. Bellardo described how we should encrypt and authenticate. We were told to only encrypt the data portion of the packet, and only authenticate the IPv4 and UDP headers along with the data.



3 Related Work/Research

We began our research by looking for current satellite encryption. We found a paper on AES (Advanced Encryption Standard) with LDPC (Low Density Parity Check Code) in order to design a secure and reliable error correction method. [16] We also found that stream ciphers lead to serious problems if used incorrectly for satellite applications. [8] We discovered someone else's work to authenticate IP protocol. [1] In order to meet the low overhead requirement, we found a low cost one-way HMAC proposal for a RFID system. [14] With all of this information, we needed a lightweight software library for encryption to use.

3.1 CyaSSL (wolfSSL)

CyaSSL is a lightweight SSL/TLS library. It advertised a special build called LeanPSK, which was an implementation of CyaSSL that could be built in as little as 20 kilobytes. LeanPSK is also 20 times smaller than OpenSSL, and uses 1 to 36 kilobytes of memory. This configuration required pre-shared keys. [4] However, after creating the binary ourselves and using their library, we found it to be complicated to build and the binary ended up to be 57 kilobytes. That was above our binary size requirement so we looked for a different library to use.

3.2 NaCl

The NaCl crypto library has all of the core operations needed to build our crypto library. NaCl advertises state of the art security, improves usability, and improved speed. However the full NaCl library is too large for us to use, so we found a smaller version of the NaCl library to use. [5]

3.2.1 TweetNaCl

TweetNaCl supports 25 of the C NaCl functions used by applications and fits into a 100 tweets. It is self contained library so it was easy to build. [13] However, it did not have HMAC build into TweetNaCl so we ported it from the original NaCl library.

4 Our Work

4.1 Key Exchange

The LiteCrypto library revolves around symmetric key cryptography. The keys used for authentication and encryption in the LiteCrypto library are pre-shared (or secret) keys. The library expects a key to be provided for all cryptographic functions. In the context of the CubeSat, encryption and authentication keys need to be agreed upon before sending the device for launch into space. Since the LiteCrypto library only provides for cryptographic functions for satellite communications via AX.25/IPv4, the security of the keys in memory or disk used for the library is considered outside the realm of this project. Therefore, we assume the keys used for this

library is appropriately secured by another system.

4.1.1 Tradeoffs Made

We decided not to perform public-key cryptography because we believed it would have been too much overhead for the CubeSat. The amount of data to be inserted into the packets would have been increased. However, TweetNaCl offers public-key cryptographic functions [13], so it is possible to easily implement public-key cryptography in LiteCrypto.

4.2 Authentication

Authentication in LiteCrypto utilizes NaCl library's HMAC-SHA512-256 authentication function. [1] The HMAC-SHA512-256 authentication scheme essentially performs HMAC-SHA512 and truncates to the first 256 bits [11]. This scheme is conjectured to meet the standard notion of unforgeability [11].

The sign function takes in a 32-byte key [11] and variable-length IPv4 packet to output a signed IPv4 packet which has a 32-byte tag [11] inserted in the head of the data portion. [2] The verify function takes a signed IPv4 packet and removes the inserted tag from the head of the data portion. Verification is then performed on the reconstructed packet by generating a new tag to compare with the provided tag. Based on the source files, the comparison in TweetNaCl is done by a constant-time string comparison function to avoid creating side-channels. If the packet verifies, the function outputs the new size of the reconstructed packet. Otherwise, it outputs -1 as $\perp$.

4.2.1 Tradeoffs Made

We chose HMAC-SHA512-256 over TweetNaCl's one-time authentication because we wanted to be able to reuse the keys. With one-time authentication scheme, reusing a key to authenticate more than one message reveals enough information to allow for forgeries [12].

4.3 Encryption

LiteCrypto utilizes TweetNaCl's secret-key encryption scheme based on xsalsa20 for encryption. The xsalsa20 scheme expands the amount of nonce bytes available to use for salsa20 [13]. Because xsalsa20 uses salsa20 as the core, the security of xsalsa20 depends on the security of the salsa20 scheme [13]. The xsalsa20 encryption scheme is conjectured to meet the standard notion of unpredictability [10].

The TweetNaCl encryption function generates a key stream from xsalsa20 using a 32-byte key and a 24-byte nonce [10], so it effectively acts as a stream cipher [13]. The key stream is deterministic in that the same key and nonce pair will generating the same key stream. The key stream is bitwise XORed with the plaintext to produce the ciphertext [13]. Note that the encryption scheme does not hide the length of the message [10]. Because the function acts as a stream cipher, the function can be used both to encrypt or decrypt. We base our encryption function around the use of TweetNaCl's function.

The encrypt function takes in a 32-byte key [13] and a variable-length IPv4 packet to output a IPv4 packet with 24-byte nonce inserted at the head of the encrypted data. The IPv4 packet headers are left untouched in

[1] TweetNaCl did not provide the function, so we ported the function from the full NaCl library over into the TweetNaCl source files for our use.

[2] Because we did get to perform live testing, we are unsure if this insertion breaks the IPv4 system. An alternative process for inserting the tag is possible.

plaintext. The nonce is randomly generated by the function by reading from urandom. The NaCl group supports that the nonces are long enough that randomly generated nonces have negligible risk of collision. [10] The size of the encrypted packet is returned on success; otherwise, -1 is returned.

The decrypt function takes in a 32-byte key [13] and a variable-length encrypted IPv4 packet to output a decrypted IPv4 packet with the nonce bytes removed. The nonce bytes are removed from the head of the encrypted data of the given encrypted packet. The encrypted data is then decrypted using the provided key and nonce bytes. We construct the new packet by appending the decrypted data to the IPv4 headers. The size of the decrypted packet is returned on success; otherwise, -1 is returned.

4.4 Authenticated Encryption

Authenticated encryption in LiteCrypto makes use of the authentication and encryption functions discussed above. To encrypt a packet, we first perform our encryption scheme then our authentication scheme. We have ultimately inserted 56 bytes into the IPv4 packet: 32 bytes from authentication tag and 24 bytes from encryption nonce. To decrypt, we first attempt to verify the packet. If the packet verifies, we perform our decryption scheme and output the decrypted packet without the 56 bytes originally inserted; otherwise, -1 is outputted as $\perp$. Our encrypt-sign and verify-decrypt schemes achieve authenticated encryption by never decrypting a non-authenticated packet.

Because we have intermediate states between encrypt-sign and verify-decrypt, we have dynamically allocated memory to contain the intermediate states. On memory allocation error, the function returns -1. Having the dynamically allocated memory does not decrease the security of LiteCrypto since the intermediate states only contain the IPv4 header which is left in plaintext at all times, the nonce bytes, and the encrypted data. An adversary would not gain any more information than if he or she caught an encrypted packet.

4.5 PBKDF2

The keys used for authentication and encryption in LiteCrypto are expected to be 32 bytes long. For the users' convenience, we have included a specialized password-based key-derivation function for this library. The function takes in two user provided parameters: a variable-length key and a 16-byte salt. The function outputs a 32-byte key to the specified location and returns the size of the key on success. On failure, the key-derivation function returns -1.

The key-derivation scheme follows the standard PBKDF2 scheme from [15] with SHA512 as the underlying pseudo-random function. Since SHA512's digest size is 64 bytes [9] and our required key size is only 32 bytes, we only have one block under the PBKDF2 scheme. Because we always have only one block, we have omitted the block number from the initial iteration in evaluating U1 of the block. Excluding U1 which uses the user-provided salt, we perform ten thousand iterations with each iteration using the previous iteration's output as salt on the user-provided key [15]. All iterations are then bitwise XORed together to form the output key [15].

5 Evaluation and Analysis

5.1 Performance

Because performance is dependent on instruction optimization, the most efficient programs

are compiled specifically for a certain processor architecture. In order to reduce space, tweetNaCl cannot provide optimal run-time performance. [13] While TweetNaCl is not fully optimized for the processor, the impact on a low-volume cryptographic communication like the CubeSat is minimal.

5.1.1 -Os vs -O2 Optimization Flag

Most typical applications use the -O2 flags for improved performance, but on a space limited platform, the -Os flag is more useful. The -Os flag enables all -O2 flags that typically do not increase size as well as performing more optimizations to reduce size. [6]

5.2 Testing

In order to ensure the functionality of the library, we tested generated our own test cases. We were able to verify, encrypt, and decrypt messages successfully. Due to a lack of proper hardware, the program was not tested live on a CubeSat and with actual packets. The other option besides testing directly on a CubeSat would be to test it against a CubeSat virtual machine, but the communications for the CubeSat require additional libraries that we were not provided. In this situation, we would provide the library to the CubeSat team and have them perform functionality tests according to their needs.

5.3 Future Work

While the initial library is completed, there are many features that can be added with minimal impact to the size constraints.

5.3.1 Key Exchange by PKE

If a key pair needs to be updated for some reason, we currently require direct access to the CubeSat in order to replace and verify the key. Because we currently use only the initial key pair, if the key was compromised, the CubeSat would be vulnerable to a user with malicious intent. In order to mitigate such risks, a key exchange algorithm should be implemented to allow the update of keys used for communication.

5.3.2 Forward Secrecy

Forward Secrecy would allow for past communications to stay private even when an attack is successful in compromising the private key. Because all the past communications were protected by forward secrecy, the attacker would not be able to decrypt previous messages. This is crucial if the CubeSat is communicating confidential information.

5.3.3 Sequence Numbers

Sequence numbers would prevent replay attacks. Because communication to the CubeSat is not stable, packets are resent continuously while the CubeSat is in range. This increases the chance that the CubeSat has received the packet. With unstable communication, if the CubeSat did receive the packet, a response packet may not be received by the sender. This form of communication means that the process of repeating the message to the CubeSat can cause a replay attack by an official source. While replaying a command may not affect the mission, there is concern when it is crucial for the CubeSat to receive the command only once. Such commands include movement, a concept that may be implemented in CubeSat in the future. If movement is implemented in the CubeSat, the CubeSat cannot be allowed to replay a movement otherwise there is the possibility the CubeSat can be knocked out of orbit or into another satellite. The simplest way to mitigate this risk is through sequence numbers. Because the official source would issue

out one sequence number per command, a repeated packet would not affect the CubeSat once it parses the packet. As long as encryption remains secure, there is no concern over another entity forging a sequence number. This leads to the last case, a used sequence number with a different command. The only time this would happen through non-malicious means is if sequence number exceeds the amount of numbers allowed. In this situation, there should be some sort of authentication to alert the CubeSat that the sequence number should be reset. In the case of a sequence reset, measures should be taken so that an attacker collecting old packets cannot then send them to the CubeSat in attempts to get the CubeSat to perform a task.

6 Conclusion

The LiteCrypto library is available at `https://github.com/WardenGnaw/LiteCrypto`. It is available for anyone to use and modify. After further testing with CubeSat, LiteCrypto will hopefully be integrated for use.

From this this project, we have learned that we should not roll our own crypto, keep libraries small and simple, and the last important lesson is always to deal with it.

References

[1] "Authenticate ip protocol." [Online]. Available: https://crypto.stackexchange.com/questions/2414/is-this-authenticated-one-way-communication-protocol-secure

How to authenticate IP Packets

[2] "Ax.25 amateur packet-radio link-layer protocol." [Online]. Available: https://www.tapr.org/pub_ax25.html

Information on AX25

[3] "Cubesat." [Online]. Available: http://www.cubesat.org/

Information on CubeSat

[4] "Cyassl crypto library." [Online]. Available: http://www.yassl.com/yaSSL/Products-cyassl.html

CyaSSL Information

[5] "Nacl crypto library." [Online]. Available: http://nacl.cr.yp.to/

NaCl Crypto Library Information

[6] "Options that control optimization." [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

Different optimizations for GCC

[7] "Polysat." [Online]. Available: http://polysat.calpoly.edu/

Information on PolySat

[8] "A study of cryptography for satelite applications." [Online]. Available: http://www.slideshare.net/mannurajesh/a-study-of-cryptography-for-satellite-applications

Information on why stream ciphers cause issues for satelites.

[9] T. L. Daniel J. Bernstein and P. Schwabe, "Hashing: crypto_hash," August 2010. [Online]. Available: http://nacl.cr.yp.to/hash.html

NaCl library documentation on SHA-512

[10] ——, "Secret-key encryption: crypto stream," August 2010. [Online]. Available: http://nacl.cr.yp.to/stream.html

NaCl library documentation on xsalsa20 stream cipher

[11] ——, "Secret-key message authentication: crypto_auth," August 2010. [Online]. Available: http://nacl.cr.yp.to/auth.html

NaCl library documentation on HMACSHA512256

[12] ——, "Secret-key single-message authentication: crypto_onetimeauth," August 2010. [Online]. Available: http://nacl.cr.yp.to/onetimeauth.html

NaCl library documentation on one-time authentication

[13] W. J. T. L.-P. S. Daniel J. Bernstein, Bernard van Gastel and S. Smetsers, "Tweetnacl: a crypto library in 100 tweets," September 2014. [Online]. Available: http://tweetnacl.cr.yp.to/papers.html

Paper on TweetNaCl's construction

[14] J. S.-h. C. Z.-y. He Lei, Lu Xin-me, "Informatics in control, automation and robotics (car)," March 2010.

Paper for low cost one-way HMAC proposed for RFID system

[15] B. Kaliski, "Pkcs #5: Password-based cryptography specification," September 2000. [Online]. Available: https://www.ietf.org/rfc/rfc2898.txt

Republication of PKCS #5 v2.0 from RSA Laboratories' PublicKey Cryptography Standards (PKCS) series including PBKDF2

[16] D. Z. Li Ning; Lin Kanfeng, Lin Wenliang, "A joint encryption and error correction method used in satellite communications," March 2014.

This paper combines the AES(Advanced Encryption Standard) with LDPC(Low Density Parity Check Code) to design a secure and reliable error correction method -SEEC(Satellite Encryption and Error Correction).