Advanced Low Power Security (ALPS): A secure communication protocol for deeply embedded devices

Simon Pfeifhofer, Distributed Ledger GmbH
Stefan Huber, Distributed Ledger GmbH
David Gstir, sigma star gmbh

November 2018

Abstract

This document proposes a solution for a sustainable secure communication for LPWAN-devices summarized- and mentioned in this document as ALPS for Advanced Low Power Security. ALPS uses established cryptographic primitives and combines them according to a framework in order to help solution-architects and other decision-makers, which are facing the mentioned restrictions, requirements and associated challenges.

1 Introduction

The number of connected Internet of Things (IoT) is already above 20 billions and will increase at an even faster pace in the upcoming years [1]. A big part of the mentioned devices will be battery powered devices which offer a limited amount of resources, limited bandwidth and the requirement to transfer only a small amount of data from remote-locations using the OTA-interface without an edge-gateway.

The security aspect of such devices gets more and more important because critical processes depend on them. End-to-end security in combination with trusted platform modules (TPMs), trusted execution environments (TEEs) and secure elements (SEs) are necessary for business-models like pay per use based on sensor-data of such devices. Widely deployed solutions to ensure confidentiality, integrity, availability and authenticity with e.g. TLS-/DTLS-ciphers with PKIs based on x.509-certificates are currently considered as secure but they are not well suited for the mentioned low-resource devices, because the used algorithms- and handshaking-processes are too heavy weight in terms of computation and the number of roundtrips. They are difficult to implement/integrate in- and not primarily designed for such devices. Among other factors the workarounds

created in such situations (e.g. no transport security, vulnerable custom protocols) yields to serious security vulnerabilities in the IoT-world.

1.1 General assumptions

The general assumptions of the proposal in terms of energy-consumption are the following:

- The transmissions and receptions on the OTA-interface consumes the significant amount of energy.
- Given typical message-sizes (see Common characteristics) the count of transmitted-/received-messages is more significant for the energy-consumption than the length of each message. Handshake-only-messages need to be avoided or reduced to a minimum.
- The algorithms and their effective implementations (in hardware or software) which are necessary to preserve confidentiality, integrity, availability and authenticity can further increase the lifetime of a device.

2 Use cases

In this section lists common characteristics for use cases for ALPS and shows a reference use case to fill placeholders.

2.1 Common characteristics

ALPS targets tiny single purpose devices with one- or a few specific sensors/actors, which use LPWAN connectivity technologies like Sigfox [2], LoRaWAN [3] and NB-IoT [4]. A typical device lifetime will be ~5 years. The lifetime can be reached with a battery-capacity of ~4000 mAh. Mentioned characteristics imply devices with ~100 Mhz clock speed, ~2 MB persistent storage (flash) and ~100 KB of volatile memory (RAM). The persistent storage contains the firmware and can be used by the application itself to store persistent data. For critical use cases the devices are equipped with a trusted platform module (TPM), trusted execution environment (TEE) or secure element (SE) in order to store cryptographic-keys and execute cryptographic-operations. Every device has a unique identification (ID). The targeted devices runs connectivity- and application-specific firmware/software on a single MCU. The addressed use-cases requires a client-server-communication-model with only a few messages per day with small payloads (~100 bytes) which will be exchanged. The resource-optimization is crucial on client-side. On server-side it's acceptable to use more resources. On the transport-layer the payload to submit has to be divided in packets. It's acceptable to loose packets. Packets can arrive out-of-order.

2.2 Reference use case - DigitalBikeTwin

DigitalBikeTwin (DBT) is a special (e-)bike tracker-solution which requires a communication between the DBT-tracker and the DBT-cloud which ensures end-to-end confidentiality, integrity, availability and authenticity. The tracker is placed inside the frame and uses NB-IoT as cellular connectivity technology for communication and positioning [5]. The design-goal of the tracker is to reach a battery-powered-lifetime of 5-10 years in order to enable new business-models. The DBT-tracker always initiates a communication and occupies the client-role. The DBT-cloud occupies the server-role. The communication will be IP-based on the OSI network-layer. On the OSI transport-layer UDP will be used. The payload of every request/response will be 32-64 bytes. Only 2-8 request-response-pairs per day are expected. Quectels LTE BC66 NB-IoT Module [6] handles the connectivity and in addition runs the application-specific software which is supported by the OpenCPU feature. For the ID a hash of the International Mobile Equipment Identity (IMEI)[7] will be used. The key-material will be predeployed in the manufacturing-process.

3 The ALPS protocol

The main goal of ALPS is to specify a generic protocol that provides state-of-the-art security properties while maintaining a low energy footprint. For this purpose ALPS is based on Noise [8], which defines a framework of protocols that use state-of-the-art cryptographic algorithms and security properties like mutual authentication and forward secrecy as it is common in recent versions of TLS/DTLS.

Contrary to TLS however, Noise allows for various handshake patterns to be defined, which in the case of ALPS help to reduce the amount of send/receive operations on low power devices. ALPS selects the `Noise_KK_25519_ChaChaPoly_BLAKE2s` protocol variant which uses the following cryptographic algorithms best suited for the mentioned use cases:

- Curve25519 is used for Diffie-Hellman (DH) key exchange operations
- ChaCha20-Poly1305 is used as AEAD cipher
- BLAKE2s is used as hash function

The Noise `KK` pattern was selected because the setting for ALPS is that of a closed system where the server knows all its clients and the clients communicate with a single known server. This inherit trust between clients and server is achieved by the long-term static DH keypairs (consisting of a public and private key) of each participant, which have to be predeployed.

3.1 Long-term keys

The Noise `KK` patterns defines a long-term, *static* DH keypair for initiator and responder (client and server in ALPS case) of the protocol. These keypairs are generated in advance and predeployed on the corresponding devices.

In the case of ALPS this means that each client is provisioned with the server's static DH public key and its own static DH keypair. The server on the other hand knows the static public keys of all clients and its own static DH keypair.

3.2 Handshake

Whenever a client wants to send an application-level message to the server or vice-versa, a secure session is required, which holds the security parameters to encrypt and authenticate this message before sending it.

We keep to the [8] terminology and distinguish between **handshake messages** and **transport messages**. **Handshake messages** are exchanged to establish the security parameters for securing the **transport messages** which themself encapsulate the application-level payload. This means that devices initially exchange handshake messages followed by transport messages.

This session is established by a simple 1-RTT handshake pattern as defined by Noise `KK`:

1. The initiator (mostly the client) generates an ephemeral DH keypair.
2. The initiator executes a DH between its ephemeral key and the server's static public DH key (`es`) followed by its own private static key and the server's static public key (`ss`). This establishes the first usable encryption state.
3. The initiator sends the initial handshake message to the responder (ie. server)
4. The responder also generates an ephemeral DH keypair and executes a DH twice (`ee`, `se`) as defined in [8].
5. The responder sends its handshake response to complete the handshake

For full implementation details, see [8].

After the handshake is finished, the secure session is established and client and server can securely exchange messages using the key(s) of the session.

3.3 Zero-RTT messaging option

ALPS also enables zero-RTT messaging such that the initial handshake messages can already contain an application payload. This is especially useful to save energy on the client because the first (handshake) message of a session can contain application data and a entire roundtrip can be avoided.

While this option allows to save energy, it also has less strong security properties than messages sent after the full session was established. Since any payload data sent within the handshake depends on the long-term static keys of initiator and responder, a key compromise of these keys will compromise the security of those messages. The full details are documented in [8, Ch. 7.7].

It is up to the user of ALPS if Zero-RTT is used or not. ALPS suggests to use Zero-RTT in order to save energy.

3.4 Session handling

ALPS does support connectionless transport protocols like UDP and is intended to work on top of such. This does however add some drawbacks:

- Messages can be lost
- Messages might arrive out of order
- Higher denial-of-service risks

3.4.1 Session state handling

To handle out-of-order message delivery and message loss, sender and responder maintain three session states (the approach is basend on [9]):

- *Current session:* the currently used session state
- *Previous session:* previously used session
- *Next session:* future session

The current session holds the session which is currently used for decryption of received transport messages in the responder and for encrypting transport messages in the responder.

The previous session holds the session state, which was the current state before. This is needed to decrypt any old messages that were sent before the new session was established, but delivered to the responder *after* the handshake messages for the new session.

The next session is only maintained by the responder. It contains the next session which will be used. This session is filled whenever a new handshake is received and the response is sent. Since the response can be lost and never delivered to the initiator of the handshake, it might never be used. Thus this new session state is kept as next session until the first transport message using this session is received.

Whenever a transport message - using the next session state - is received, the session states are rotated and previous session is cleared (zeroed) and replaced by the value of current session. The current session is replaced by the value of next session. Next session is then also cleared. In case a new handshake

is received instead of a transport message using next session, next session is simply cleared (zeroed) and replaced with this new session. Current session and previous session remain unchanged.

Note that messages which cannot be decrypted using any of these three states are dropped. This means that messages that are delayed for a very long time before being delivered to the responder, will never be received by the application.

Creation of new sessions is always up to the initiator of the connection. This will support situations where the initiator loses its session state (e.g. through a power cycle). Situations where the responder loses its state are handled by the responder simply dropping messages since it has no active state. If that happens, the initiator will never receive a response and has to initiate a new session before sending any new messages. This also means that it is required for the responder to acknowledge transport messages. It's up to the initiator (within preset limits, see Rekeying) to decide how many messages without acknowledge are allowed to submit using the same session.

3.4.2 Explicit nonces

A message can be delivered out-of-order or even be lost, it is required to sync the nonce value (`n` in [8]) between sender and responder. This is done by sending the nonce as part of the transport and handshake message.

To prevent replay attacks of transport messages, the responder keeps track of used nonce values by using a sliding window. This sliding window is kept per session state and any message which reuses a nonce value for the same session is dropped.

An efficient implementation of the sliding can be found in [10]. In terms of the mentioned RFC ALPS suggest to use N=8 and M=2 which yields to a supported window size of (M-1)*N=8 which is appropriate for ALPS use cases.

3.4.3 Rekeying

ALPS is designed for IoT and sensor networks, therefore a session will be initiated by the client.

ALPS doesn't use rekeying as specified by [8, Ch. 11.3]. ALPS performs a new handshake in order to establish a new session. According to Noise, a single session has to consider the upper limit of a maximum of 2^64-1 transport messages and every transport message can be less than or equal to 65535 bytes in length. ALPS suggests a new handshake to be performed every 30 days of after 10 KB of transmitted data (whatever comes first). The frequency of the handshake is a tradeoff between power-consumption and session-duration/-volume with associated information leakage in case of a key-compromise.

On client- and server-side the re-handshake criteria are known. The negotiation of the criteria is not part of ALPS because it highly depends on the application. The client is responsible for the re-handshake. If the server detects a violation of the re-handshake it drops all messages until a new handshake is received.

3.5 Message format

As with Noise, the theoretical upper size limit for ALPS messages is 65535 bytes.

ALPS distinguishes between two different message types:

- Handshake
- Transport

Zero-RTT handshake messages are mentioned explicitly.

Table 1: Format of handshake-messages

field	length (byte)	encrypted	fixed-value
type	1	no	0x01
id	4	no	
nonce	8	no	
ephemeral public key	32	no	
timestamp	12	yes	
auth-tag	16	yes	

Table 2: Format of transport-messages

field	length (byte)	encrypted	fixed-value
type	1	no	0x02
id	4	no	
nonce	8	no	
ciphertext	0 - 65502	yes	
auth-tag	16	yes	

Table 3: Format zero-RTT-messages

field	length (byte)	encrypted	fixed-value
type	1	no	0x01
id	4	no	
nonce	8	no	
ephemeral public key	32	no	
timestamp	12	yes	

field	length (byte)	encrypted	fixed-value
ciphertext	0 - 65502	yes	
auth-tag	16	yes	

The field `type` is necessary to distinguish between handshake- and transport messages. The field `id` allows to identify the party (client or server) which transmits a message and lookup the predeployed long-term public static DH keys. The field `nonce` has to be part of any message because ALPS supports transports where messages will be lost or arrive out-of-order [8, Ch. 11.4]. The field `ephemeral public key` is part of the handshake-messages in order to allow the DH operations. As countermeasure for replay-attacks the field `timestamp` is part of any handshake message in order to avoid disruption of ongoing secure sessions [9, Ch. 5.1]. The field `auth-tag` is part of any message and delivered by- or fed into the ChaCha20-Poly1305 AEAD cipher.

4 Security properties

This chapter discusses the thread model and security properties of ALPS. Many will match those defined by [8].

4.1 Choice of algorithms

For the **DH function** we use X25519 [11] since it is the most appropriate selection for ALPS. The most suitable alternative is X448. Curve25519 and Curve448 are intended for different security levels: Curve25519 for ~128 bits. Curve448 for ~224 bits [12]. X448 might offer extra security in case a cryptanalytic attack is developed against elliptic curve cryptography. Software-implementations of X25519 are very likely to execute faster and consume less energy in comparison to X448 for ALPS use cases. X25519 is recommended by [8, Ch. 13] in combination with the later on selected hash function.

Chacha20-Poly1305 was selected as **symmetric cipher** in ALPS because there exists a considerable faster software implementations over AES-GCM [13]. Additionally, implementations of Poly1305 are quite straightforward and easy to get right [14]. ChaCha20-Poly1305 has recently seen more widespread adoption as alternative to AES-GCM in major protocols like TLS [15] and IPSec [16].

The BLAKE2s [17] [18] **hash function** is used by ALPS. BLAKE2s is a hash function which was first released 2012, has a high adoption and no known security issues.

It clearly outperforms the second candidate SHA256 [19]. ALPS favors BLAKE2s over BLAKE2b because BLAKE2b is optimized for 64-bit platforms

and BLAKE2s for smaller architectures. The ALPS use cases focus on 32- or even less bit architectures.

See the chapter future optimizations for future alternatives to the current choices.

4.2 Denial-of-service mitigation

In order to ensure availability requests to unknown IDs- and too frequent requests of a single address or (known) ID will be blocked. On client- and server-side the communication will be limited to two destination-endpoints and only requests from authenticity- and integrity-verified known IDs will be processed.

4.3 Information leakage

ALPS will prevent major information leakage by encrypting any application data. On the other hand, some metadata will be leaked. That is:

- ID
- Message size
- Client and server network addresses

The `ID` should be a hash of the underlying device identifier. It is possible for users of ALPS to add padding to each message to obscure the actual message size.

4.4 Collision attacks

Collision attacks as described by [20, pp. 33–35] exploit excessive reuse of the same key. *Birthday attacks* are based on the birthday paradox of probability theory and consider the likelihood of key reuse. Meet-in-the-middle attacks go a step further and an attacker precomputes the ciphertext under a (large) set of random keys for a single selected message and thus increases the likelihood to find a key reuse. This also requires the same plaintext to be transmitted.

For meet-in-the-middle attacks the attacker requires the plaintext to be sent to match the plaintext used for his precomputed lookup table. For ALPS use cases in sensor networks, this is actually easily achieved, if the attacker is able to influence the sensor measurements (e.g. position, temperature).

ALPS prevents these kinds of attacks by regular rekeying through new session states.

4.5 Replay protection

An attacker with the capability to capture network traffic between server and client and retransmit these packets can launch replay attacks on the protocol. Depending on the protocol payload this can eg. transmit false sensor measurements or simply outdated data.

ALPS prevents these kinds of attacks for handshake-messages with timestamps. For transport-messages the sliding-window and an explicit nonce are in place for prevention.

4.6 Forward secrecy

Key exposure is generally fatal for any security system. To mitigate it, ALPS provides *forward secrecy* by using ephemeral DH keys in the handshake to establish individual session keys. The session keys are expiring after a certain time has passed or certain amount of data was transmitted (whichever comes first).

Should an attacker be able to find the key of a single session, the security of the other sessions is still intact since their keys were chosen independently.

Note the security implications of using the Zero-RTT messaging option as for this case, long-term, static key compromise is worse than for regular transport messages.

5 Implementation notes

5.1 Key storage

The private keys of each static keypair have to be stored securely on the individual devices. The details on how this has to be achieved is up to the application. However, hardware security modules (HSMs) are commonly used for such tasks. Low-power devices often have hardware features that allow for secure storage. One example is the i.MX6 CAAM engine.

5.2 Random number generation

The protocol requires a random number generator (RNG) to generate ephemeral and static DH keypairs. Since static keypairs are required during the provisioning phase, these keys can be generated offline on specialized hardware with true random number generators (TRNG).

The ephemeral DH keypairs on the other hand have to be generated on each client and thus the client requires proper number generation. Common low power hardware today offers hardware-based RNGs that can be utilized for this purpose.

6 Future optimizations

This section lists future optimizations for ALPS.

6.1 Alternative cryptographic algorithms

The FourQ [21] DH function offers better performance and less energy-consumption across different architectures and without the consideration of specific hardware as Curve25519 [11]. There are efficient implementations available (e.g. FourQLib [22]) and special countermeasures for side-channel-attacks exist. For ALPS, which is a proposal for a sustainable solution which can be implemented at the time of writing, FourQ was not selected as primary DH function because it is quite new and not yet very widespread. In contrast, Curve25519, first released 2005, has withheld more research since then and has currently wider adoption. However FourQ is a promising alternative for Curve25519 that will allow future version of ALPS to further reduce energy consumption while maintaining a similar level of security.

Potential candidates for symmetric ciphers are the two finalists for use case 1 of the CAESAR challenge [23] ASCON [24] and ACORN [25]. Both are lightweight AEAD ciphers that are promising competitors for AES-GCM and ChaCha20-Poly1305.

References

[1] Gartner, "Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions)." 2018 [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. [Accessed: 04-Nov-2018]

[2] S. S.A., "Sigfox." [Online]. Available: https://www.sigfox.com. [Accessed: 17-Dec-2018]

[3] L. Alliance, "LoRaWAN." [Online]. Available: https://lora-alliance.org. [Accessed: 17-Dec-2018]

[4] 3GPP, "NB-iot, 3GPP releases 13 & 14." [Online]. Available: http://www.3gpp.org. [Accessed: 17-Dec-2018]

[5] F. G. Xingqin Lin Johan Bergman, "Positioning for the internet of things: A 3GPP perspective." [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1705/1705.04269.pdf. [Accessed: 27-Dec-2018]

[6] "LTE bc66 nb-iot module." [Online]. Available: https://www.quectel.com/product/bc66.htm. [Accessed: 27-Dec-2018]

[7] D. M. E. M. Montemurro Ed. A. Allen Blackberry, "7254: A uniform resource name namespace for the global system for mobile communications association (gsma) and the international mobile station equipment identity (imei)." 2014 [Online]. Available: https://tools.ietf.org/html/rfc7254. [Accessed: 16-Nov-2018]

[8] T. Perrin, "The noise protocol framework." [Online]. Available: http://noiseprotocol.org/. [Accessed: 05-Dec-2018]

[9] J. A. Donenfeld, "WireGuard: Next generation kernel network tunnel." 2018 [Online]. Available: https://www.wireguard.com/papers/wireguard.pdf. [Accessed: 27-Dec-2018]

[10] T. T. X. Zhang, "IPsec anti-replay algorithm without bit shifting." 2012 [Online]. Available: https://tools.ietf.org/html/rfc6479. [Accessed: 28-Dec-2018]

[11] S. T. A. Langley M. Hamburg, "Elliptic curves for security." 2016 [Online]. Available: https://tools.ietf.org/html/rfc7748. [Accessed: 06-Dec-2018]

[12] S. J. Y. Nir, "Curve25519 and curve448 for the internet key exchange protocol version 2 (ikev2) key agreement." 2016 [Online]. Available: https://tools.ietf.org/html/rfc8031. [Accessed: 29-Dec-2018]

[13] D. M. J. Salowey A. Choudhury, "AES galois counter mode (gcm) cipher suites for tls." 2008 [Online]. Available: https://tools.ietf.org/html/rfc5288. [Accessed: 20-Dec-2018]

[14] A. L. Y. Nir, "ChaCha20 and poly1305 for ietf protocols." 2015 [Online]. Available: https://tools.ietf.org/html/rfc7539. [Accessed: 20-Dec-2018]

[15] N. M. A. Langley W. Chang, "ChaCha20-poly1305 cipher suites for transport layer security (tls)." 2016 [Online]. Available: https://tools.ietf.org/html/rfc7905. [Accessed: 27-Dec-2018]

[16] Y. Nir, "ChaCha20, poly1305, and their use in the internet key exchange protocol (ike) and ipsec." 2015 [Online]. Available: https://tools.ietf.org/html/rfc7634. [Accessed: 27-Dec-2018]

[17] J.-P. A. M-J. Saarinen, "The blake2 cryptographic hash and message authentication code (mac)." 2015 [Online]. Available: https://tools.ietf.org/html/rfc7693. [Accessed: 21-Dec-2018]

[18] Z. W.-O. Jean-Philippe Aumasson Samuel Neves, "BLAKE2: Simpler, smaller, fast as md5." 2013 [Online]. Available: https://blake2.net/blake2_20130129.pdf. [Accessed: 21-Dec-2018]

[19] V. -Virtual Applications and I. R. Lab, "EBACS: ECRYPT benchmarking of cryptographic systems." [Online]. Available: https://bench.cr.yp.to/results-hash.html. [Accessed: 21-Dec-2018]

[20] T. K. Niels Ferguson Bruce Schneier, *Cryptography engineering*. Wiley Publishing, Inc., 2015 [Online]. Available: https://doi.org/10.1002/9781118722367

[21] "FourQ on embedded devices with strong countermeasures against side-channel attacks." [Online]. Available: https://eprint.iacr.org/2017/434.pdf. [Accessed: 19-Dec-2018]

[22] "FourQlib v3.0 (c edition)." [Online]. Available: https://github.com/Microsoft/FourQlib. [Accessed: 19-Dec-2018]

[23] D. J. Bernstein, "Announcement of the caesar finalists." 2012 [Online]. Available: https://cr.yp.to/talks/2018.03.05/slides-djb-20180305-caesar-4x3.pdf. [Accessed: 21-Dec-2018]

[24] F. M. Christoph Dobraunig Maria Eichlseder, "ASCON - a family of authenticated encryption algorithms." 2016 [Online]. Available: https://ascon.iaik.tugraz.at. [Accessed: 21-Dec-2018]

[25] H. Wu, "ACORN: A lightweight authenticated cipher (v3)." 2016 [Online]. Available: https://competitions.cr.yp.to/round3/acornv3.pdf. [Accessed: 21-Dec-2018]