Cubedate: Securing Software Updates in Orbit for Low-Power Payloads Hosted on CubeSats

Authors
Something, France. Email: first.lastname@something.fr

***Abstract*—CubeSat design is facilitated by the increasing availability of open source software in the domain, and a variety of low-cost hardware blueprints based on commodity microncontrollers. We attain the rock-bottom price to reach orbit as entities which design, launch and operate CubeSats started selling to multiple tenants tiny rack slots (typically 0,25U each) for low-power payloads which may be hosted on their CubeSat. The question arises of how to provide state-of-the-art security for software updates on a multi-tenant CubeSat, whereby mutual trust between tenants is limited. In this paper, we provide a case-study: ThingSat, a low-power payload we designed, currently hosted on a CubeSat orbiting at 500km altitude operated by a separate entity. We then design Cubedate, a framework for securing continuous deployment of software to be updated on orbiting multi-tenant CubeSats. We also provide a highly portable open source implementation of Cubedate, based on the IoT operating system RIOT, which we evaluate experimentally.**

I. Introduction

Driven in parts by the increasing involvement from public research and the academic community [1], more and more open source software and hardware are available, maturing and used on CubeSats.

Their availability has lowered the bar of entry, with the aim of designing low-cost CubeSats which can more reliably achieve a successful launch – until recently, the failure rate was still around 60% for first-time satellite builders [2].

With the same objective, another trend has been to buy tiny "rack-space" in orbit [3]. With this model, a stakeholder (a user) can place a small (<1U) payload slot hosted inside a CubeSat provided by an operator (a different stakeholder). The user only needs to design and operate this payload, instead of designing and operating the whole CubeSat, drastically reducing users' costs and lead time. Typically, such a payload boils down to a printed circuit board (PCB) with sensors, a low-power CPU (sometimes with separate wireless communication capabilities) and a bus communication interface to the CubeSat main on-board flight computer (OBC). As such, a hosted payload on a CubeSat resembles small embedded devices usually found in the Internet of Things (IoT).

Nevertheless, even the entry bar lowered through the combination of a hosted payload and leveraging open source, software embarked on launched CubeSats tend to be minimalist (think: space rush, time pressure to meet hard launch deadlines...) and buggy, as most software is. In effect, hosted CubeSat payload software must typically be updated over-the-air (OTA) regularly, after it is deployed in orbit and in operation. Beyond enabling (OTA) software updates on a hosted CubSat payload, cybersecurity is a crucial emerging aspect. Indeed, on one hand software updates can be used to fix vulnerabilities but on the other hand tampering with software updates can be used as a cyberattack vector [4].

State-of-the-art has so far focused on single-stakeholder CubeSats use cases, where the software embarked on the CubeSat is managed by a single entity. In this paper, we instead focus on the challenge of securing software updates on low-cost multi-tenant CubeSats, whereby the OBC and the payload are operated by different stakeholders which do not necessarily trust each other. In particular, we focus on payloads based on low-power microcontrollers, which are essential on low-cost CubeSats, where low power consumption is key.

II. Related Work

Open source hardware and software are increasingly used on low-cost CubeSats, including on microcontroller parts. For instance pyCubed [2] proposes open source CubeSat hardware based on an Arm Cortex-M4 (atsamd51) microcontroller and corresponding open source software based on Python (Adafruit CircuitPython). OreSat [5] designs microcontroller based CubeSat, and provides corresponding open source software. Organizations such as the Libre Space Foundation [6] harbor a number of open source software code basis for CubeSats, such as UPSat, Qubik.

Once the CubeSat is in orbit, during its lifetime, the software it embarks must typically be updated over-the-air. Some work such as [7] have focused on the reliability of the ground-flight link for firmware updates. Other work have focused on mitigating radiation effects corrupting firmware storage and on error correction, such as [8] or partly in [9]. A vanilla approach to securing low-cost CubeSat software updates, such as described in [10], only uses weak security primitives (e.g., MD5 integrity checks) instead of strong primitives such as authentication with digital signatures and encryption. However, recent cyberattacks such as the ViaSat outage in Ukraine [11] suggest that, as CubeSat systems become more popular, they are likely to also become more prone to cyberattacks. A crucial related challenge is thus **how to provide strong security for in orbit software updates**. Prior work such as NUTS [12], [13] have focused on authentication and communication security over the ground-flight uplink.

To the best of our knowledge, however, no prior work exists on providing strong security for software updates on a low-power CubeSat *payload*, via a satellite uplink and an OBC that are *both potentially untrusted*. This hosted CubeSat use-case is depicted in Figure 1.

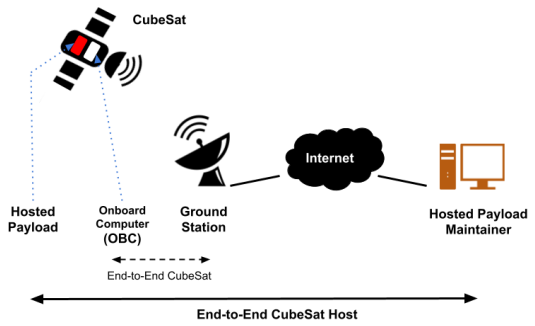

Fig. 1. CubeSat hosted payload software update security end-to-end.

Paper contributions

The main contributions of this paper are the following

- We provide a case-study: ThingSat, a low-power, low-cost payload hosted on a CubeSat, currently in-orbit;
- We analyze its software update requirements and constraints;
- We define Cubedate, a generic architecture enabling standards-based, secure software updates for payloads hosted on a low-power CubeSat;
- We provide and evaluate an open source implementation of Cubedate.

III. Case Study: ThingSat

The ThingSat project [14] aims to benchmark ground-space LoRa links on different frequency bands and demonstrate the effectiveness of that technology inside a LEO (Low Earth Orbit) CubeSat.

ThingSat is deployed as a hosted payload on a shared 3U CubeSat: STORK-1 from the polish start-up SatRevolution. The CubeSat was launched on January 13th, 2022,currently in orbit at an altitude of 525km (see its Two-Line Elements).

The ThingSat payload now in orbit is used for Ocean level monitoring. However, its design can be adequate for a wider variety of use cases, in Earth science academic research (e.g. tacking the melting of glaciers, pirate fishing...) and in the industry for companies using geographically dispersed devices (e.g. monitoring of tank ships...).

A. Distributed System Architecture

Figure 2 describes the ThingSat deployment components, it gives an overview of a typical CubeSat ecosystem, whereby the interaction with this payload traverses untrusted elements.

Low-power Space Segment

The Space Segment comprises the on-board computer (OBC) and hosted payloads, whom, interconnected via a CAN bus, which share resources on the CubeSat.

- The OBC provided by the satellite operator consists on a microcontroller with all its subsystems to operate the CubeSat: Attitude Determination and Control System (ADCS), communication subsystems (UHF/VHF/S-band for uplink/downlink and antennas) and power subsystem (Battery Management, Energy Harvesting with Solar Panels, Auxiliary Power Supply).
- We designed the ThingSat payload, using a STM32F405RG microcontroller featuring an ARM Cortex-M4 core and open source firmware based on RIOT [15]. The ThingSat payload embeds both a Semtech SX1302 transceiver for communications on the 863-870MHz band and a Semtech SX1280 transceiver for communications on the 2400-2500MHz band. Furthermore we designed a corresponding dual-band patch antenna (868MHz, 2.4GHz). When active and using the 863-870MHz band, the ThingSat payload consumes at 3.3V: (i) 90mA in standby, (ii) 110mA during a frame reception (RX) and (iii) 300mA during a frame transmission (TX) at 27dBm.

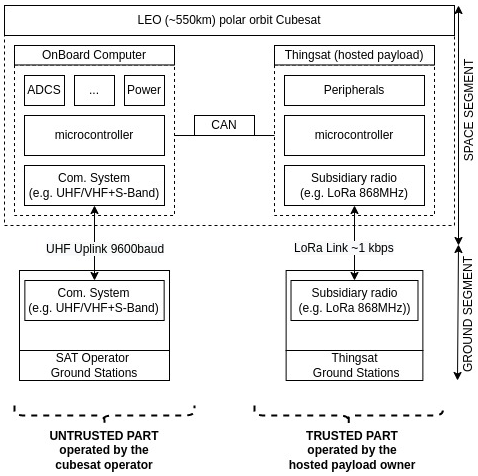

Fig. 2. ThingSat hosted payload: deployed components and architecture.

Ground Segments

Ground Segment elements that communicate with the ThingSat payload are:

- **SatRevolution Ground Stations**: provided by the CubeSat operator [1] to communicate via UHF/VHF with the OBC, and indirectly with the payload. This can be done directly od through a Command & Control Center, which acts as a broker between payload maintainers and hosted payload (indirect access).
- **ThingSat LoRa Ground Stations**: that we designed, deployed and maintain, which can communicate via LoRa **directly** with the ThingSat payload. These stations are based on an ESP32 microcontroller, and a 2.4GHz SX1280-LoRa transceiver, also running an open source firmware based on RIOT.

B. Communication Characteristics Overview

ThingSat payload communicates either directly via low-power WAN, or indirectly via the UHF/VHF link provided by the CubeSat's OBC.

[1] not necessarily owned by the CubeSat operator

Direct communication patterns via low-power WAN: ThingSat can communicate directly with LoRa. In principle, although it is not used as such so far, this communication link could also be used to transport software updates. As shown in Figure 3 the ThingSat payload may act as either: (i) a Sat-IoT end-device (ED) that will send LoRa frames to terrestrial LoRaWAN gateways or ThingSat ground stations (GS), or (ii) an in-orbit LoRa sniffer, or (iii) a store-carry-and-forward LoRa gateway.

Patterns (i) and (ii) allow to benchmark simple ground-space LoRa links by computing statistics over multiple sent/received frames. Pattern (iii) is a more complex scenario: the satellite stores packets received from GS/ED and deliver them once GS/ED destinations are inside the footprint of the satellite.

Indirect communication characteristics via UHF/VHF: CubeSat-GS communications are typically done on amateur frequency bands (UHF/VHF) with typically low data rates ranging from 9.6Kbps to 100Kbps. A polar LEO satellite will typically pass over a given ground station 2 to 4 times/day, each pass having a communication window of to 10 minutes.

For ThingSat, the CubeSat Operator provides only 2 ground stations (both in Europe), communicating with the CubeSat via a 10-Kbps UHF/VHF link. Thus, the daily throughput is roughly 1500KB (corresponding to 2GS x 2 passes/day x 5-min pass duration x 10Kbps). However, this throughput must be shared between communications to/from the OBC (for telecommand/telemetry/update) and to/from hosted payloads. Therefore in practice, the total communication budget available for ThingSat via the UHF/VHF link is around 300KB/day.

Intermittent communication/power supply: Last but not least, the ThingSat payload is not constantly powered on. Typically, at point any time, only a single hosted payload is powered on. For a 3U, 1U is dedicated to the OBC and the remaining 2U, available for hosted payloads (8 payloads slots of 0.25U in the case of ThingSat). Therefore, on average ThingSat is powered only 1/8th (12.5%) of the time (modulo other factors such as mission specificities, regulations, battery level etc.).

C. Hosted Payload Software Updates Requirements

Data exchanges between the Payload Maintainer and ThingSat consist on **downlinks**: used by ThingSat to send mission results (radio metadata, frame stats, collected LoRa frames) and diagnosis data (debug info on failed missions/updates) **uplinks**: used for software updates of two categories:

1) Firmware updates: to fix bugs, add/improve functionality (typically ∼200KB per FW, 1 FW/month);
2) Mission updates: to configure scenarios (typically ∼700B per mission scenario, 1 scenario/day);

IV. CUBEDATE: STANDARD SECURITY FOR CONTINUOUS DEPLOYMENT OF LOW-POWER CUBESAT SOFTWARE

In this section we describe Cubedate, a structured approach to securing heterogeneous software updates for in-orbit CubeSat software.

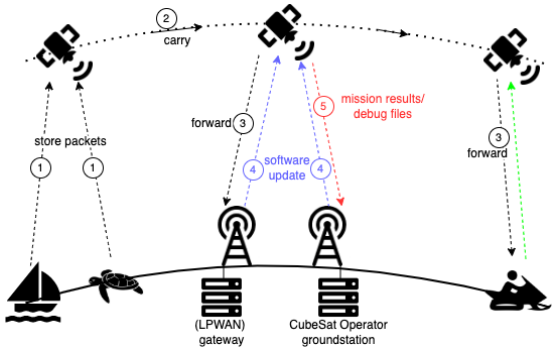

Fig. 3. ThingSat in-orbit communication patterns.

A. Security Requirements

The minimal security guarantees that we aim for with Cubedate are authenticity and integrity of software updates delivered over the network, during the lifetime of the satellite mission (5-10 years). Cubedate must allow for crypto agility, i.e. update the crypto primitives used to secure update to the satellite while in operation. This need can be dictated either by cryptography's evolving state-of-the-art (implementation/algorithm vulnerabilities are discovered) or by the need to transfer the trust anchor to a new entity (the authorized maintainer has changed). Additional guarantees beyond authenticity/integrity should also be possible with Cubedate, such as confidentiality, software update replay attacks, or software update mismatch attacks.

B. Trust Anchor

Our model is based on a single trust anchor: the authorized maintainer for the CubeSat hosted payload. There is no mitigation if this trust anchor used is compromised. We thus rely on the maintainers' ability to keep their private keys secure. Extensions using a (hierarchical) public key infrastructure are possible but out of scope for this paper.

C. Cubedate Software Life-Cycle Phases

The basic process we use for securing authenticity and integrity of software updates is decomposed in six phases shown Figure 4. During a preliminary, pre-flight phase (*Phase 0*) the authorized maintainer for the CubeSat-hosted payload produces and flashes the payload with commissioning material: a bootloader, the initial firmware, and authorized crypto material (a public key, and a cryptographic hash function). Once the hosted payload is commissioned it can be sent to the CubeSat operator of installation in the CubeSat.

Once the CubeSat is in orbit, the hosted payload maintainer can trigger iterations through cycles of Phases 1-5, whereby the authorized maintainer can build a new software update (*Phase 1*), hash the update and sign the hash (*Phase 2*) then push a network transfer (PUT) towards the hosted payload via the ground station and the OBC (*Phase 3.1*). The next time it wakes up, the hosted payload can then ping and fetch (GET) the update from the OBC (*Phase 3.2*), proceed to verify the signature and the hash (*Phase 4*), and upon successful

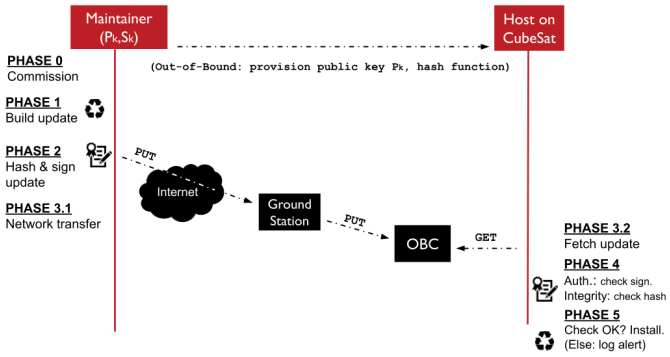

Fig. 4. CubeSat hosted payload secure software update process.

verification, install/boot the new software (*Phase 5*), otherwise the update is dropped.

D. Supporting Network Transport Heterogeneity

This aspect concerns *Phase 3.1* and *3.2* in Figure 4. Security guarantees on software updates must remain valid ënd-to-end.̈ Depending on the use-case, ënd-to-end̈spans differently, as depicted for example in Figure 1. In the most complex case we tackle in this paper, end-to-end means all the way from the hosted payload software maintainer to the payload hosted in orbit on the CubeSat.

Software updates may be transported over one or more network links of varying nature:

- **Developer to Groundstation**: Internet;
- **Groundstation to CubeSat**: UHF/VHF, LoRa...
- **Intra-CubeSat**: CAN, I2C, RS-232...

Intermittent power supply, combined with orbiting and radio range limitations impacts the intermittence of network connectivity to/from the hosted payload: establishing a delay-tolerant path and in-network data caching might be required. To cope with this wide variety of network paths and links (including ultra-constrained low-power elements), different approaches can be envisioned at the network layer, the transport layer and the application layer. Approaches span from proprietary solutions to standards such as the low-power IPv6 protocol stack (6LoWPAN, UDP, CoAP [16]) or experimental stacks such as information-centric networking which benefits from in-network caching even with small caches on microcontrollers [17].

Nevertheless, in order to retain generality, Cubedate does not specify any particular approach at the network, transport and application layers to enable the delivery of software updates across the network. Cubedate only aims to guarantee end-to-end security properties for the software update binaries that are delivered, somehow, over the network.

E. Supporting Updated Software Heterogeneity

This aspect concerns both *Phase 1* and *Phase 5* in Figure 4. As seen in subsection III-C, software updates may be of various nature and size. Cubedate aims to support the same mechanism, workflow and guarantee to update the CubeSat (1) firmware updates, (2) mission scenario files and (3) runtime configuration files.

For this reason, we choose not to rely on specialized approaches such as DFU (Device Firmware Update [18]) which assumes that the software is firmware and that the device is connected directly via some local bus connection (e.g. USB).

Instead, we aim to combine the use of generic and standard metadata characterizing software updates and state-of-the-art cryptographic primitives applicable on most low-power microcontrollers and a large variety of low-power networks, as described below.

F. Low-power End-to-End Security using SUIT

Cubedate leverages the SUIT manifest [19] which specifies a metadata format to describe software updates. This format uses Concise Binary Object Representation (CBOR) for data serialization, and a security wrapper which protects the metadata end-to-end, leveraging the CBOR Object Signing and Encryption (COSE) specification - all of which are IETF open standards for low-power communication security [20].

The Cubedate software update binary itself can be either encapsulated in the SUIT manifest, or transferred separately based on the URI provided in the manifest. For instance, the metadata includes a sequence number (preventing unwanted rollbacks), the expected device type (preventing software mismatch), the SHA256 digest of the software update binary and of the manifest, and the ed25519 digital signature of the manifest (the metadata). As such, using Cubedate, software updates for payload hosted on CubeSats mitigate attacks including:

- **Tampered Software Update Attacks –** An attacker may try to update the IoT device with a modified and intentionally flawed software image. To counter this threat, Cubedate uses digital signatures on a hash of the image binary and the metadata to ensure integrity of both the firmware and its metadata.
- **Unauthorized Software Update Attacks –** An unauthorized party may attempt to update the IoT device with modified image. Using digital signatures and public key cryptography, Cubedate ensures that only the authorized maintainer (holding the authorized private key) will be able to update the device.

G. Supporting Crypto Agility

The first level of crypto agility enabled by Cubedate uses flexibility provided by the SUIT standard specification: while keeping the same metadata and workflow, diverse crypto primitives backends can be used. For instance, to upgrade from pre- to post-quantum security, digital signature performed with ed25519 (elliptic curve crypto), can be swapped for hash-based signatures (LMS [21]).

The second level of crypto agility enabled by Cubedate leverages a dedicated embedded runtime architecture: on the

one hand, we place the software update manager (implementing SUIT-related operations) in the firmware image itself. On the other hand, we perform cryptographic operations in software only.

Thus, changing the trust anchor stored is as simple as swapping a public key in the next firmware's update manager. Authorization to update the firmware can thus be easily delegated to another maintainer, who can take over the production and the roll out of authorized updates. Furthermore, the update manager in the next firmware image could implement and use upgraded cryptographic primitives.

H. Guarantees beyond Authenticity/Integrity

Cubedate may also guarantee confidentiality by encrypting (optional) software updates transmitted over the network. It is performed using the encrypt/decrypt mechanism provided by the SUIT specifications [22], using a symmetric cryptographic key commissioned in the update manager by the authorized maintainer. Confidentiality can mitigate additional cyberattacks leveraging analysis of CubeSat firmware/software binaries.

Going beyond authenticity, integrity and confidentiality guarantees for software updates delivered over the network, using Cubedate also mitigates other attacks including:

- **Software Update Replay Attacks –** An attacker may try to replay a valid, but old (known-to-be-flawed) software. This threat is mitigated by using a sequence number. Cubedate uses a sequence number, which is increased with every new software update.
- **Software Update Mismatch Attacks –** An attacker may try replaying a software update that is authentic, but for an incompatible device. Cubedate includes device-specific conditions, which can be verified before installing a software binary, thereby preventing the device from attempting to use an incompatible software image.

V. Cubedate Implementation

We implemented Cubedate by extending the open source ecosystem around RIOT, which we combined with our initial firmware running on the ThingSat payload, also open sourced, which we published and maintain at []git:cubedate-repo/

Communication Bus: To provide a simple socket-like API on top of the raw CAN bus connecting hosted payloads and OBC on the CubeSat, we used libCSP, a library implementing the CubeSat Space Protocol (CSP version 1), which we integrated into RIOT. This stack is deployed on Satellites such as Nuts[12], GomSpace and used by SatRevolution, as it was a requirement for communicating with the OBC. Roughly, CSP provides an ëquivalentöf the IP/UDP stack and static routing.

SUIT Implementation: To fully support the wide variety of software updates case required by Cubedate, we extended existing RIOT implementation of SUIT [23] which, initially, supported only firmware updates, internal storage, and WPAN network delivery. We generalized the SUIT state machine to add support for:

- **Heterogeneous update delivery mechanisms**: configurable {message model, network stack, network interface} bundle, or FileSystem *read/write* functions.
- **Heterogeneous storage destination**: configurable internal/external memory: Volatile (e.g. RAM for mission files, or for tiny runtime execution containers such as FemtoContainers [24])or non-volatile (e.g. FileSystem or internal Flash).
- **Heterogeneous update data URI**: either a local file (e.g.: mounted USB device) or a remotely accessible file (eg CoAP or HTTP endpoint);

Network Pull Implementations: We implemented several alternatives for the client/server interaction which must be initiated by the hosted payload to pull (GET) software update data over the network from either the OBC, or a (LPWAN) groundstation.

The first pull mechanism we implemented uses CoAP [25], an open standard low-power IPv6 protocol following a general-purpose REST architecture. In particular, it is not only applicable over the CAN/CSP link, but also usable over VHF links [26] or LoRa links [27], which paves the way for some Cubedate software updates over the direct link from a ThingSat LoRa groundstation to the ThingSat payload.

However, Cubedate is agnostic w.r.t the pull mechanism. To demonstrate this, we also implemented an alternative pull mechanism using UP (the Universal Platform protocol), a dedicated proprietary protocol provided by the CubeSat operator for request/response interaction with the OBC. This option is currently deployed and running on ThingSat in orbit.

VI. Cubedate Evaluation and Discussion

In the following, code measurements where generated compiling with ARM GCC 10.2.1, optimized for code size. As code base, we used RIOT release 2022.01 and SUIT configured with ed25519 digital signatures provided by the C25519 crypto library as backend (which has small footprint as shown in prior work [23]).

1) Memory Footprint Overhead: To evaluate the RAM and Flash footprint of our Cubedate implementation, we apply it to the ThingSat use-case, compiled for the hosted payload hardware described in subsection III-A (based on a Cortex-M microcontroller).

In Table I we compare the RAM and Flash memory requirements for a ThingSat firmware without/with Cubedate-compliant secure software updates. We observe that Cubedate requires a memory budget of $\sim$4KB of RAM and $\sim$19KB of Flash, which represents roughly a 10% increase in the total RAM and Flash memory budget for ThingSat.

2) Network Transfer Overhead: On the UHF/VHF link at 1kb/s, the additional network transfer time induced by the Cubedate firmware size overhead (19KB) is roughly 15 seconds. This overhead is reasonable, but not negligible keeping in mind that a connection to the CubeSat is segmented in time windows of $\sim$300 seconds.

Next, in Table I we look in more details at the metadata (SUIT manifest) used to secure Cubedate software updates

	ThingSat		ThingSat+Cubedate	
	RAM	Flash	RAM	Flash
CAN	10774	8762	10774	8762
Crypto	633	7386	64	13760
CoAP	1024	2192	1024	1632
CSP	8541	11771	8541	12653
SUIT	0	0	3200	7425
LoRa GW	22300	45688	22300	45688
Firmware	7008	108881	8225	114088
Total	**50280**	**184680**	**54128**	**204008**

TABLE I
CUBEDATE IMPLEMENTATION: MEMORY FOOTPRINT IN BYTES.

for ThingSat. As we can see, the metadata including all CBOR/COSE formatting, digests (SHA256 hashes) and authentication data (ed25519 signature) amounts to ∼330B. The metadata overhead thus incurs negligible overhead (+0,15%) in case of a ThingSat firmware update (of size 200KB on average, recall subsection III-C). However, in case of a smaller software update such as updating a mission scenario (average size 700B) the overhead of adding Cubedate metadata is significant (almost +50%). Nevertheless, on the UHF/VHF link at 1kb/s, this overhead remains negligible in terms of additional network transfer time.

A. Discussion & Perspectives

1) Portability: Bolted on top of RIOT, our Cubedate implementation works out-of-the-box (or is trivially portable) on a very wide variety of low-powr hardware built on Cortex-M microcontrollers: the bulk of the 200+ boards supported by RIOT. However, additional work would be needed to support other hardware based on diffrent 32-bit microcontroller architectures (one thinks of RISC-V).

2) Network Stack Simplification & Standardization: The use of CSP was mandated by the cubesat operator. The purpose of CSP was to provide an ultra-low footprint equivalent of the IP protocol stack for cubesats with legacy (8-bit) microcontrollers. However, on modern (32-bit) microcontrollers such as those used in ThingSat, this approach is questionable. An alternative to CSP based on more widely spread standards seems possible for the same ”price”. For example, RIOT’s default low-power IPv6 (6LoWPAN) stack used with static routing has a footprint in RAM/Flash memory that is comparable (or smaller) than libCSP memory footprint. The 6LoWPAN stack could run direcly on the CAN bus or on LoRa (see 6loCAN [28] and SCHC [29]).

3) Alternative cryptographic primitives: In our experimental evaluation, we used ed25519 (elliptic curve cryptography) digital signatures providing 128-bit pre-quantum security. One can consider either alternative primitives, while remaining compliant with Cubedate and the SUIT standard. One compromise to significantly decrease network transfer and memory footprint with Cubedate is to use HMAC (symmetric crypto) instead of digital signatures for authentication. Another option would be to upgrade security to 128-bit post-quantum security by using hash-based signature instead of ed25519.

VII. NEXT STEPS & FUTURE WORK

RIOT/SUIT differential firmware updates:
Business logic containerization:

VIII. CONCLUSION

ACKNOWLEDGMENT

REFERENCES

[1] NASA. CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers. https://www.nasa.gov/sites/default/files/atoms/files/nasa_csli_cubesat_101_508.pdf.

[2] M. Holliday, A. Ramírez, C. Settle, T. Tatum, D. G. Senesky, and Z. Manchester, “Pycubed: An open-source, radiation-tested cubesat platform programmable entirely in python,” 2019.

[3] SatRevolution. CubeSat STORK Mission. https://satsearch.co/products/satrevolution-in-orbit-demonstration-iod.

[4] A. Greenberg, “Software has a Serious Supply-Chain Security Problem,” *Wired*, Sep 2017, https://www.wired.com/story/ccleaner-malware-supply-chain-software-security.

[5] C. Spivey and E. Gizzi, *A Modular, Open Source CubeSat Structure*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2021-1256

[6] LibreSpaceFoundation. Open Source Projects. https://libre.space/.

[7] S. Fitzsimmons, “Reliable software updates for on-orbit cubesat satellites,” Master’s thesis, CalPoly, 2012.

[8] B. Yuen and M. Sima, “Low cost radiation hardened software and hardware implementation for cubesats,” *arXiv preprint arXiv:1902.04117*, 2019.

[9] I. Sünter *et al.*, “Firmware Updating Systems for Nanosatellites,” *IEEE Aerospace and Electronic Systems Magazine*, 05 2016.

[10] M. I. Monowar and M. Cho, “Over-the-air firmware update for an educational cubesat project,” *International Review of Aerospace Engineering (IREASE)*, vol. 14, no. 1, 2021.

[11] Reuters. Satellite Outage Caused Huge Loss in Communications at War’s Outset. https://www.reuters.com/world/satellite-outage-caused-huge-loss-communications-wars-outset-ukrainian-official-2022

[12] R. Birkeland and O. Gutteberg, “Overview of the nuts cubesat project,” 02 2014.

[13] B. Bezem *et al.*, “Authenticated uplink for the small, low-orbit student satellite NUTS,” 2013.

[14] “Thingsat public gitlab repository,” https://gricad-gitlab.univ-grenoble-alpes.fr/thingsat/public.

[15] E. Baccelli *et al.*, “RIOT: An Open Source Operating System for Low-end Embedded Devices in the IoT,” *IEEE Internet of Things Journal*, 2018.

[16] R. Morabito and J. Jiménez, “Ietf protocol suite for the internet of things: Overview and recent advancements,” *IEEE Communications Standards Magazine*, vol. 4, no. 2, pp. 41–49, 2020.

[17] O. Hahm *et al.*, “Low-power Internet of Things with NDN & cooperative caching,” in *Proceedings of the 4th ACM Conference on Information-Centric Networking*, 2017, pp. 98–108.

[18] J. Beningo, “Update firmware in the field using a microcontrollers DFU mode,” https://www.digikey.com/en/articles/update-firmware-field-using-microcontroller-dfu-mode, 2018.

[19] B. Moran *et al.*, “CBOR-based Serialization Format for the SUIT Manifest,” *IETF Internet Draft draft-ietf-suit-manifest*, Oct 2021.

[20] H. Tschofenig and E. Baccelli, “Cyberphysical security for the masses: A survey of the internet protocol suite for internet of things security,” *IEEE Security & Privacy*, vol. 17, no. 5, pp. 47–57, 2019.

[21] G. Banegas *et al.*, “Quantum-Resistant Security for Software Updates on Low-power Networked Embedded Devices,” *to appear in International Conference on Applied Cryptography and Network Security*, Jun 2022.

[22] H. Tschofenig *et al.*, “Firmware Encryption with SUIT Manifests,” Tech. Rep. draft-ietf-suit-firmware-encryption-03, Mar. 2022.

[23] K. Zandberg *et al.*, “Secure firmware updates for constrained IoT devices using open standards: A reality check,” *IEEE Access*, 2019.

[24] K. Zandberg and E. Baccelli, “Femto-containers: Devops on microcontrollers with lightweight virtualization & isolation for iot software modules,” *arXiv preprint arXiv:2106.12553*, 2021.

[25] Z. Shelby *et al.*, “The Constrained Application Protocol (CoAP),” Jun. 2014.

[26] D. Palma, "Enabling the maritime internet of things: Coap and 6lowpan performance over vhf links," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5205–5212, 2018.
[27] R. Sanchez-Iborra *et al.*, "IPv6 communications over LoRa for future IoV services," in *IEEE World Forum on Internet of Things (WF-IoT)*, 2018.
[28] A. Wachter, "IPv6 over Controller Area Network," Internet Engineering Task Force, Internet-Draft draft-wachter-6lo-can-01, Feb. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-wachter-6lo-can-01
[29] A. Minaburo *et al.*, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation," RFC 8724, Apr. 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8724