Towards Formal Security Verification of Over-the-Air Update Protocol: Requirements, Survey and UpKit Case Study

Christophe Ponsard and Denis Darquennes
CETIC Research Centre, Charleroi, Belgium

Keywords: Cybersecurity, Formal Modelling, Verification, Update Protocol, Internet of Things.

Abstract: The fast growing number of connected devices through the Internet of Things requires the capability of performing secure and efficient over-the-air updates in order to manage the deployment of innovative features and to correct security issues. Pushing an updated image in a device requires a complex protocol exposed to security threats which could be exploited to block, spy or even take control of the updated device. Hence, such update protocols need to be carefully designed and verified. In the scope of this paper, we review some representative update protocols and related threats based on MQTT, TUF (Uptane) and the blockchain. We then show how the adequate management of those threats can be verified using a formal modelling and verification approach using the Tamarin tooling. Our work is applied to the concrete case of the UpKit protocol which exhibits an interesting design.

1 INTRODUCTION

The Internet of Things (Iot) is a key driver of the digitisation of our society and economy by providing global interconnection between objects and people through communication networks. It is quickly expanding to all domains, including consumers (e.g. smart homes), transportation (e.g. connected cars, railways signaling protocol) and manufacturing (Industry 4.0). In 2020, the number of IoT endpoints will reach 5.8 billion with an annual growth rate of about 20% which is expected to continue over the coming years (Gartner, 2019).

Achieving an efficient maintenance of such a large network is critical to ensure the progressive deployment of applications and to update them with new features, functional improvements or security patches (Acosta Padilla et al., 2016). Physical maintenance is cost-prohibitive and too much time-consuming for most implementations, so remote updates are used to ensure the evolution of the firmware code running inside the IoT devices. Such updates may even be carried out over-the-air for mobile devices such as in connected cars.

Of course, the update phase is a critical process requiring to go through different steps. Key steps of a typical update process are depicted in Figure 1: the vendor (V) generates a new firmware which is made available for distribution by specific repositories (R) from which it is transferred to the target device (D), possibly through some intermediary device (e.g. a smartphone). Optionally, secondary devices (S) under the control of the main device can also be updated in the same process (e.g. secondary ECUs in a car). The various communication links cannot be assumed secured Hence, end-to-end security needs to be enforced to avoid any security threats which could be exploited by an attacker to block the updated device or to inject some malware to start spying, stealing data or even to take control over the device.

Figure 1: Global overview of firmware update process.

This paper aims at giving a comprehensive overview about existing firmware update protocols and how the securely of such protocols can be assessed. Our goal is not to perform an exhaustive survey but rather to show commonalities and differences and also to highlight useful tools to understand and analyse more deeply those protocols using formal modelling and verification. For this purpose, we selected one of the studied protocol called UpKit and used the Tamarin tool to illustrate how typical security properties can be specified and analysed. Based on this, we also discuss some lessons learned so far and identify some

Ponsard, C. and Darquennes, D.
Towards Formal Security Verification of Over-the-Air Update Protocol: Requirements, Survey and UpKit Case Study.
DOI: 10.5220/0010431408000808
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 800-808
ISBN: 978-989-758-491-6

research directions to deepen our work and the work of others in this field.

The paper is structured as follows. First, Section 2 reminds about the main security requirements for performing firmware updates and reports about some known attacks targeting those requirements. Section 3 then reviews how some representative implementations cope with those requirements and attacks by providing informal justification. Section 4 shows how formal modelling and verification can provide deeper confidence on the correctness of an update protocol based on the Tamarin tool. Section 5 discusses some lessons learned so far about designing and verifying such protocols. Finally, Section 6 concludes and presents our future work.

2 KEY SECURITY REQUIREMENTS AND RELATED ATTACKS

Firmware is a very sensitive information to protect. The key requirements are that:

- *Only Genuine Firmware may be Installed on Managed Devices*. This requirement relates to integrity of the image and authentication of the source, both can be achieved using digital signatures.
- *The Process Must be Transactional*: either succeed with new version of keep old version but not result in a bricked state of the updated device.
- *Devices can Only Upgrade and Generally to the Last Version*. Downgrading can result in lost of compatibility and also exposes to vulnerabilities corrected in a more recent version.
- *Devices Need to Updated in Some Defined Time Frame*. It means the system must be available and the freshness of the requests must be guaranteed.
- *A Proof of Update May be Required*, e.g. to make sure all managed devices have been correctly updated.
- *In Case of Dependencies among Subsystems, All Subsystems Must be Updated Consistently*, hence the need for a protocol involving primary/secondary devices. This is a consistency property.
- *The Firmware Itself Shall not be Analysable by Third Parties*. This is a confidentiality security property and requires to secure communications with encryption techniques.

An attacker targeting a firmware update system might aim at trying to:

- *Read the Firmware*, in order to extract some information from it. This may be designed secret but also possibly as a preparation phase for a more elaborated attack, e.g. to install a malware in the IoT device.
- *Block Updates*, preventing the deployment of new functions and keeping the infrastructure vulnerable.
- *Block the System*, making the system unavailable through some form of corrupted updates.
- *Take Control of the System*, through the installation of its own code, possibly to issue malicious/harmful commands, to steal information or to progress in the achievement of a larger attack.

Based on the above requirements and attacker goals, a number of attacks on update systems are already known:

- *Arbitrary Software Installation*. An attacker is able to install its own image in response to download request without being detected.
- *Indefinite Freeze Attacks*. The attacker replaces updates by current version preventing all update.
- *Rollback Attacks*: the attacker is able to install an old version that may contain vulnerabilities.
- *Fast-forward Attacks*. The attacker increases the version number and makes it difficult to recover a legitimate versions appearing as rollbacks.
- *Distributed Denial of Service Attack*. The attacker is flooding an update server with requests so it will crash or won't be able to serve legitimate requests.
- *Endless Data Attacks*. The attacker responds to a file download request with an endless stream of data, possibly causing failures if the case is not well managed.
- *Slow Retrieval Attacks*. An attacker responds to clients with a very slow stream of data preventing update process to complete.
- *Mix-and-Match Attacks*. An attacker presents clients with inconsistent bundles resulting in update failures or other complications.
- *Wrong Software Installation*. An attacker provides a client with a trusted file that is just not the one the client wanted.
- *Vulnerability to Key Compromises*. An attacker can take control of private keys, especially in system relying on a single key.

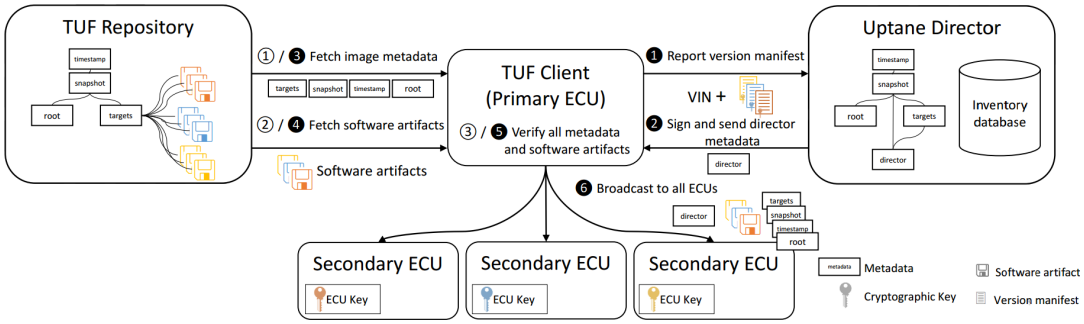

Figure 3: TUF+Uptane Architecture (Asokan et al., 2018).

3 SURVEY OF FIRMWARE UPDATE PROTOCOLS

This section reviews and compares different update protocols. The aim is not to be exhaustive but more representative of existing solutions in order to be able to discuss them. A more systematic survey is proposed by (Mtetwa et al., 2019).

3.1 MQTT-based Design

Message Queuing Telemetry Transport (MQTT) is a lightweight, publish-subscribe network protocol to transport messages between devices. It is an open OASIS and ISO standard (ISO/IEC 20922) (OASIS, 2019) and is designed to fit IoT needs with devices with small code footprint and/or limited bandwidth. Various firmware update implementations rely on this protocol and some of its features such as the publish/-subscribe protocol. As part of the FreeRTOS Real-time operating system for microcontrollers, an over-the-air (OTA) client library is available to manage the notification of a newly available update, download the update, and perform cryptographic verification of the firmware update using MQTT, with also the ability to manage secondary devices (AWS, 2020).

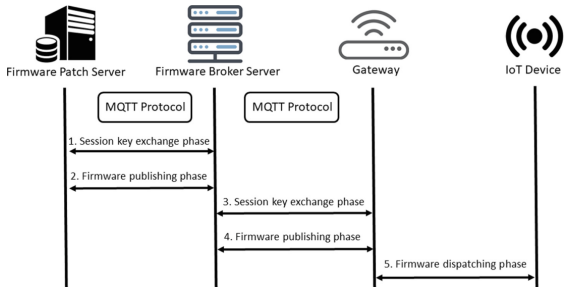

Figure 2: MQTT-based design.

An MQTT-based update protocol is also specified in details in (Lo and Hsu, 2019) and depicted in Figure 2. The first and second steps rely on MQTT to mutually authenticate a firmware patch server and a firmware broker server, to establish the session key between them and to transfer a new firmware. Its integrity and source are then verified. The third and fourth phases also utilize MQTT to mutually authenticate the broker server and the gateway and then publish the new firmware to the gateway. Again, the firmware integrity and source are verified. The final fifth phase mutually authenticates the gateway and the IoT device before dispatching the new version of firmware to the corresponding IoT device where is is installed.

3.2 TUF/Uptane Automotive Design

The Update Framework (TUF) is a flexible framework to secure new and existing software update systems. It was designed in 2010 to meet the needs of a wide variety of software update systems, to easily integrate with existing software update systems (Samuel et al., 2010). It is characterised by a high resilience against compromised keys and other attacks that can spread malware in a repository or inside an device. It is based on the definition of separate roles which sign specific aspects of the metadata with their own keys. In addition, TUF requires a threshold number of signatures, supports for explicit and implicit revocation of keys, and can mandate the offline management of vulnerable keys.

Uptane is a variant of TUF which better fits the needs of consistently updating the several Engine Control Unit (ECUs) found in cars (Kuppusamy et al., 2018). It is organised in two levels: more powerful primary ECUs are managing simpler secondary ECUs. The main addition is the introduction of a director repository to manage the update process and two levels of verification (full or partial). It is resilient to the best efforts of nation state attackers and has become the de facto standard in automotive. It is also considered for adaptation in other domains, e.g. railways (Galibus, 2019).

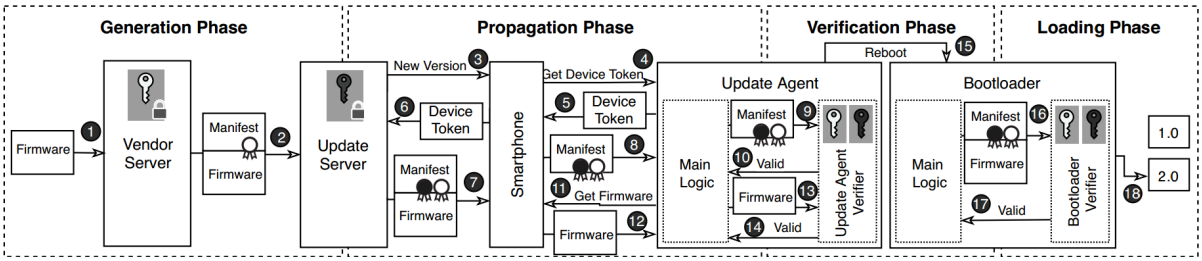

Figure 4: UpKit Update Architecture (Langiu et al., 2019).

Figure 3 shows the sequence of events in Uptane. (1) the primary ECU reports a vehicle version manifest (i.e. its signed software configuration) to the remote director repository, along with the Vehicle Identification Number (VIN). (2) The director repository determines the correct, up-to-date software configuration for all ECUs of the target vehicle and signs director metadata describing all software artifacts each ECU must install. (3-5) The primary ECU retrieves and verifies all metadata and software artifacts on behalf of secondary ECUs and (5) distributes them over the vehicle's local-area network along with the metadata. Secondary ECU may only perform a partial verification of the director signature if not powerful enough to perform a full verification.

3.3 Lightweight UpKit Design

UpKit is a portable and lightweight software update framework for constrained IoT devices (Langiu et al., 2019). It proposes a refined update architecture independent of the firmware distribution channels. It covers all phases of the update process of Figure 1. It is able to guarantee update freshness thanks to a double-signature process and can reject invalid software at an early stage to avoid an unnecessary reboot which could impact availability.

The full process is detailed in Figure 4. It starts with a classical generation phase where (1-2) the image is transferred to the image repository together with a manifest allowing its verification. In the propagation phase, (3) the new version is advertised for availability and (4-6) target devices send back a request token which is used in (7) to generate a request specific firmware image (possibly differential for reducing footprint and processing needs) together with the addition of a second signature from the repository server. (8) The image and manifest can then be transferred to the device possibly through intermediary devices such as a smartphone. In the IoT device, an update module is running in front of the bootloader and (9) performs an early check of the double manifest signatures. Only in case of success (10), the image is transferred (11-13) to go in the loading phase. In (16-17), a final verification occurs just before (18) switching versions otherwise the current image is kept.

3.4 Blockchain-based Design

In contrast with the previous architectures based on the client-server model, a blockchain architecture is proposed in (Lee and Lee, 2017). Blockchain is an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way (Iansiti and Lakhani, 2017). It is thus interesting both for its ability to ensure integrity but also for its distributed nature resulting in a high resilience compared to client-server architecture where the server is a single point of failure that can undergo Distributed Denials of Service (DDoS) attacks.

Figure 5 shows the blockchain-based firmware update. The manifest is distributed through the blockchain network and contains a unique hash value of the firmware image. Images are stored in a distributed file system with hash values of firmware images being stored in the blockchain. To cope with the limited computational of IoT devices, all verifications are performed by blockchain nodes. Specific blockchain nodes are configured to manage the registration, i.e. to upload the manifest and firmware image) and the retrieval, i.e. to handle IoT devices requests for manifest and firmware images.

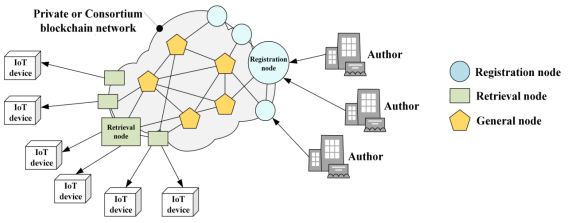

Figure 5: Blockchain-based firmware update platform (Lee and Lee, 2017).

The Update process is as follows:

1. author registration of a new firmware to a registration node.
2. registration check: the author is registered if not yet and the manifest is checked against the author

Table 1: Comparison summary.

Design	Comm.	Image Integrity	Image Freshness	Image Availability	Secondary Update	Other remarks
MQTT-based	MQTT	Signature	Publish/ subscribe	Server	Yes for OpenRTOS	
TUF Uptane	Generic	Multiple signatures	Timeserver	Server	Yes	Multiple roles Partial Verif.
UpKit	Generic	Double signature	Double signature	Server	No	Early check
Blockchain	Generic	Hash in blockchain	Blockchain	Distributed	No	Blockchain processing (trust)

public key.

3. manifest stored in the blockchain which cannot be deleted nor tampered.
4. firmware image stored on distributed file system with control hashed kept in the blockchain.
5. IoT devices perform periodic check for updates and blockchain provides last version information which can trigger an update process.
6. manifest downloaded to the IoT device (assuming secured channel).
7. image downloaded to the device from the P2P file system and checked using the author public key.

Key advantages over client-server architecture are that the repository cannot be tampered, that update will stay available even if the vendor disappears and that it can also cope with DDoS attack. It also moves computation tasks to the blockchain provided a trust assumption can be made.

3.5 Comparison Summary

Table 1 summarises the main security characteristics of the presented update protocols. Note that confidentiality is not covered here although firmware analysis could help gaining knowledge about vulnerabilities, the most important requirements are integrity and availability. With respect to integrity MQTT provides simple signatures while Uptane/UpKit supports more elaborated signatures schemes. Blockchain is also very good to ensure image integrity, especially avoiding the tampering of the repository itself. A weak point of Uptane is the time server based design which is addressed by UpKit using a delivery protocol on an instance basis. The blockchain design can provide trusted timestamping services at the required precision although the precise timekeeping on distributed system is known to be a harsh problem (Hong, 2020). Image availability is also more reliable on a blockchain design given the distributed nature compared to the traditional server-based design of the other solutions.

The rest of this paper will focus on UpKit protocol as simple and lightweight protocol to illustrate how to carry out a verification activity against some of the identified security requirements.

4 FORMAL MODEL AND PARTIAL VERIFICATION OF THE UpKit PROTOCOL

A striking fact is that some industrial protocols like TUF and Uptane claim to be able to cope with attacks at nation state level but the available papers only provide informal arguments about how the protocol design is achieving such resistance (Kuppusamy et al., 2018). When deploying security critical protocols to manage large network of devices, it is interesting to produce more rigorous analysis and evidence that the protocol is able to ensure security requirements and resists known attack patterns described in Section 2. Some generic formal tools have been applied to formal security verification, e.g. Alloy (Grisham et al.,). However, analysing security requires to reason about knowledge, to model specific cryptographic primitives (e.g. hashing, signing, PKI infrastructure) and to capture the attacker behaviours (e.g. Dolev-Yao adversary model (Dolev and Yao, 1983)). In the scope of this work, we selected the Tamarin tool (Basin et al., 2017) which has proven its ability to analyse complex protocol and to manage the tracking of design changes. It was also already used to model IoT protocols (Kim et al., 2017). Unbounded verification tools such as ProVerif (Blanchet et al., 2005) are also reported to be efficient, but without guarantee of termination in the case of complex protocol.

This section first introduces the general infrastructure and the attacker capabilities before performing partial verification on the generation and propagation phases of the UpKit protocol presented in Section 3.3. Various Tamarin constructs are also introduced.

4.1 Infrastructure Modelling

Listing 1 details how distribution actors can declare a pair of public/private keys and a device can be attributed an ID. Generic Tamarin rules are used for this and take the form `[ precondition ] --[ action ]-> [postcondition]`. A fresh fact is used to generate new values, e.g. a fresh key `Fr(~ltkA)`, which can be associated with specific public variables `($A)` using a specific fact to record that knowledge, e.g. `!Ltk($A,~ltka)`. A public `Out` channel is available to send information and make it known by everybody, including attackers, e.g. `Out(pk(~ltkA))` is distributing the public key. In addition, the `Reveal_ltk` rule shows how to model the fact and attacker might compromise a private key.

Listing 1: Modelling: PKI and devices identification.

```
// Public key infrastructure
rule Register_pk:
[ Fr(~ltkA) ]          // fresh private key
-->
[ !Ltk($A, ~ltkA)
// private key known by owner
, !Pk($A, pk(~ltkA))
// corresponding public key
, Out(pk(~ltkA)) ]     // public key is public

rule Register_device:
[ Fr(~id) ]            // generate device ID
-->
[ !Id($D,~id)          // ID known by device
, Out(~id) ]           // ID public

// Compromising an agent's private key
rule Reveal_ltk:
[ !Ltk(A, ltkA) ]      // key in question
--[ Reveal(A) ]->
// reveal action (for use in lemma)
[ Out(ltkA) ]          // private key divulgated
```

4.2 Generation Phase

Listing 2 shows the generation phase which is based on two rules respectively modelling the emission by the vendor `V_1_send_img` and reception by the repository `Repo_1_recv_img` of an image together with its manifest (simply the signature here). Specific predicates capture the send and receive actions as well as the signature verification. Signature primitives are directly available through the signing builtin package. Two proof lemmas are used: the `repo_update_possible` simply checks that the model can produce traces for transferring an image while the `vendor_authentication` lemma captures the security requirement that the repository can only record a correctly verified image unless the private key has been compromised (cf. attacker capabilities). In order to perform the verification, traces are restricted only to those verifying the signature. Tamarin autoprovers manage to prove both lemmas almost instantly.

Listing 2: Verification of the generation phase.

```
// Vendor V sends firmware to repo server
rule V_1_send_img:
let img = <V, ~fw>
in
[ Fr(~fw)
, !Ltk(V, ltkV)
, !Pk(R, pkR)    ]
--[ SendToRepo(V, img) ]->
[ Out(<img, sign(img, ltkV)>) ]

// Update repo server receiving firmware
rule Repo_1_recv_img:
let img=<V,fw> in
[ !Pk(V, pkV)
, !Ltk(R, ltkR)
, In(<img, sig>)
, Fr(~newVersion) ]
--[ RecvInRepo(R, img)
, Eq(verify(sig,img,pkV),true)
, Authentic(V,img), Honest(R), Honest(V)
, RepoNewImage(R,img) ]->
[ !Repo(R, fw, V, sig),
  Out(~newVersion) ]

restriction Equality:
"All x y #i. Eq(x,y) @i ==> x = y"

lemma repo_update_possible:
// checking model not void
exists-trace
"Ex S R img #i #j. SendToRepo(S,img) @i
                 & RecvInRepo(R,img) @j"

lemma vendor_authentication: // security property
"All b m #i. Authentic(b,m) @i
==> (Ex #j. SendToRepo(b,m) @j & j<i)
| (Ex B #r. Reveal(B)@r & Honest(B) @i & r < i)"
end
```

4.3 Propagation Phase

The propagation phase is described in Listing 3. After advertising a new image is available, the manifest is enriched with a second signature corresponding to match a specific request from the device to enforce freshness. The model is quite similar to the previous generation phase except the exchange is initiated by the repository. Again two lemmas are used: `device_check_possible` checks about the feasibility of the exchange to detect flaws in the model and `double_signature_ok` checks the second signature,

again provided the attacker has not compromised the repository private key. Note the model is simplified as the intermediary smartphone is not modelled and the first signature is not checked given the image was not yet requested. Again Tamarin autoprovers manage to check the solution almost instantaneously.

Listing 3: Verification of the generation phase.

```
rule D_3_Token:
[ In(newVersion),
  !Id(D,id),
  Fr(~req)  ]
--[ DeviceRequest(D,~req) ]->
[ !Token(D,~req)
, Out(~req) ]

rule R_7_Update:
[ In(req),
  !Repo(R, fw, V, sig),
  !Ltk(R, ltkR) ]
-->
[ Out(<sig,sign(<sig,req>,ltkR)>) ]

rule D_9_manifest:
[ In(<sig,sig2>),
  !Token(D,tok),
  !Pk(R, pkR) ]
--[ Eq(verify(sig2,<sig,tok>,pkR),true),
    ManifestOK(D,sig) ]->
[ !Checked(D,sig) ]

restriction Equality:
"All x y #i. Eq(x,y) @i ==> x = y"

lemma device_check_possible:
exists-trace
"Ex R D sig #i #j. RepoNewImage(R,sig) @i
                 & ManifestOK(D,sig) @j"

lemma double_signature_ok:
"All D sig #i. ManifestOK(D,sig) @i
==> (Ex R #j. RepoNewImage(R,sig) @j & (j<i))
| (Ex B #r. Reveal(B)@r & (r<i))"
```

5 DISCUSSION OVER RELATED WORK

The comparison of our work with a systematic survey (Mtetwa et al., 2019) reveals that we cover two key categories of designs: server-based and block-chained based. The survey identified 5 blockchain solutions while we identified only 2 of them but we could capture all the important features of such solution. Over the air update to vehicles is largely present in the survey and in our work, although the authors seems to have missed TUF, Uptane and UpKit. On the technological side, we also detailed a MQTT-based solution which is is more at implementation level but used in frameworks such as OpenRTOS from Amazon.

Modelling using a formal language and tool as always a noticeable learning curve. In our case, the Tamarin tooling is quite easy to install and benefits from a nice web interface. Many reference documents are available: manual, tutorial and an important model repository together with explanatory papers (Tamarin Team, 2016). However, getting into the notations with the correct state of mind requires some effort. Developing a model is a progressive endeavor relying of different progressive milestones where intermediary checks can be introduced. In our case, the division into different logical phases was a natural decomposition of the problem. During the work, we also identified a number of modelling patterns, e.g. to deal keys, identifiers, signatures, specific agents which can be quite systematically reused to accelerate the modelling and verification of the remaining steps.

So far, the process mainly helped to clarify the specification and to precisely define what is required as information in the various requests. The model is still partial but can easily be extended to cover the full update cycle. The knowledge gained can also be applied to more complex processes than UpKit, for example Uptane to cope with the split of responsibilities across different roles with specific keys and more complex configuration including secondary updates. Modelling the timeserver seems also possible using the time ordering operators. Some attacks at technical level seem however beyond the reach of formal validation tools such as the direct control of the communication stream unless this is explicitly prohibited through the use of secured channels.

Tamarin has been widely used for verifying security protocols. Related to IoT, it has been applied to the formal analysis of the Sigfox, LoRa and MQTT protocols (Kim et al., 2017). Those authors show that the majority of IoT protocols are vulnerable to cryptographic DoS attacks and propose a protection based on a server puzzle. In this work, we did not model such attacks at this point but identified blockchain design providing a more intrinsic protection to such attacks due to the absence of single pointer of failure at server repositories. However, we did not investigate possible performance or scalability issues related to this technology.

Another formal verification approach is to consider statistical model checking where attack success has some probability. A simulation-based approach is used to reason about precise properties specified in a stochastic temporal logic (Legay et al., 2019). However, these techniques are currently used to perform a more abstract form of security analysis based on at-

tack and defense trees. This results in quantitative analysis techniques using tools such as Uppaal (Kumar and Stoelinga, 2017) or PRISM (Aslanyan et al., 2016). In our work, we identified threats and countermeasures but did not organise them in a systematic attack-defense tree. This could be another complementary approach to analyse the system security.

6 CONCLUSION AND NEXT STEPS

In this work, we reviewed a representative sample of firmware update protocols and identified how they can cope with security requirements and related attacks. We highlighted the need of a more formal verification approach. We proposed a partial modelling of the UpKit protocol and could successfully verify key integrity and freshness properties. Although partial and anchored in a specific protocol, our results so far show several commonalities and the modelling and verification approach can be generalised to most protocols and security properties. The sketched approach can also be used to verify security properties on the MQTT or Uptane protocols. The distributed nature of the blockchain design requires specific techniques to get security assurance. They can however be captured by assumptions to carry out proofs on the interaction between the device and the blockchain.

In the next steps of our research, we plan to cover the lasts steps of the UpKit protocol and then to model the more complex Uptane protocol, including multiple roles with related keys, secondary ECUs and the use of a timeserver. We also plan to investigate more deeply the blockchain-based designs and to better structure security requirements, attacks and design countermeasures using an attack-defense tree.

REFERENCES

Acosta Padilla, F. J. et al. (2016). The Future of IoT Software Must be Updated. In *IAB Workshop on Internet of Things Software Update (IoTSU)*. Internet Architecture Board (IAB).

Aslanyan, Z., Nielson, F., and Parker, D. (2016). Quantitative verification and synthesis of attack-defence scenarios. In *Proc. 29th IEEE Computer Security Foundations Symposium (CSF'16)*, pages 105–119. IEEE.

Asokan, N. et al. (2018). Assured: Architecture for secure software update of realistic embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11).

AWS (2020). FreeRTOS User Guide. https://docs.aws.amazon.com/freertos/latest/userguide.

Basin, D., Cremers, C., Dreier, J., and Sasse, R. (2017). Symbolically Analyzing Security Protocols Using Tamarin. *ACM SIGLOG News*, 4(4):19–30.

Blanchet, B., Abadi, M., and Fournet, C. (2005). Automated verification of selected equivalences for security protocols. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, pages 331–340.

Dolev, D. and Yao, A. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208.

Galibus, T. (2019). Securing software updates for trains. In *Critical Information Infrastructures Security - 14th International Conference, CRITIS 2019, Linköping, Sweden, Sept. 23-25,*, volume 11777, pages 137–148. Springer.

Gartner (2019). IoT endpoints 2020: the industries and use cases driving growth. https://www.i-scoop.eu/internet-of-things-guide/iot-endpoints-2020.

Grisham, P. S., Chen, C. L., Khurshid, S., and Perry, D. E. Validation of a security model with the alloy analyzer.

Hong, D. (2020). On decentralized clocks: How time became the biggest security threat on blockchain systems. https://unifiedh.medium.com.

Iansiti, M. and Lakhani, K. R. (2017). The Truth About Blockchain. Harvard Business Review https://hbr.org/2017/01/the-truth-about-blockchain.

Kim, J. Y., Holz, R., Hu, W., and Jha, S. (2017). Automated analysis of secure internet of things protocols. In *Proc. of the 33rd Annual Computer Security Applications Conference*, ACSAC 2017, page 238–249.

Kumar, R. and Stoelinga, M. (2017). Quantitative security and safety analysis with attack-fault trees. In *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*.

Kuppusamy, T. K., DeLong, L. A., and Cappos, J. (2018). Uptane: Security and customizability of software updates for vehicles. *IEEE Vehicular Technology Magazine*, 13(1):66–73.

Langiu, A., Boano, C. A., Schuß, M., and Römer, K. (2019). Upkit: An open-source, portable, and lightweight update framework for constrained iot devices. In *IEEE 39th Int. Conference on Distributed Computing Systems (ICDCS)*, pages 2101–2112.

Lee, B. and Lee, J.-H. (2017). Blockchain-based secure firmware update for embedded devices in an internet of things environment. *J. Supercomput.*, 73(3):1152–1167.

Legay, A. et al. (2019). *Statistical Model Checking*, pages 478–504. Springer International Publishing, Cham.

Lo, N.-W. and Hsu, S.-H. (2019). A secure iot firmware update framework based on mqtt protocol. In *Proc. of 40th Anniversary Int. Conf. on Information Systems Architecture and Technology – ISAT 2019*, pages 187–198.

Mtetwa, N. S., Tarwireyi, P., Abu-Mahfouz, A. M., and Adigun, M. O. (2019). Secure firmware updates in the internet of things: A survey. In *International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, pages 1–7.

OASIS (2019). MQTT Version 5.0. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

Samuel, J. et al. (2010). Survivable key compromise in software update systems. In *17th ACM Conf. on Computer and Communications Security*, page 61–72.

Tamarin Team (2016). Tamarin-Prover Manual - Security Protocol Analysis in the Symbolic Model. https://tamarin-prover.github.io/manual.