Open access • Proceedings Article • DOI:10.1109/GIOTS.2018.8534565

LATe: A Lightweight Authenticated Time Synchronization Protocol for IoT

Renzo E. Navas, Laurent Toutain

Published on: 04 Jun 2018 - The Internet of Things

Topics: Synchronization (computer science) and Synchronization

LATe: A Lightweight Authenticated Time Synchronization Protocol for IoT

Renzo Efrain Navas, Laurent Toutain

LATe: A Lightweight Authenticated Time Synchronization Protocol for IoT

Renzo E. Navas, Laurent Toutain
Network Systems, Cybersecurity and Digital Law Department
IMT Atlantique
Cesson-Sevigne, France
{renzo.navas, laurent.toutain}@imt-atlantique.fr

***Abstract*—Time synchronization is fundamental for a wide variety of IoT applications. Time is also fundamental to provide security services such as certificates or OAuth-token validation. Having a secure source of time is a fundamental problem, and the first step to provide other services for applications. There is no standardized lightweight and secure time synchronization solution suitable for IoT. We propose a Lightweight Authenticated Time (LATe) Synchronization Protocol. Our proposal is based on IETF open standards and is agnostic to underlying communication technologies. We also provide a computer-aided proof of the security claims using the Scyther tool.**

***Index Terms*—time ; synchronization; secure; authenticated; protocol design ; ietf ; formal method; verification ; Scyther;**

I. Introduction

Synchronized time is needed in several Internet of Things (IoT) applications, from time-stamping of sensor data to the establishment of authenticated secure channels. However, many time synchronization protocols are not secure: they assume existing secured communication channels. The establishment of secure channels, in most cases, assumes a secure source of time e.g. to assure freshness of transactions. This creates a *circular dependence* problem that has already been spotted on the standardization community. Time protocols are being designed to overcome this, such as the Internet Engineering Task Force (IETF) work-in-progress Network Time Security (NTS) [1]. However, NTS or secure-versions of existing time protocols, are not designed for the IoT constraints, e.g. in NTS the simplest time synchronization takes at least six messages, including a Datagram Transport Layer Security (DTLS) handshake, and the format of each of the message is not optimized in terms of size.

Our study provides a solution for a coarse-grained *secure time synchronization* problem in a *lightweight* manner, i.e. requiring the least possible messages at the synchronizing node, minimizing the number of cryptographic operations and optimizing the traffic sent on the network. It is not a goal to provide precise time synchronization; we also use open standards to encode the messages, and the proposed protocol is agnostic to the underlying technologies (e.g. a packet switched network), guaranteeing end-to-end security properties among heterogeneous networks and in the presence of untrusted nodes.

The rest of this paper is structured as follows: Section II briefly discuss state of the art and related work on time synchronization. Sections III-IV describe our proposed solution. Section V presents a formal proof of the protocol and Section VI analyses possible attacks. On Section VII we compare our solution against different protocols. Finally, Sections VIII-IX offer some perspectives and a final conclusion for our work.

II. State of The Art for Secure Time Synchronization

Prominent standardized time synchronization protocols are the IETF Network Time Protocol (NTP) [2], IEEE 1588 Precision-Time-Protocol (PTP), and satellite-based Global Navigation Satellite System (GNSS). An excellent overview of time synchronization protocols over packet-switched networks is done in [3], it also analyses security threats and solutions. Moussa et al. [4] focus on time synchronization for the smart grid and its security requirements. Current standardized solutions to achieve secure time synchronization include *Annex K* of PTP, and authenticated mode of NTP. Design of secure time synchronization protocols from scratch is an active topic, such as the aforementioned Network Time Security for NTP [1]. The IETF has released a document [5] that specifies the threats and security requirements for future time protocols. Current standardization efforts do not deal with the specific constraints of IoT, and focus mostly on precision and robustness at the expense of increased requirements at the node and network. A standard suitable for IoT is an unsolved problem.

Outside standardization bodies the secure time synchronization problem has been prominently studied for wireless-sensor-networks (WSN) [6][7][8][9]. WSN share many of IoT constraints. However the aforementioned solutions either require already loose time synchronization, use asymmetric cryptography, or they use nonces but requiring more messages exchanges than our proposed solution. On Section VII we will compare them to our proposed solution. To our knowledge none of the proposed lightweight time synchronization methods have been formally proved with computer-aided cryptographic tools.

III. Lightweight Authenticated Time (LATe) Synchronization Protocol: Semantics

A. Background and justification

The non-cryptographic part of the proposed protocol can be traced to Cristian's time synchronization protocol [10].

However, the problem that needs to be solved concurrently is related to security and is *how to assure the freshness and authentication of an exchange of information in the absence of time-awareness*. The concept of authenticated and fresh exchange of information is generalized by Bauer et al. [11] with the concept of *event-markers*.

The proposed solution is intended to be the simplest possible to the secure time synchronization problem: namely using an *event-marker* for a two-message protocol; but our contribution also has the added value of using open standards suitable for the IoT and presenting a computer-aided security proof.

B. LATe Synchronization Protocol Entities

The nonce-based *Lightweight Authenticated Time (LATe) Synchronization Protocol* is our solution that allows to securely bootstrap time. The protocol involves two entities. *Time Client (TC)*: the entity that attempts to update its local time representation. *Time Server (TS)*: the entity that provides its local time representation. *TC* and *TS* have valid pre-shared cryptographic material. The messages are transported over unsecured communication channels.

C. Protocol Goals

Functional Goal: Provide an entity, i.e. the *Time Client*, with the time representation from a trusted party, i.e. the *Time Server*.

Security Goals: **(1)** ***Data Authentication***: The time representation must be *authenticated*, data origin-authentication: coming from the intended party. **(2)** ***Data Integrity***: The time representation must be *integrity-protected*, an alteration of the original information must be detected. **(3)** ***Freshness***: The time representation must be *fresh*, it corresponds to the current run of the protocol and not replayed from an earlier run.

Design Goals: *(1) Lightweight*: Minimize the number of messages to exchange; minimize the cryptographic operations to execute (in terms of complexity, that will be equivalent to minimize CPU processing power-time needed at the entities); minimize the information to exchange and provide a compact-representation of the information over the channel[1]. *(2) Agnostic to underlying communication technologies:* The protocol messages should be easily transported over any underlying communication technology (wired, wireless, Ethernet, IP, non-IP, datagram oriented, etc)[2]. *(3) Cryptographic agility:* The crypto-primitives used by the protocol must be easily interchangeable, e.g. ready for future algorithms, or if an attack is discovered in current one easily to replace with other.

Non-goal: Precise, fine-grained, time synchronization its not a goal. e.g. not synchronize at the order of μs but rather at ms (will be determined by round-trip delay time of the network).

D. The LATe Synchronization Protocol

The Lightweight Authenticated Time (LATe) Synchronization Protocol consists of two messages exchanged between a *Time Client (TC)* and a *Time Server (TS)*. K_{CS} is a symmetric pre-shared key between TC and TS. $MAC_{K_k}(M)$ is a message authentication code of message M using shared key K_k. A protocol run can be described as follows:

1) TC generates a random nonce N_C
2) *TC sends to TS Message 1*. Containing: ID_C the identity representation of TC, and N_C.
3) *TS sends to TC Message 2*. Containing: N_C, $Time_S$ the local time representation of TS, and $MAC_{K_{CS}}(N_C, Time_S)$ a message authentication code of N_C and $Time_S$ using the key K_{CS}
4) TC can synchronize its internal time representation according to subsection III-E

The protocol is described on Figure 1.

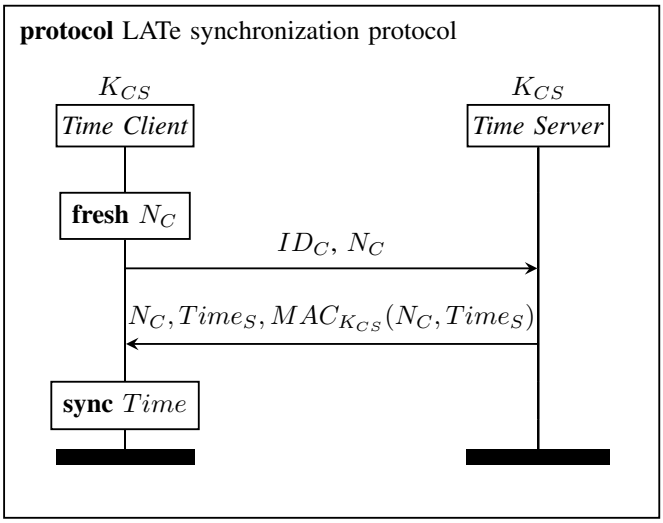

Fig. 1. LATe Synchonization Protocol Diagram. K_{CS} is a symmetric pre-shared key between the *Time Client (TC)* and the *Time Server (TS)*. ID_C is the identity representation of TC. N_C is a nonce generated by TC.

E. Time Synchronization Calculation

The *Time Client (TC)* will have to run the following steps to achieve authenticated time synchronization:

1) Timestamp when it sends *Message 1*: T_1.
2) Validate *Message 2*:
   - Verify nonce N'_C on *Message 2* matches N_C sent on *Message 1*. (*Freshness*)
   - Verify *data authentication and integrity*: C Calculates $MAC_{K_{CS}}(N_C, Time_S)$ and compares with the received value on *Message 2*.
3) Calculate *Round Trip Time (RTT)* as $RTT = T_2 - T_1$, where T_2 is the local time of TC when performing this calculation.
4) Set the internal time representation T_C as $T_C = Time_S + \frac{RTT}{2}$, the associated uncertainty is $\pm\frac{RTT}{2}$

IV. LATe Synchronization Protocol: Syntax

A. IETF standards: CBOR and COSE

The Internet Engineering Task Force (IETF) is an open standards organization that has developed and published many

[1]Not a semantic goal but strictly related with the syntax of the protocol.
[2]Idem footnote 1.

TABLE I
CBOR MAP "TIC INFORMATION" OBJECT DEFINITION

Parameter name	CBOR Key	Value Type	Description
nonce	4	binary string	A random nonce
kid	5	binary string	Key-ID is an opaque value and identifies the cryptographic key to be used in the response
alg (optional)	6	int	Identifies the crypto-graphic algorithm to be used in the response
server (optional)	7	string	Identifies the intended Server for time syn-chronization (Absulute URI)

of the protocols that are in use on the Internet (e.g. IP and TCP). We use the CBOR and COSE IETF standards to encode the LATe messages. The *Concise Binary Object Representation* (CBOR)[12] is a binary data format inspired by JSON and provides a compact representation of most common data types used at Internet standards; it also has the explicit goals of a lightweight implementation in terms of code and RAM needed. For the security services of the messages we use *CBOR Object Signing and Encryption (COSE)* [13]. COSE describes how to create and process encryption, signatures and message authentication codes using CBOR for serialization. Security using COSE is at the application-layer of the network it is also referred as *object security*, the security properties can be maintained end-to-end (even if different technologies are used at lower layers, and -untrusted- intermediate nodes are involved) and can be set on a per-message basis (as opposed to session oriented security, e.g. IPSec, D-TLS).

B. LATe Message Encodings

The protocol consists of two messages encoded with CBOR. COSE is used to cryptographically protect the second message. We define two new CBOR objects: *TIC Information* and *TOC Response*. Those objects are CBOR Maps which consist of *key-value* pairs of information. Additionally, to give semantic meaning to the objects without relying on external information we assign a CBOR Tag to each of the objects. CBOR Tag values range between $\pm$65536, and are registered on the Internet Assigned Numbers Authority, tags on the 1-23 range take one byte when encoded -but all are allocated-; tags in the 24-255 range take two bytes: we chose values in this range.

1) Message 1 - TIC Information: The message will consist of a new CBOR MAP *TIC Information* as defined on Table I, we propose the CBOR Tag 59 to describe a *TIC Information* object. ***About the nonce generation***: Nonce must be at least 64-bits and cryptographically secure randomness is needed, a pseudo-random number generator may be used if the seed has sufficient entropy, for details see [14].

TABLE II
CBOR MAP "TOC RESPONSE" OBJECT DEFINITION

Parameter name	CBOR Key	Value Type	Description
time	3	unsigned int	Time representation information
nonce	4	binary string	A random nonce

The *Key-ID* is an opaque identifier of the key to be used by the server, it is the equivalent of the client's identity. The *Alg* field allows cryptoagility, some recommended algorithms are `HMAC w/SHA-256 truncated to 64 bits` (using a 256-bit pre-shared-key), `AES-CBC-MAC` or `AES-CMAC` (for both, 128-bit key will suffice). The client can explicitly request for a time server, e.g. in cases where the message is dealing with intermediate nodes. On Listing 1 we show a TIC Information object on human-readable CBOR diagnostic notation.

```
{ nonce:h'73616E206C6F7265',
  kid  :h'0001',
  alg  :4/*HMAC w/SHA-256 truncated to 64 bits*/}
```

Listing 1: *TIC Information* on CBOR diagnostic notation.

The binary representation of the same *TIC Information* object is found on Listing 2 the size of the message is 19 bytes.

```
D83B                    # tag(59) (TIC Info.)
  A3                    # map(3)

    04                  # unsigned(4) (=nonce)
    48                  # bytes(8)
    73616E206C6F7265    # Nonce Value

    05                  # unsigned(5) (=kid)
    42                  # bytes(2)
    0001                # Key-ID Value

    06                  # unsigned(6)   (=alg)
    04                  # unsigned(4)
```

Listing 2: *TIC Information* CBOR object (19 Bytes).

2) Message 2 - TOC Response: The message consists of a new CBOR MAP *TOC Information* as defined in Table II, we propose the CBOR Tag 60 to describe a *TOC Information* object. The TOC Information object contains the representation of the *time* from the server and a *nonce*.

The *TOC Response* object needs to include a Message Authentication Code, this security service will be provided by COSE using a `COSE_Mac0` object. A TOC Response authenticated and wrapped in COSE can be found on Listing 3 on CBOR diagnostic notation.

```
{protected: { /* Protected header of COSE_Mac0 Object*/
     kid: h'0001',
     alg: 4 /* HMAC w/ SHA-256 truncated to 64 bits */
    },
 payload    : { /* TOC Response CBOR MAP*/
     time   : 1477307841,
     nonce  : h'73616E206C6F7265'
    },
 tag        : h'36f5afaf0bab5d43' /* MAC Code*/}
```

Listing 3: *TOC Information* on CBOR diagnostic notation.

V. Formal method verification using Scyther

A. Security protocols verification

There are currently two main approaches to verify security protocols: the *provable security* and the *formal method* approach. *Provable security* defines a rigorous framework to define and prove (*theorem-proof*) cryptographic properties from a *mathematical* point of view, proving a protocol secure is hard on the provable security approach, and although there is criticism to this approach [15] it is still regarded as the most sound proof possible for a protocol. The *formal method* approach proposes a simpler model to describe an analyze cryptographic protocols, by abstracting basic properties (e.g: encryption), it assumes perfect cryptography (e.g. the cryptoprimitives can not be broken), and the attacker capabilities need to be modeled also (and restricted), then logical flaws can be found on such model. Several *formal methods* exists; the most known is the Burrows-Abadi-Needham (BAN) logic or *logic of beliefs*, and is deprecated: flaws have been found on protocols that have been proved secure on the BAN logic.

State-of-the-art approaches include the automatic falsification or verification of protocols with computer-aided tools like: Coq, CertiCrypt, EasyCrypt and CryptoVerif, all these aimed at achieve or help to manually achieve *computational security* -a subset of provable security-, in which the proof of security is *reduced* to the computational infeasibility of solving some mathematical problems for an adversary e.g. semi-prime factorization- (these methods cannot find particular attacks just prove they exist); on the other hand, tools like ProVerif, Scyther, and Tamarin are all three on a higher abstraction level (*formal methods* assuming a particular attacker model, e.g. the Dolev-Yao and perfect cryptography), they provide a weaker proof than a computational security one, but is easier to model complex cryptosystems.

B. The Scyther tool and a formal proof of LATe

The choice of the formal proof method for this paper is using the Scyther tool [16]. The reasoning behind is its simplicity to model cryptosystems, the attacker model found adequate to our setting, and the possibility to find concrete attacks. Scyther assumes perfect (or black-box) cryptography: the cryptoprimitives can not be broken. Another important assumption is the Dolev-Yao adversary model [17]. In Dolev-Yao an adversary has complete control over the communication channel: it can eavesdrop, intercept; modify, delete, and insert any message; the adversary is a legitimate user of the network.

To prove LATe security claims using the Scyther tool and its model we needed notably two additional tasks not straightforward: (1) express a message authentication code (MAC) function (the primitive does not exist); (2) express properly the security-authentication goals claimed.

To represent a MAC function over message m we use two primitives: $Enc_k(m)$ symmetric encryption of message m using key k, and a non-cryptographic hash function $H(m)$ (a hash function on Scyter is a one-way-function and known to every agent); then to obtain the keyed $MAC_k(m)$ of a message m we chose to encrypt-then-hash as follows $H(Enc_k(m))$, the captured semantical meaning is that only an agent in possession of the key k will be able to produce this one-way function over m.

Regarding the modeling of the authentication and freshness claims, Scyther offers the check of *secrecy* of a variable m, and the following notions of authentication: *aliveness*, *weak agreement*, *non-injective agreement* and *non-injective synchronization*. *Non-injective synchronization* requires that *all protocol messages occur in the expected order with the expected values*. Proving *non-injective synchronization* will implicitly include aliveness, weak agreement and non-injective agreement. For a deep analysis on authentication hierarchies and precise definitions see [18] and [19].

1) The LATe Protocol Description: The LATe Synchronization Protocol defined on the Security Protocol Description Language (SPDL) from Scyther is shown on Listing 4.

```
# LATe: Authenticated Time Synch Protocol

hashfunction H1;
usertype TimeStamp;

protocol LATe(I,R)
{
 role I   # Time Client - Initiator
 {
  fresh Na : Nonce;
  var T : TimeStamp;

  send_1(I,R,I,Na);
  recv_2(R,I,Na,T,H1({Na,T}k(I,R)));#encrypt-then-hash

  claim_I1(I,Nisynch); #encrypt-then-hash
  claim_I2(I,Niagree);
  claim_I3(I,Alive);
  claim_I4(I,Weakagree);
 }

 role R # Time Server - Responder
 {
  var Na : Nonce;
  fresh T : TimeStamp;

  recv_1(I,R,I,Na);
  send_2(R,I,Na,T,H1({Na,T}k(I,R)));#encrypt-then-hash
 }
}
```

Listing 4: LATe Protocol on Scyther's SPDL

C. Verify Results

We verify our protocol using Scyther v1.1.13 compiled from source running on OS Ubuntu 17.04 x64. The Scyther settings are: Maximum number of runs 0 (unbounded), Matching type "find all type flaws", advanced parameters were left to default values. The results are the following: all claims have been verified (*Nisynch*, *Niagree*, *Alive* and *Weakagree*).

Notably we achieved *non-injective synchronization* for the protocol. Secrecy of the *server time* was not a goal. The *data authentication-integrity* claims are satisfied by these results. However, the *non-injective synchronization* does not guarantee, by itself, the *freshness* goal of the LATe protocol, we will discuss this on Section VI.

VI. Attacks, mitigations and real-world issues

This section studies possible attacks, its mitigations, and discuss other real-world issues that affect the LATe protocol.

A. Replay-attack, Injectivity and the Freshness claim

Our protocol satisfies the notion of *non-injective synchronization*, however, this is not enough to claim resilience to *replay-attacks*. This kind of attacks can be formally ruled out by the notions of *injective agreement* and *injective synchronization*. *Injective-synchronization* is the strongest notion of authentication on the model we are using and -informally- is defined as follows: *"an Initiator I considers a protocol injectively synchronizing if the protocol (non-injective) synchronizes and each run of I corresponds to a unique run of Responder R."*. The *freshness* goal of our protocol is strictly related to the *injectivity* property. The question arises if our protocol satisfies *injective synchronization*, while we will not make a formal proof, that will involve to prove the LOOP property proposed in [19], but an affirmative response can be done, informally justified by observing that every client run will have a unique and unpredictable Nonce N_i which is used in all the messages exchanges with the Server in that run. This guarantees a one-to-one correspondence between all the messages of the same run, and message from others runs will not be able to be injected. On the formal model, a response that matches the nonce on the request, corresponds to the current run of the protocol and not another, it is *fresh*.

B. Real nonces and pre-play attack

The *injective synchronization* claim, who assure *freshness*, relies on the (idealized) properties of the Nonce as being unique and unpredictable. On practice this will not be the case, and the guarantees will be limited by the randomness quality of the nonce generation and by its length (not infinite). Shorter nonces will be more prone to collisions and *pre-play* attacks e.g. an attacker obtaining all possible nonce responses from the server, will be able to reply these responses -with old values of time- to any future client run of the protocol. To mitigate this risk one straightforward solution is to use longer nonces: e.g. 128-bits (the MAC-tag should also be increased accordingly). To make *pre-play* attacks infeasible (i.e. an attacker will not be able to obtain responses from the server to inject on the client) we define a stronger version of the protocol that includes the authentication of the first message as shown on Listing 5. **Avoiding randomness**: Authentication of the first message allows another refinement, the nonce does not need to be random and a counter (i.e. a sequence number) will suffice; the counter value must be stored on persistent memory to avoid being reset by an attacker.

1 : $C \rightarrow S : ID_C, N_C, MAC_{K_{CS}}(ID_C, N_C)$
2 : $S \rightarrow C : N_C, Time, MAC_{K_{CS}}(N_C, Time)$

Listing 5: LATe w/MAC of first message: N_C can be a counter

1 : $C \rightarrow S : ID_C, N_C, MAC_{K_{CS}}(ID_C, N_C)$
2 : $S \rightarrow C : Time, MAC_{K_{CS}}(ID_C, N_C, Time)$

Listing 6: LATe Synchronization Protocol v2

C. Reflection Attack

Another attack can be done if the Time Client also acts as a Time Server: on the original LATe protocol an attacker can use a message generated by the actor in the Time Server role, to be injected in another run of the protocol with the same actor acting as a Time Client. The modified version on Listing 5 does not suffer from this attack. This can also be avoided if the second message includes the recipient ID in the MAC.

D. Symmetric cryptography: server key management issues

The use of symmetric cryptography comes at a burden at the server: it has to keep a copy of all clients' keys. We assume an IoT setting where the constrained node (i.e. Time Client) has a well-known trusted party which it uses for many purposes e.g. an *Authorization Server* (AS) as defined on IETF Authentication and Authorization for Constrained Environments [20] framework. On such a setting the AS can also act as a Time Server. LATe has also the flexibility to use asymmetric crypto to relieve the Time Server key management issues if fits better the envisioned IoT use case.

E. Protocol refinement

Using the Scyther tool we verified that the same security claims from the original LATe synchronization protocol are hold true in a protocol using a more compact Message 2. By omitting the Nonce in the response, but still using it to calculate the MAC, all the security claims hold still true, but we achieve a non-negligible gain in message size. This can be done only if we assume that a client can run only one concurrent run of the protocol (i.e. when receiving a response it can assume implicitly the nonce to use to calculate the MAC), this assumption is reasonable.

To conclude this section we gather all the mitigations proposed for attack plus this optimization to propose a stronger version of the LATe Synchronization Protocol on Listing 6.

The authentication of the first message that mitigates completely pre-play attacks, can also be used to mitigate Denial-of-Service attacks at the server-side. A version that does not authenticate the first message is still useful on real environments if the users are aware of the pre-play and nonce considerations of section VI-B.

VII. Comparison of Time Sync. Protocols

To define a common baseline to compare several time synchronization protocols we do not take in account underlying layers overhead (e.g. IEEE 802.15.4), but only application data. We also simplify the encoding of the messages, assuming no overhead for metadata, and we assume the following data

TABLE III
SECURE TIME SYNCHRONIZATION PROTOCOLS BASELINE COMPARISON

Protocol	Nr. of Msg.	Avg. msg. size (Bytes)	Total Bytes	Crypto Ops. at Node
SPS [6]	2	21	41	$1 \times MAC$ $1 \times Nonce$
E-SPS [6]	3	17	50	$1 \times MAC$ $1 \times Nonce$
TinySeRSync [7]	2	21	42	$2 \times MAC$
Guo et al. [8]	3	39	116	$2 \times Signature$ $1 \times MAC$
E-SPBS [9]	3	35	104	$1 \times Signature$ $1 \times Nonce$
LATe	2	15	30	$1 \times MAC$ $1 \times Nonce$
LATe v2	2	15	30	$2 \times MAC$ $1 \times Nonce$

sizes: *Timestamp* representation is 4 bytes, *Node Identity* is 2 bytes, a *Nonce* is 8 bytes, and a *MAC* is 8 bytes. In E-SPBS [9] an ECDSA signature is 48 bytes; In Guo et al. [8] we assume an Unspecified Signature being of 16 bytes, and non-cryptographic hash 16 bytes; In [6][7] syn-ack information of 1 byte. On Table III we can se the results.

We also calculated values for *NTS Extensions for NTPv4* after Key Establishment [1]: 2 Messages; 134 bytes avg. msg. size; 268 total bytes; 2 AEAD (symmetric) operations. And for *PTP with Annex-K* after Security Association: 4 Messages; 128 bytes avg. msg. size; 512 total bytes; $4 \times MAC$. Both are one order of magnitude greater due to the calculations taking in account real applicative messages and not simplified encoding.

At the IoT constrained node *energy* is the scarcest resource and, simplifying, the total bytes to be exchanged is the most important factor to be minimized. LATe minimizes both the number of messages and the total bytes count, needing $\approx 25\%$ less application data exchange than the second-lowest Secure Pairwise Synchronization Protocol (SPS). This percentage will vary if we include other protocols' overhead, or change the application data representation estimations, however LATe will still be strictly inferior. In terms of cryptographic burden LATe is also the lightest, with one MAC operation and one nonce generation.

VIII. PERSPECTIVES

Two main topics have been highlighted on this work: the need for a lightweight secure time synchronization protocol in the context of IoT, and the use of computer-aided tools to prove the security claims of protocols. The first issue needs to be solved, WSN time solutions aim at homogeneous one-hop precise time synchronization, and the end-to-end NTP-based IETF solution is not lightweight in any way; meanwhile IoT nodes need to securely bootstrap time, and we assume real-world implementations are using home-brew solutions. This paper has shown how difficult is to prove secure even a simple protocol like LATe, and this reinforces the need for an open standardized effort that will benefit from the scrutiny of experts and the academic community.

IX. CONCLUSION

Secure time synchronization is a fundamental service for many IoT applications. Notably, security services need a secure source of time. We proposed a nonce-based lightweight authenticated time synchronization protocol, which allows a client to securely bootstrap time with a trusted server. The proposed protocol guarantees end-to-end security among heterogeneous networks. We provided a specification of the protocol using state-of-the-art IETF standards suitable for IoT. We used the computer-aided tool Scyther to prove some security claims of our protocol on a formal model.

ACKNOWLEDGMENT

The authors would like to thank Göran Selander and Ludwig Seitz for its contributions on the design of the LATe protocol and his efforts on the IETF community.

REFERENCES

[1] D. F. Franke, D. Sibold, and K. Teichel, "Network Time Security for the Network Time Protocol," IETF, Internet-Draft draft-ietf-ntp-using-nts-for-ntp-11, Mar. 2018, work in Progress.
[2] J. Burbank, W. Kasch, and P. D. L. Mills, "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC 5905, 2010.
[3] M. Lévesque and D. Tipper, "A Survey of Clock Synchronization Over Packet-Switched Networks," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 4, pp. 2926–2947, 2016.
[4] B. Moussa, M. Debbabi, and C. Assi, "Security Assessment of Time Synchronization Mechanisms for the Smart Grid," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 3, pp. 1952–1973, 2016.
[5] T. Mizrahi, "Security Requirements of Time Protocols in Packet Switched Networks," RFC 7384, 2014.
[6] S. Ganeriwal, C. Pöpper, S. Čapkun, and M. B. Srivastava, "Secure Time Synchronization in Sensor Networks," *ACM Transactions on Information and System Security*, vol. 11, no. 4, pp. 1–35, 2008.
[7] K. Sun *et al.*, "TinySeRSync: secure and resilient time synchronization in wireless sensor networks," *Proceedings of the 13th ACM conference on Computer and communications security*, p. 264, 2006.
[8] L. Guo *et al.*, "A Lightweight Secure Time Synchronization Mechanism for ISO/IEC/IEEE 21451 Sensor Networks," in *IEEE Precision Clock Synchronization for Measurement, Control, and Communication*, 2015.
[9] C. Benzaid *et al.*, "An Enhanced Secure Pairwise Broadcast Time Synchronization Protocol in Wireless Sensor Networks," *Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing*, 2014.
[10] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, vol. 3, no. 3, pp. 146–158, 1989.
[11] R. K. Bauer, T. A. Berson, and R. J. Feiertag, "A key distribution protocol using event markers," *ACM Transactions on Computer Systems*, vol. 1, no. 3, pp. 249–255, 1983.
[12] C. Bormann and P. Hoffman, "Concise Binary Object Representation (CBOR)," RFC 7049 (Proposed Standard), RFC Editor, Oct. 2013.
[13] J. Schaad, "CBOR Object Signing and Encryption (COSE)," RFC 8152, Jul. 2017.
[14] D. E. E. 3rd, S. Crocker, and J. I. Schiller, "Randomness Requirements for Security," RFC 4086, Jun. 2005.
[15] N. Koblitz and A. J. Menezes, "Another look at "provable security"," *Journal of Cryptology*, vol. 20, no. 1, pp. 3–37, 2007.
[16] C. J. F. Cremers, "The Scyther Tool: Automatic Verification of Security Protocols," *Computer Aided Verification*, vol. 5423, pp. 414–418, 2008.
[17] D. Dolev and a. C. Yao, "On the security of public key protocols," *22nd Annual Symposium on Foundations of Computer Science*, no. M, 1981.
[18] G. Lowe, "A hierarchy of authentication specifications," *Proceedings 10th Computer Security Foundations Workshop*, pp. 31–43, 1997.
[19] C. J. F. Cremers *et al.*, "Injective synchronisation: An extension of the authentication hierarchy," *Theoretical Computer Science*, no. 1-2, 2006.
[20] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)," IETF, Internet-Draft draft-ietf-ace-oauth-authz-11, Mar. 2018, work in Progress.