Secure Time Synchronization Protocol

Faten Mkacher*¶, Xavier Bestel*, and Andrzej Duda¶
*Gorgy Timing, F-38350 La Mure d'Isère, France
¶Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France
Email: {faten.mkacher@imag.fr, andrzej.duda@imag.fr}

***Abstract*—This paper describes the Secure Time Synchronization (STS) protocol that enables client and server mutual authentication, supports the property of non-repudiation, and offloads the negotiation and authorization phases to an Authorization Server (AS). We also propose a solution for bootstrapping time synchronization to solve the problem of certificate validation that depends on time. We analyze the main security properties of STS with the ProVerif tool, implement STS by extending OpenNTPD, and compare its precision to unauthenticated NTP.**

I. Introduction

The security of NTP is an important issue raised lately after the discovery of several implementation flaws [1], [2], [3]. Several proposals tried to enhance NTP security: NTPv3 with authentication based on symmetric keys and NTPv4 Autokey [4] with authentication based on public key cryptography and digital signatures. The IETF NTP Working Group develops NTS (Network Time Security) [5].

The recent ANTP protocol [6], [7], [8] supports authentication of an NTP server based on symmetric key cryptography and delayed authentication codes computed on NTP packets to avoid the impact on synchronization precision. The design of ANTP aimed at making the server stateless: clients obtain the state encrypted with a long-term symmetric key and they send the state back to the server in a subsequent NTP request. The server retrieves the symmetric session key for signing NTP packets from the encrypted state. Server authentication relies on the server certificate that clients need to verify during the negotiation phase, which requires already synchronized time. Moreover, ANTP does not authenticate NTP clients nor guarantees non-repudiation (only digital signatures based on public key cryptography can guarantee non-repudiation), the property required in some NTP deployments.

In this paper, we propose to go further in the support of NTP authentication with the Secure Time Synchronization (STS) protocol that enables client and server mutual authentication, supports the property of non-repudiation, and offloads the negotiation and authorization phases to a third party—an Authorization Server that relieves the Time Server of the setup and negotiation phases. We assume that the Authorization Server as well as the Time Server benefit from a precise time source so they do not require additional time synchronization. The Authorization Server checks for authorizations and provides the required cryptographic material to Time Clients and Time Servers over DTLS (Datagram Transport Layer Security) sessions [9]. During the time synchronization phase between a Time Client and a Time Server, time critical operations rely on symmetric cryptography and if non-repudiation property is required, STS supports fast digital signatures based on recent high-performance schemes.

We also propose a solution for bootstrapping time synchronization to solve the problem of certificate validation that depends on time. The solution builds on the timestamps in blocks of the immutable public blockchain as the basis for the approximate time, which avoids trusting revoked certificates.

We have analyzed the main security properties of STS with the ProVerif tool and implemented STS based on OpenNTPD [10]. We have evaluated the overhead of the most time-critical operations and show that the chosen cryptographic primitives for generation of authentication codes and digital signatures introduce little overhead, which contributes to the capacity of the Time Server to accommodate a large number of clients. We have also measured the precision of STS compared to unauthenticated NTP.

The rest of the paper presents the requirements (Section II), discusses related work (Section III), and describes the operation of the protocol (Section IV). We then report on the verification of its security properties (Section V), discuss the implementation, evaluate the overhead of the cryptographic operations, and provide a measurement-based comparison with NTP (Section VI). Section VII concludes the paper.

II. Requirements

The motivation for the STS protocol comes from SCPTime, a research project on the dissemination of the certified legal time [11] to use in any form of commercial transactions or applications. Achieving this goal requires an underlying time synchronization protocol that copes with many security threats to a larger extent than the requirements for NTP [12]. In particular, it needs to: i) authenticate servers and clients, ii) authorize clients to access the time service, iii) authenticate and protect integrity of the time information, and iv) enforce non-repudiation of the time information. At the same time, the security mechanisms integrated with the protocol should not degrade the precision of time synchronization, which requires some kind of a well balanced trade-off between security and performance. Moreover, servers providing the time service need to guarantee high availability, which means that their processing rate should be high enough to support clients despite possible DoS attacks. Finally, STS clients also require some support for secure time bootstrapping: as certificate

validation depends on valid time, STS clients need to start with some initial rough notion of time to validate certificates used in authentication.

A. Security Threats

We consider the threat model specified in the NTP requirements [12]. We assume that STS clients and servers are sufficiently protected against internal attacks performed by exploiting vulnerabilities in devices. We thus focus on *external* (also called in-band) attackers that may have control over the network (including access to a trusted segment of the network). In particular, they can [13]:

- intercept, modify, or remove a message sent in the network,
- record messages and replay them later,
- generate messages with any values and claiming to possess any identity they choose.

The protocol needs to consider many types of attacks. In the *Man in the Middle (MITM)* attack, the attacker can intercept packets, change and relay them to their respective destinations to alter the protocol operation. In a *replay* attack, the attacker intercepts and resends previous packets. An attacker can also attempt a *delay* attack to increase the time packets spend in transit and impact time synchronization. In the *masquerade* or *spoofing* attack, the attacker takes the identity of a legitimate client or a server. In addition to all these attacks, one or more attackers can collaborate in a *Denial of Service* (DoS) attack that attempts to make the service unavailable by overwhelming protocol entities with a high level of requests. We assume that there is no need for guaranteeing confidentiality.

III. Related Work

For the sake of brevity, we focus below on ANTP that inspired our design.

ANTP [6], [7], [8] supports authentication of NTP servers based on certificates and guarantees message integrity via MAC (Message Authentication Code) computed with a symmetric key. ANTP operates in three phases: negotiation, key exchange, and time synchronization.

In the first phase, the client and the server agree on supported cryptographic algorithms. The server sends its certificate and opaque state C_1 containing the hash computed over the client negotiation message, the certificate, and the negotiated algorithms, encrypted with long-term secret s. The client validates the server certificate and obtains its public key.

In the key exchange phase, the client uses a key encapsulation mechanism based on the server public key to establish shared session key k. The server offloads the state by replying with opaque state C_2 containing the algorithms to use in the synchronization phase and session key k.

After these phases, the client sends an NTP synchronization message along with offloaded state C_2 and a nonce to prevent replay attacks. The server retrieves session key k from C_2, responds immediately with an NTP reply message, and sends an additional message with MAC based on session key k that authenticates and guarantees the integrity of the NTP reply message. This way of operation involves little impact of the security mechanisms on the precision of time synchronization.

ANTP assumes an existing out-of-band method for validation of the server certificate, so the problem of having already synchronized time for certificate validation is not solved by the protocol. It is a vicious circle: certificate validation requires the knowledge of time and authenticated time synchronization relies on certificate validation (Mizrahi also points out the problem without suggesting a possible solution [12]). Moreover, ANTP does not authenticate clients nor guarantees non-repudiation: as MAC that authenticates and guarantees the integrity of the NTP reply message is based on shared key k, the server may refuse to admit the provision of the time information that could have caused some damage to a client.

As the protocol uses public key operations only in the first two phases, the time synchronization phase benefits from good performance. Nevertheless, clients need to renegotiate session key k periodically to keep the key fresh.

One of the ANTP design goals was to make the server stateless: it offloads the required information to the client in opaque C_1 and C_2, which contributes to a high capacity of the server to serve a large number of clients.

IV. STS Protocol

Our protocol has the following design goals:

- server and client mutual authentication,
- authentication of time synchronization messages,
- guaranteed integrity and/or non-repudiation,
- little impact on time synchronization precision,
- stateless, lightweight operation of the Time Server.

A. Architecture

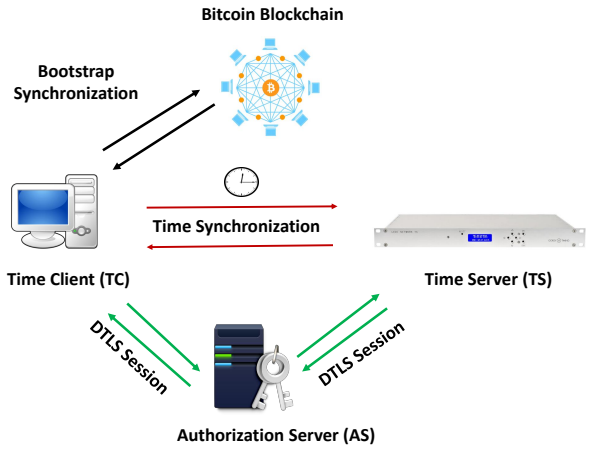


Figure 1. STS architecture

To make the time synchronization server the most efficient and lightweight possible, we propose to offload computationally heavy operations to a third party—an *Authorization Server*. Figure 1 presents the architecture of all the parties involved in the protocol: a time client (TC), a time server (TS), and an authorization server (AS). We assume that AS

and TS benefit from a precise time source so only TC needs to synchronize its time. To deal with the problem of certificate validation by clients, we propose a scheme for rough time synchronization of a client based on the Bitcoin blockchain—when the client boots and its clock is not yet synchronized, it obtains an approximate value of the current time from the blockchain timestamps.

AS takes care of managing authorizations and storing the algorithms supported by servers. TS establishes a DTLS session with mutual authentication with AS to provide the supported algorithms and obtains long-term secret S as well as a pair of public/private keys (K_e, K_d). Similarly, TC starts a DTLS session with mutual authentication with AS to obtain the algorithms to use with a given server TS, its public key K_e, and symmetric key K. To authenticate AS, TC uses the approximate time from the blockchain to validate the AS certificate.

B. Protocol Description

We present below the details of the protocol operation.

1. Bootstrap time synchronization. In this initial phase, a client obtains approximate time in a secured way so that it can validate the server certificate. The idea is to begin with rough time precision of several hours to avoid trusting revoked certificates. We use the timestamps in blocks of the immutable public blockchain as the basis for the approximate time: in fact, the Bitcoin protocol provides a distributed timestamp service on a peer-to-peer basis [14].

More specifically, we assume that initially, TC does not have a synchronized clock and it is configured with the hash of the $N-m$ block in the Bitcoin blockchain, N being the last block at the time of the configuration. A reasonable value for m is for instance 10 to be sure that the block is immutable—there are some blocks chained after block $N-m$. When TC starts to operate, it behaves like a Simplified Payment Verification (SPV) lightweight blockchain client: it discovers peers in the Bitcoin P2P network and synchronizes the blockchain headers with a peer without storing the whole blockchain. To find peers, TC can use several DNS seeds or hardcoded IP addresses proposed by the Bitcoin Core. Then, TC requests from a peer the headers starting from the $N-m$ block hash using a `GetHeaders` message of the Bitcoin protocol. A peer takes the hash of the $N-m$ block and replies with up to 2000 headers chained after block $N-m$. TC repeats the header synchronization to reach the end of the blockchain and chooses the $Last-m$ block to get the timestamp from its header, this block considered as the last immutable block. The timestamp represents the rough time.

To avoid MITM and masquerade attacks, TC can repeat the synchronization process with several peers to check whether the returned headers correspond to the Bitcoin blockchain.

2. Client/Server Setup. This phase starts with the establishment of a DTLS session between TS and AS with mutual authentication based on certificates. Using the secure channel, TS sends its supported MAC, Digital Signature (DS), and encryption algorithms to AS and obtains two parameters required for its operation: long-term secret S and public/private key pair (K_e, K_d).

After bootstrapping time synchronization, TC opens a DTLS session with AS and validates the AS certificate against the approximate time. It sends its supported MAC schemes to AS and $TSID$, the identifier of TS with which it wants to synchronize its clock (AS can also propose TS to use). It obtains the following parameters: the algorithms to use in the synchronization phase, symmetric key K, server public key K_e, and state C encrypted with long-term secret S of TS. State C encodes all the information required by TS to process client NTP requests. To provide this information to TC, AS maintains the relationships: $TSID$—$[S, (K_e, K_d)]$, and $TCID$—$[K, C]$, where $TCID$ is the client identifier.

3. Time Synchronization. In this phase, TC sends a time synchronization request along with a nonce and opaque state C to chosen TS. Then, it sends a second message with MAC computed with symmetric key K over the first message. TS decrypts symmetric key K and algorithms to use from opaque state C, and verifies MAC of the second message with key K. Then, TS sends the reply message and generates another one with MAC computed with symmetric key K over the request and reply messages. If TC requires non-repudiation, TS generates DS, a digital signature over the request and reply messages computed with TS private key K_d. TC computes the offset based on the timestamps transmitted in unauthenticated NTP messages to avoid impacting time precision and updates its clock after validating MAC or DS received in the last message. If computed RTT is greater than parameter Δ (the bound on RTT to eliminate outliers), TC rejects the message and aborts time synchronization, which prevents TC from delay attacks.

Figure 2 presents the principles of the setup and time synchronization phases of the STS protocol.

C. Discussion

Security of STS. DTLS supports mutual authentication of TS and AS as well as TC and AS. A random nonce prevents from replay attacks. MAC computed on the request message guarantees its integrity and allows the server to authenticate its sender. Similarly, MAC on the response message guarantees integrity and authenticates TS. If a digital signature (DS) is used instead of MAC, it guarantees non-repudiation. Δ, the bound on RTT prevents TC from delay attacks.

Bootstrap synchronization. The Roughtime project by Google [15] attempts to provide a similar service based on a set of time servers. A client starts with the first one, passes the response to the second one, and so on with other servers. The process results in finding the maximum of the time at different servers due to the causal relation between successive queries. Nevertheless, servers are not authenticated, so a client can suffer from masquerade attacks.

Stateless server. The server does not need to keep state per client because clients send the necessary cryptographic material (symmetric key, negotiated algorithms) in opaque

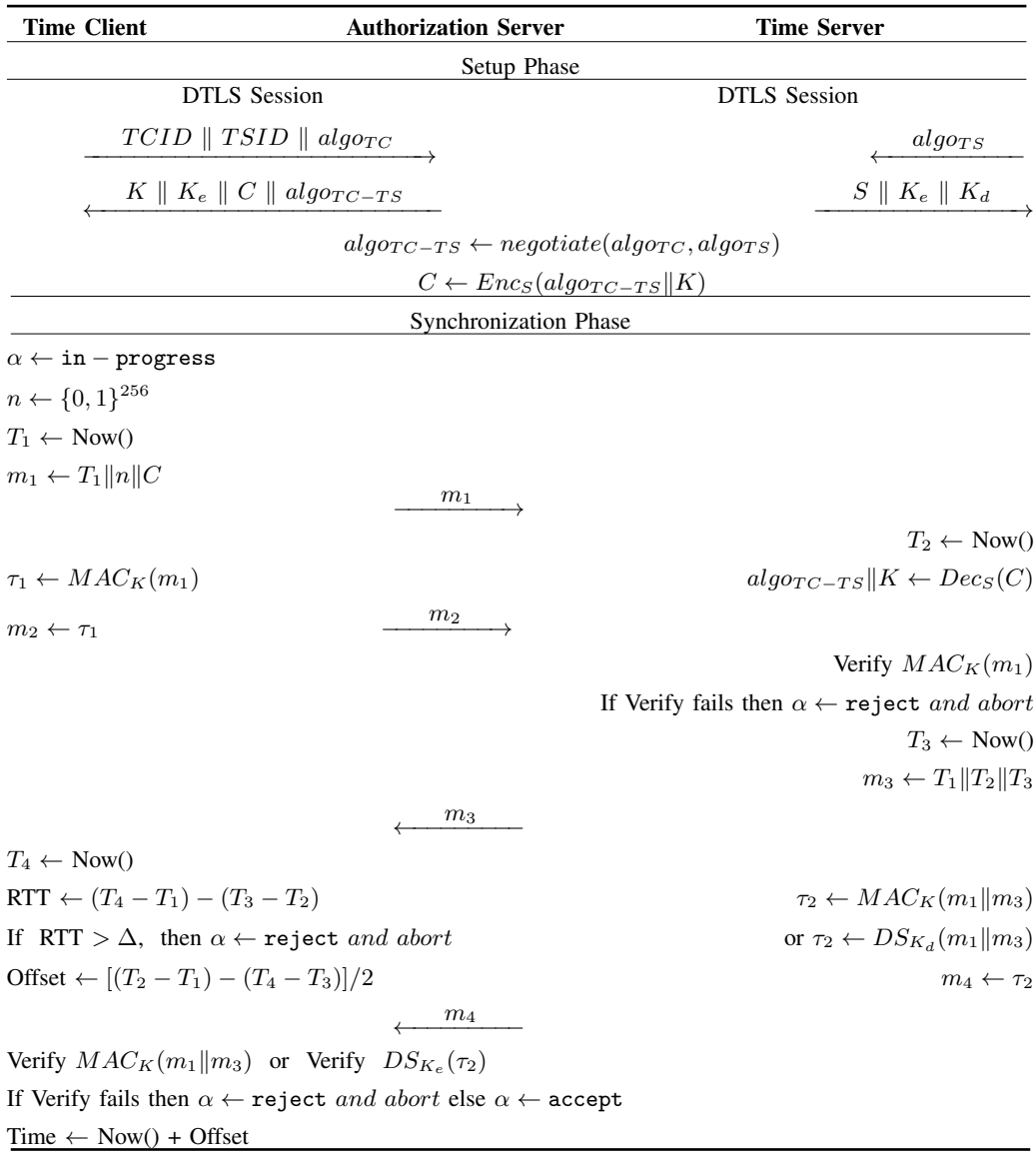


Figure 2. Principles of the STS protocol. The protocol flow assumes that TS either uses MAC or DS for signing reply messages. Notation: $Enc_S(), Dec_S()$ - encryption/decryption with symmetric key S, $MAC_K()$ - MAC code with key K, $DS_{K_d}()$ - digital signature with key K_d, α - session state ($\texttt{in-progress}, \texttt{accept}, \texttt{reject}$), Δ - bound on RTT.

value C. DTLS operates in a lightweight way compared with TLS.

Time-critical operation. As in ANTP, the entities send unauthenticated NTP messages to avoid increasing the transmission time with cryptographic operations. The overhead that may be introduced at this stage is related to the copy of the NTP reply message with the timestamp inserted by hardware just before transmission—the server can only compute MAC/DS after the insertion of the timestamp.

Key freshness. Servers and clients need to refresh the cryptographic material periodically by contacting AS. When TS decides to refresh its keys, it redoes the setup phase with AS, obtains new keys, and replies to client requests with an error indicating that they need to refresh the keys. Clients thus need to redo the setup phase with AS and proceed with the synchronization phase with new keys.

Efficient cryptography. Digital signatures may impact the performance of the time server because they take more time than MAC operations. Nevertheless, efficient schemes for signature exist such as Ed25519 [16] and MQQ-SIG [17]. Annessi et al. used this kind of high-performance signatures in the context of secure multicast time synchronization [18].

DoS. The design of TS aims at the capacity to serve a large number of clients without keeping state per client. Moreover, TS avoids acting as a DoS amplifier by never responding to a request with a packet larger than the request.

V. STS Security Verification

After the specification of the protocol, we have performed a formal analysis of STS. Since it is difficult to perform

an analysis on protocols depending on time [19], [20], we have limited its scope to the analysis of security properties: authentication, integrity protection, and secrecy of keys. A formal model checker seemed a good approach because it provides an easy and fast way to identify vulnerabilities in protocol specifications.

We have used ProVerif [21], a powerful automatic tool for proving properties on traces of protocols. ProVerif supports a wide range of cryptographic primitives defined by *rewrite rules* [22] or by equations [21]. It can prove various security properties: secrecy, authentication, and process equivalence. It takes as input a description of the protocol in a dialect of the applied Pi calculus, translates it into Horn clauses, and determines whether the desired security properties hold by resolution on these clauses [21].

As stated in Section II-A, we consider an attacker that may have control over the network, which corresponds to the Dolev-Yao model [13]. We have modeled the participant roles in STS as processes in the ProVerif input language and fed them into ProVerif to analyze the protocol and prove reachability properties, correspondence assertions, as well as observational equivalence. To test properties, we have specified *queries* for which ProVerif attempts to prove that the state in which the query does not hold is unreachable—if the query is proved, it means that there is no successful attack, otherwise ProVerif discovers an attack against the desired security property.

We have analyzed the setup phase based on a formal analysis of DTLS to which we have added our message exchange between AS, TC, and TS to be sure that the key material is protected during the DTLS session. Then, the analysis of the time synchronization phase has shown that STS achieves authentication and session key secrecy under standard assumptions on the security of cryptographic primitives. If a client considers the data from the response message from the server as authentic, then the server has indeed sent the response message with the same time data and the same nonce. If the server accepts state C from the request message as being legitimate, then the content of C is unknown to the attacker. Finally, we have verified the propriety of non-repudiation guaranteed by digital signatures based on public/private keys.

VI. STS Implementation and Performance

In this section, we describe the implementation of STS based on OpenNTPD [10] and the OpenSSL libcrypto library for cryptographic operations [23].

A. Cryptographic Primitives

We have chosen the following algorithms for cryptographic primitives:

- AES-GCM [24] for symmetric encryption the server uses to decrypt the opaque value sent by the client,
- HMAC-SHA256 [25] and AES-CMAC [26] for the MAC algorithms,
- Ed25519 [16], [27] and MQQ-SIG [17], [28] for the DS algorithms.

Table I
Execution Time of MAC Primitives

MAC Algorithms	Execution Time
HMAC-MD5 (generation)	4 μs
HMAC-SHA256 (generation)	4 μs
AES-CMAC (generation)	3 μs

Table II
Execution Time of DS Primitives [18]

DS Scheme	Execution Time
Ed25519 (generation and verification)	75 μs
MQQ-SIG (generation and verification)	28 μs

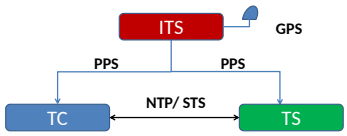


Figure 3. LAN testbed for measurements.

The choice of HMAC-SHA256 and AES-CMAC is motivated by the state of standardization and availability of their open source implementations (available in the OpenSSL library) [29].

With respect to performance, STS mostly adds MAC and DS generation and verification compared with the regular NTP operation. To evaluate their impact, we have tested different algorithms and measured the time needed for the operations.

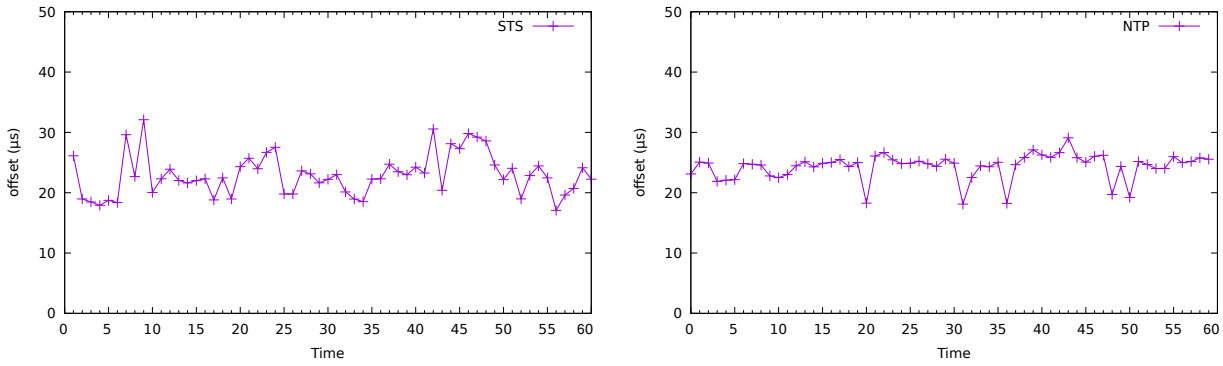
We have evaluated three types of MAC: two hash-based HMAC-SHA256 and HMAC-MD5, and a block cipher-based AES-CMAC. Table I presents the execution time of the MAC primitives computed over the longest message in the STS protocol (a client request that includes two extension fields with a nonce and opaque state C). The client ran Linux 4.13.0-39 on an Intel Core i5-6200U processor with 8GB RAM.

For digital signatures, we report the performance data on the Ed25519 and MQQ-SIG schemes measured by Annessi et al. [18]: Table II presents the execution time of the DS schemes computed over an NTP message. The choice of Ed25519 or MQQ-SIG also depends on the key size: MQQ-SIG generates smaller signatures than Ed25519 (32 B vs. 64 B), but the size of its public key is larger than that for Ed25519 (32 kB vs. 517 B) [18].

B. STS Implementation

We take advantage of the NTP extension fields for transporting the information specific to STS. The implementation extends OpenNTPD with the operation described in Figure 2.

To evaluate the STS performance in the time synchronization phase, we have measured its precision on a LAN testbed presented in Figure 3: we run two PCs as TC and TS synchronized with the Gorgy Timing LEDI Network ITS v2m (GPS time reference) as the time source over the PPS (Pulse-Per-Second) interface. TC runs STS and unauthenticated NTP



(a) STS (b) Unauthenticated NTP

Figure 4. Offset estimation precision during a 1 min. period.

for a comparison based on the time reference at TC also obtained from ITS through the PPS interface.

Figure 4 shows the precision of the offset estimation by STS and NTP over a period of 1 min. We can observe that the precision of both protocols is comparable and we cannot really distinguish the overhead of the STS operation (note that the data for two protocols are not gathered at the same time).

We have also measured the latency of client-server interactions on the LAN testbed: STS - 600 μs, NTP - 510 μs (RTT estimated with ping is 400 μs). The results show that STS only introduces a small overhead to time synchronization.

VII. Conclusion

In this paper, we have proposed the STS protocol that enables client and server mutual authentication, supports the property of non-repudiation, and offloads the negotiation and authorization phases to an authorization server. We have implemented the protocol based on OpenNTPD. In measurement experiments, we have evaluated the overhead of the chosen cryptographic primitives for generation of authentication codes and digital signatures as well as compared the precision of STS to unauthenticated NTP. The evaluation shows that the primitives introduce little overhead and STS provides precision comparable to NTP.

Acknowledgments

This work has been partially supported by the French Ministry of Research project PERSYVAL-Lab under contract ANR-11-LABX-0025-01. Useful comments by reviewers are gratefully acknowledged.

References

[1] "CVE-2014-9295: Multiple Stack-Based Buffer Overflows in NTPD in NTP before 4.2.8," National Vulnerability Database, 2014. [Online]. Available: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-9295

[2] A. Malhotra *et al.*, "The Security of NTP's Datagram Protocol," in *21st International Conference Financial Cryptography and Data Security, FC 2017, Sliema, Malta*, 2017, pp. 405–423.

[3] A. Liska, "Vulnerabilities in NTP," in *Understanding NTP*. Apress, Berkeley, CA, 2016, ch. 3.

[4] B. Haberman and D. Mills, "Network Time Protocol Version 4: Autokey Specification," 2010, RFC 5906.

[5] D. Franke, D. Sibold, and K. Teichel, "Network Time Security for the Network Time Protocol," 2017, IETF draft-ietf-ntp-using-nts-for-ntp-10.

[6] B. Dowling, D. Stebila, and G. Zaverucha, "Authenticated Network Time Synchronization," in *25th USENIX Security Symposium*, Austin, TX, 2016, pp. 823–840.

[7] G. Zaverucha, B. Dowling, and D. Stebila, "ANTP: Authenticated NTP Implementation Specification," Tech. Rep., February 2015, Microsoft Research.

[8] B. Dowling, "Provable Security of Internet Protocols," Ph.D. dissertation, Queensland University of Technology, 2017.

[9] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," 2012, RFC 6347.

[10] B. Cook. [Online]. Available: https://github.com/openntpd-portable/openntpd-portable

[11] Minalogic, "Gorgy Timing SCPTime Box." [Online]. Available: http://www.minalogic.com/en/product/box-scptime

[12] M. T. Mizrahi, "RFC 7384: Security Requirements of Time Protocols in Packet Switched Networks," October 2014.

[13] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[14] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," May 2011.

[15] Google, "Roughtime Project." [Online]. Available: https://roughtime.googlesource.com/roughtime

[16] D. J. Bernstein *et al.*, "High-Speed High-Security Signatures," *Journal of Cryptographic Engineering*, pp. 77–89, 2012.

[17] D. Gligoroski *et al.*, "MQQ-SIG," *Trusted Systems*, pp. 184–203, 2011.

[18] R. Annessi, J. Fabini, and T. Zseby, "It's about Time: Securing Broadcast Time Synchronization with Data Origin Authentication," in *26th ICCCN, Vancouver, BC, Canada*, 2017, pp. 1–11.

[19] M. Archer, "Proving Correctness of the Basic TESLA Multicast Stream Authentication Protocol with TAME," in *WITS '02 Workshop on Issues in the Theory of Security*, 2002, pp. 14–15.

[20] P. Hopcroft and G. Lowe, "Analysing a Stream Authentication Protocol Using Model Checking," *International Journal of Information Security*, vol. 3, no. 1, pp. 2–13, Oct 2004.

[21] B. Blanchet, "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif," *Foundations and Trends in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016.

[22] ProVerif. [Online]. Available: http://prosecco.gforge.inria.fr/personal/bblanche/proverif/

[23] OpenSSL. [Online]. Available: https://www.openssl.org

[24] D. A. McGrew and J. Viega, "The Security and Performance of the Galois/Counter Mode of Operation (Full Version)," *IACR Cryptology ePrint Archive*, vol. 2004, p. 193.

[25] T. Hansen and D. Eastlake, "US Secure Hash Algorithms (SHA and HMAC-SHA)," 2006, RFC 4634.

[26] J. Song, R. Poovendran, J. Lee, and T. Iwata, "The AES-CMAC Algorithm," 2006, RFC 4493.

[27] D. J. Bernstein. [Online]. Available: https://github.com/floodyberry/supercop/tree/master/crypto_sign/ed25519

[28] R. E. Jensen and D. Gligoroski. [Online]. Available: https://github.com/floodyberry/supercop/blob/master/crypto_sign/mqqsig256/ref

[29] A. Malhotra and S. Goldberg, "Message Authentication Codes for the Network Time Protocol," 2006, IETF draft-aanchal4-ntp-mac-02.