# Optimization of Time Synchronization Techniques on Computer Networks

Faten Mkacher

Université Grenoble Alpes

# THÈSE

Pour obtenir le grade de

## DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Informatique**
Arrêté ministériel : 25 mai 2016

Présentée par


## Faten MKACHER


Thèse dirigée par **Andrzej DUDA**

et coencadrée par **Fabrice GUERY**

Préparée au sein du **Laboratoire d'Informatique de Grenoble (LIG)**, dans **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique (EDMSTII)**.

# Optimization of Time Synchronization Techniques on Computer Networks

Thèse soutenue publiquement le **02 juin 2020**, devant le jury composé de :

**Noel de Palma**
Professeur, Université Grenoble Alpes, Président

**Katia Jaffrés-Runser**
Maître de conférence, Université de Toulouse, Rapporteur

**Hervé Rivano**
Professeur, Université INSA de Lyon, Rapporteur

**Andrzej Duda**
Professeur, Grenoble INP, Directeur de thèse

**Fabrice Guery**
Responsable Innovation, Gorgy Timing, Invité

## Abstract

Nowadays, as society has become more interconnected, secure and accurate time-keeping becomes more and more critical for many applications. Computing devices usually use crystal clocks with low precision for local synchronization. These low-quality clocks cause a large drift between machines. The solution to provide precise time synchronization between them is to use a reference clock having an accurate source of time and then disseminate time over a communication network to other devices. One of the protocols that provide time synchronization over packet-switched networks is Network Time Protocol (NTP). Although NTP has operated well for a general-purpose use for many years, both its security and accuracy are ill-suited for future challenges. Many security mechanisms rely on time as part of their operation. For example, before using a digital certificate, it is necessary to confirm its time validity. A machine with an inaccurate clock can accept an expired or revoked certificate.

This thesis first provides a background on time synchronization starting with the definition of some fundamental concepts such as the clock model, the problem of clock synchronization, and some notions like accuracy, precision, and stability of clocks. We study the most common time synchronization protocols used in packet-switched networks, and among others NTP.

Then, we consider the security of time synchronization by presenting the possible security threats against time synchronization protocols and the security requirements of these protocols. We zoom in on the security of the NTP protocol as described by the standard NTP and other related work that tried to enhance NTP security. We also discuss the importance of having a well-balanced trade-off between security and performance.

In our first contribution, we propose to go further in the support of NTP security with Secure Time Synchronization protocol (STS), a new secure authenticated time synchronization protocol suitable for widespread deployments. We describe the operation of STS and prove its design secure with a formal analysis using two security protocol verification tools: Proverif and Tamarin. We present the implementation of STS based on the OpenNTPd project, and evaluate its performance by comparing the STS precision with unauthenticated NTP.

We point out the circular dependency between certificate validation and time synchronization. In fact, reliable time synchronization requires cryptographic materials that are valid only over designated time intervals, but time intervals can be only enforced when participating hosts are reliably synchronized. We present a solution for bootstrapping time synchronization based on the Bitcoin blockchain to break this circular dependency.

In our second contribution, we propose a method for improving the accuracy of the NTP protocol by taking into account asymmetric transmission delays due to different bandwidth or routing on the forward and backward paths. In fact, asymmetry is quite prevalent in the Internet, which leads to low accuracy of NTP that relies on the symmetric delay assumption to compute the clock offset. This method builds on using an NTP client synchronized with GPS to measure precisely the one-way transmission delay on the forward and backward path with his time server. In this way, it is possible to calibrate NTP to take into account asymmetry.

## Keywords

time synchronization, Network Time Protocol (NTP), security, accuracy, One-Way-Delay (OWD).

## Résumé

De nos jours, alors que la société est toujours plus interconnectée, une synchronisation temporelle sûre et précise devient de plus en plus critique pour de nombreuses applications. Les dispositifs informatiques utilisent souvent des oscillateurs à cristal de faible précision pour conserver le temps en local. Cette imprécision engendre une dérive entre les machines. La solution pour assurer une synchronisation précise de l'heure entre elles est d'utiliser une horloge de référence avec une source précise de temps, puis de diffuser le temps sur le réseau. Un des protocoles qui assurent la synchronisation temporelle est *Network Time Protocol* (NTP). Bien que NTP ait bien fonctionné pour un usage général pendant de nombreuses années, sa sécurité et sa précision sont mal adaptées aux défis futurs. De nombreux mécanismes de sécurité dépendent du temps dans le cadre de leur fonctionnement. Par exemple, avant d'utiliser un certificat électronique, il est nécessaire de confirmer sa validité temporelle. Une machine avec une horloge imprécise pourrait accepter des certificats expirés ou révoqués.

Cette thèse présente d'abord le contexte de la synchronisation temporelle en commençant par la définition de certains concepts fondamentaux tels que le modèle d'horloge, le problème de la synchronisation d'horloge et certaines notions comme l'exactitude, la précision et la stabilité des horloges. Nous étudions les protocoles de synchronisation temporelle les plus courants des réseaux de communication, et entre autres NTP.

Ensuite, nous considérons la sécurité de la synchronisation temporelle en présentant les possibles menaces de sécurité contre les protocoles de synchronisation temporelle et les exigences de sécurité de ces protocoles. Nous nous concentrons sur la sécurité du protocole NTP tel que décrit par le standard, et les travaux connexes qui ont tenté de l'améliorer sur ce point. Nous discutons également de l'importance d'avoir un compromis bien équilibré entre sécurité et performance.

Dans notre première contribution, nous proposons d'aller plus loin que NTP avec *Secure Time Synchronization Protocol* (STS), un nouveau protocole de synchronisation de l'heure, qui est authentifié et sécurisé, et adapté aux larges déploiements. Nous décrivons le fonctionnement de STS et prouvons sa conception sécurisée, à l'aide d'une analyse formelle faite par deux outils de vérification de protocole de sécurité : Proverif et Tamarin. Nous présentons l'implémentation de STS basée sur le projet OpenNTPd, et évaluons ses performances en comparant la précision de STS avec celle de NTP non authentifié.

Nous soulignons la dépendance circulaire entre la validation du certificat et la synchronisation temporelle. En réalité, une synchronisation temporelle fiable nécessite des matériaux cryptographiques qui ne sont valables que sur des intervalles de temps désignés, mais ces intervalles de temps ne peuvent être comparés à l'heure actuelle que lorsque les hôtes participants sont synchronisés de manière fiable. Nous présentons une solution qui fournit, lors de l'amorçage, une synchronisation approximative basée sur le *blockchain* Bitcoin, pour rompre cette dépendance circulaire.

Dans notre deuxième contribution, nous proposons une méthode pour améliorer l'exactitude du protocole NTP, en tenant compte des délais de transmission asymétriques dus à une bande passante ou à un routage différent sur le chemin d'aller et de retour. En fait, l'asymétrie est assez répandue sur Internet, ce qui dégrade la performance de NTP qui fait l'hypothèse de délais symétriques. Cette méthode s'appuie sur l'utilisation d'un client NTP synchronisé par GPS, pour mesurer le délai unidirectionnel minimal aller et retour jusqu'à son serveur de temps. Ainsi, il est possible de calibrer NTP en prenant en compte cette asymétrie.

## Mots-clefs

synchronisation temporelle, Network Time Protocol (NTP), sécurité, exactitude, précision, délai unidirectionel.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

**ADSL** Asymmetric Digital Subscriber Line. 50
**AEAD** Authenticated Encryption with Associated Data. 59
**AS** Autonomous System. 49, 50
**AS** Authorization Server. 70

**BC** Boundary Clock. 43
**BGP** Border Gateway Protocol. 49
**BMC** Best Master Clock algorithm. 44

**CERN** European Organization for Nuclear Research. 18, 33, 47
**CPS** Cyber-Physical Sytems. 18

**DC** Direct Current. 29
**DDMTD** Digital Dual-Mixer Time Difference. 47
**DDoS** Distributed Denial of Service. 18, 54
**DoS** Denial Of Service. 56
**DTLS** Datagram Transport Layer Security. 70, 72

**EEC** Ethernet Equipment Clock. 46

**FLL** Frequency Locked Loop. 48, 49

**GNSS** Global Navigation Satellite System. 18
**GPS** Global Positioning System. 22

**HSTS** HTTP Strict Transport Security. 54
**HTTPS** Secured Hypertext Transfer Protocol. 54

**IPPM** IP Performance Metrics. 97
**IPsec** Internet Protocol Security. 61
**ITU** International Telecommunication Union. 46
**ITU-T** International Telecommunication Union–Telecommunication Standardization Sector. 31

**JET** Joint European Torus. 18

**LATe** A Lightweight Authentication Time Synchronization Protocol. 63, 64
**LHC** Large Hadron Collider. 18

**M2M** Machine-to-Machine. 17
**MAC** Message Authentication Code. 57, 58
**MCXO** Microcomputer-compensated Crystal Oscillators. 30

**MD5** Message Digest 5. 57
**MiFID II** Markets in Financial Instruments Directive. 17
**MiTM** Man-in-the-Middle. 55, 58
**MTIE** Maximum Time Interval Error. 31

**NIST** National Institute of Standards and Technology. 18
**NTP** Network Time Protocol. 4, 18, 20–23, 33–36, 40, 48–50, 54–57, 69
**NTS** Network Time Security. 57, 59

**OC** Ordinary Clock. 42
**OCXO** Oven-controlled oscillator. 30
**OWD** One-Way Delay. 98

**PLL** Phase Locked Loop. 46–49
**PON** Passive Optical Network. 45
**PRC** Primary Reference Clock. 46
**PTP** Precise Time Protocol IEEE1588. 33, 40, 47, 49–51

**QCO** Quartz Crystal Oscillators. 30

**RADclock** Robust Absolute Clock and Difference Clock. 48, 49
**RFC** Requests For Comments. 35
**RTT** Round Trip Time. 18, 49, 97

**SDH** Synchronous Digital Hierarchy. 45
**SNTP** Simple Network Time Protocol. 33, 35
**SONET** Synchronous Optical Network. 45
**SPF** Shortest Path First. 50
**SSH** Secure Shell. 61
**SSU** Synchronization Supply Unit. 46
**STS** Secure Time Synchronization protocol. 4, 21–23, 69
**SyncE** Synchronous Ethernet. 33, 45–47
**SYRTE** Sytèmes de Référence Temps Espace. 19

**TAACCS** Time Aware Applications, Computers and Communications Systems. 18
**TC** Transparent Clock. 43
**TC** Time Client. 63, 64
**TCXO** Temperature-compensated Crystal Oscillators. 30
**TDEV** Time Deviation. 31
**TLS** Transport Layer Security. 59, 61
**TS** Time Server. 64
**TSN** Time Sensitive Networking. 17

**UTC** Coordinated Universal Time. 17, 19, 20

**VCXO** Voltage-Controlled crystal Oscillator. 30

**WLAN** Wireless Local Area Network. 51
**WR** White Rabbit. 33, 47, 51

# Foreword

## Acknowledgments

*This thesis could not have been completed without the support, help and encouragement of many people that have been by my side during this experience.*

*I express my deep gratitude to the company GORGY Timing for having hosted me in its R&D team SCPTime, especially to Maurice Gorgy, Nicolas Gorgy, Fabrice Guery, Jacques Thimonier and Eric Mistelet for the support they provided during my research. The time spent with them has strengthened and shaped my personality. It has been an honor and an absolute pleasure to work with them.*

*I am indebted to Andrzej Duda for the enthusiasm he transmitted, for his competence, as well as for the richness of his guidelines and his suggestions. The time he spent with me in discussing and reviewing my work was extremely precious. Our discussions helped me to enhance the accuracy and clarity of my research and stimulated my curiosity for the formal and analytical aspects of my research.*

*I would like to thank all members of the jury for having accepted to review my thesis and for the interest they demonstrated in respect to my work.*

*I am grateful to my colleagues and friends that have shared with me the journey of my thesis from the beginning.*

*Thanks also to all members of LIG laboratory who have contributed in many ways to the fun and comfort of my stay.*

*My family deserves a special gratitude for the patience and the attention that I enjoyed: it was a priceless assistance throughout this period. I am indebted to my dear husband Beyram and my lovely son Iyad for having supported me in the most critical moments encountered during this thesis.*

*This Thesis is dedicated to my parents for their endless love, support and encouragement.*

# Introduction

*"It's really clear that the most precious resource we all have is time."*

Steve Jobs

## Context

Synchronizing clocks has been a long-standing important problem. Accurate clocks enable applications to operate on a common time basis across different nodes, which in turn enables key functions like consistency, event ordering, causality, and the scheduling of tasks and resources with precise timing.

Nowadays, as society has become more interconnected, accurate time-keeping is more and more critical for the future development and improvements of several applications: On the finance sector, new regulations require precise time synchronization of business clocks in trading systems like the new EU legislation called Markets in Financial Instruments Directive (MiFID II) (MiFID II) [10] that requires a range from 1 microsecond to 1 second of timestamping granularity and a range from 100 microsecond to 1 second of maximal divergence from the Coordinated Universal Time (UTC) depending on the type of trading activity. We find similar requirements set by the US Securities and Exchange Commission.

Industry is another field where a common notion of time is necessary to maintain the production process. For example, smart power grid systems based on two-way flow of energy and Machine-to-Machine (M2M) communications need to share synchronized information to improve the efficiency and reliability of power delivery [11]. Another example is Time Sensitive Networking (TSN), a set of IEEE 802 Ethernet sub-standards that enable deterministic real-time communication over Ethernet [12] by using time synchronization and a schedule shared between network components.

Even today's scientific applications require a very precise time such as Large Hadron Collider (LHC) in European Organization for Nuclear Research (CERN) or Joint European Torus (JET) nuclear fusion reactor.

Some research groups like Time Aware Applications, Computers and Communications Systems (TAACCS), a sub-group led by National Institute of Standards and Technology (NIST), study the challenges of timing and synchronization in Cyber-Physical Sytems (CPS): systems co-engineered to interact with physical, computational, and communications components in many fields: emerging smart cars, intelligent buildings, robots, unmanned vehicles, and medical devices [13].

A solution to provide time synchronization among networked devices requiring time alignment is to incorporate an accurate source of time in each device (e.g., atomic clock or a Global Navigation Satellite System (GNSS) receiver). However, this solution is not ubiquitous because it is expensive and not practical in many scenarios.

Given the wide deployment of computing devices using simple crystal clocks, the solution is to use a reference clock having an accurate source of time and then disseminate its time over a communication networks to other devices. One of the protocols that provide time synchronization over packet-switched networks is NTP. In its client/server mode, it allows clients to synchronize their clocks to NTP servers through the exchange of packet timestamps transported in NTP packets.

NTP accuracy depends on network delays and is around a few milliseconds. However, this accuracy depends on the assumption about symmetric delays during client-server communication. As NTP cannot directly measure one-way transmission times of its synchronization packets for clock offset calculation, it relies on the symmetric delay assumption to estimate the one-way transmission time as half of the Round Trip Time (RTT). However, this is a poor assumption because asymmetry is quite prevalent in the Internet, which will lead to synchronization errors in NTP.

Over the past couple of years, NTP has been in the news a number of times after the discovery of some security holes in the protocol that can be exploited to initiate Distributed Denial of Service (DDoS) attacks [14]. The developers of NTP have responded quickly with fixes or recommendations for remediating this kind of attacks. However, their efforts have not translated to an improved security for NTP.

The consequences of poorly secured NTP are even affecting the security of many other protocols and services that rely on time as part of their operation. For example, validating a certificate requires confirming that the current time is within the certificate validity period. Validation with an inaccurate clock may cause expired or revoked certificates to be accepted as valid.

Although NTP has operated well for a general-purpose use for many years, both its security

and accuracy are ill-suited for future challenges. In this thesis, we propose some improvements to enhance both its security and accuracy.

## Motivations of the Work


Figure 1 – SCPTime partners

The motivation of this work comes from SCPTime [15], a collaborative project led by Gorgy Timing company [16] that gathers French experts in Time/Frequency and industrial partners (see Figure 1).

SCPTime aims at disseminating **S**ecure, **C**ertified, **P**recise, and **T**raceable legal time from national observatories and metrology labs to final users all over the world.

SCPTime disseminates the country legal time defined based on the UTC(OP), which is the real time realization of UTC generated at the Sytèmes de Référence Temps Espace (SYRTE), Observatory of Paris. After adding 1 or 2 hours depending on summer or winter time, we get the legal time disseminated in France.

UTC(OP) is maintained to a few nanoseconds close to UTC, and it is considered as one of the best UTC(k). Its operation is based on the output signal of a hydrogen maser frequency daily steered thanks to the calibrations provided by the SYRTE atomic fountains, using a microphase stepper. In fact, SYRTE is the pioneering laboratory in the development of atomic fountains. These atomic fountains take full benefit of atom laser cooling techniques, which enables a gain in performances by several orders of magnitude compared to conventional clocks based on thermal beams.

SCPTime sets a whole infrastructure for the production, distribution, dissemination, and ac-

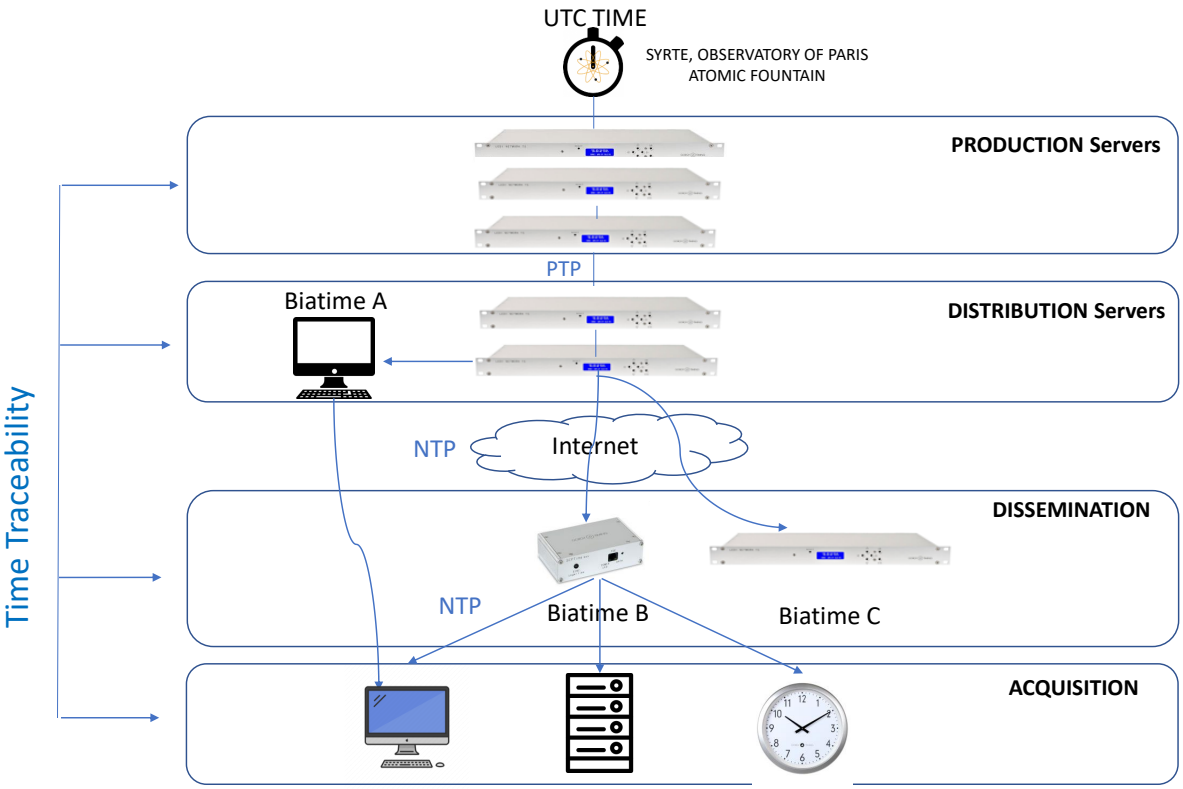



Figure 2 – SCPTime Architecture

quisition of time as depicted in Figure 2. The production servers are located close to the source of legal time and ensure permanent and secure time production thanks to redundant servers. The distribution servers cover various areas, they receive time information sent by production servers and disseminate time to dissemination servers. In the dissemination layer, SCPTime proposes 3 types of products: Biatime A, Biatime B, and Biatime C. Biatime A is an SCPTime Agent that can be downloaded on computers. Biatime B is a time dissemination system that can easily be installed at the heart of small IT systems, offices, or networks. SCPTime also proposes Biatime C for applications requiring high accuracy . SCPTime offers time traceability by deploying a system that tracks time messages from the UTC legal time to final users and delivers a certificate as a proof of the security, the accuracy, and the traceability of the disseminated time.

As depicted in Figure 2, SCPTime mainly uses NTP over Internet to disseminate the legal time of a country to final users. So, achieving the desired security and accuracy goals requires:

- a new secure variant of NTP that copes with many security threats to a larger extent than the requirements for standard NTP. In particular, SCPTime requires a time protocol that authenticate servers and clients, authorize clients to access the time service, authenticate and protect integrity, and enforce non-repudiation of time information. Besides, servers providing the time service need to guarantee high availability, which means that their processing rate should be high enough to support a large number of clients. At the same time, the

security mechanisms integrated with the protocol should not degrade the quality of time synchronization, which requires some kind of a well balanced trade-off between security and performance.

- a new method to improve NTP accuracy. In fact, NTP accuracy and precision depend on the validity of the assumption related to symmetric transmission delays between the client and the server. If this assumption does not hold, which is a common case in the current Internet, the NTP synchronization scheme results in significant errors that degrade its accuracy.

These SCPTime requirements inspired our research work in this thesis. Besides, the lack of robust security in the NTP protocol led us to propose a new secure variant. During our research, we noticed that some equipment does not have accurate time during bootstrap. The investigation led us to study the issue of circular dependency between time synchronization and authentication mechanisms, this issue inspired us to propose a new use case of the Bitcoin blockchain to get rough time synchronization.

As mentioned previously, the dissemination of time in SCPTime is mainly done with NTP over Internet and the measurements of NTP accuracy show that asymmetry can significantly degrade the accuracy of NTP. This issue prompted us to propose a method to improve NTP accuracy when delays are asymmetric.

## Main Contributions

This thesis presents two main contributions:

1. we have studied the existing efforts to secure the NTP protocol. We proposed a new secure variant of the NTP protocol that we called STS. Its design goals address the requirements of the SCPTime project. We have specified its operation and proved its security with a formal analysis using two model checker tools: Proverif and Tamarin. The formal analysis helped us to correct some details of STS design. The final version of STS was implemented based on OpenNTPd. We compared the performance of STS to standard NTP. We also dealt with the problem of certificate validation in the case of unsynchronized clients by using a scheme to obtain rough time synchronization based on the Bitcoin blockchain [17].

2. We propose a method of improving the accuracy of NTP time synchronization by taking into account asymmetric transmission delays due to different bandwidth or routing conditions. The method consists of estimating precisely the one-way transmission delays on the forward and backward path and finding the minimal delays that we use to calibrate the estimation of the clock offset at the client side. Unlike many works that validate their proposed schemes to

improve the accuracy of time protocols with simulations, we performed real measurement experiments to compare the clock offsets computed by standard NTP and calibrated NTP based on the Global Positioning System (GPS) time reference.

## Structure of the thesis

This document is primarily split into two parts. The first part describes the state of the art on time synchronization and its security. The first chapter of this part explains the fundamental concepts of time, clocks, and synchronization then it presents the main time synchronization protocols used in packet-switched networks. The second chapter discusses the security of the NTP protocol and presents the related work. Chapter 3 presents the circular dependency between security and time synchronization.

The second part of this thesis contains the main contributions. Chapter 4 presents our proposal of STS, its formal analysis, its implementation evaluation, and a solution to bootstrap rough time based on the Bitcoin blockchain. Chapter 5 presents our proposed method to improve the accuracy of NTP.

In particular, the content of the thesis is as follows:

**Part I, Chapter 1, Time Synchronization Concepts.** This chapter provides a background on time synchronization. We start by defining some concepts of time synchronization like the clock model, the problem of clock synchronization, and some notions like accuracy, precision, and stability of clocks. We present the most common time synchronization protocols used in packet-switched networks. We also point out the impact of delay asymmetry that exists in current networks on the time synchronization accuracy. Finally, we present related work that tried to deal with the asymmetry issue.

**Part I, Chapter 2, Security of Time Synchronization.** This second chapter zooms in on the security of the NTP protocol by presenting the possible security threats against time synchronization protocols in general and the security requirement for these time protocols. Then, we describe the history of the security of the standard NTP and other related work that proposed new secure variants of NTP. Finally, we briefly discuss the importance of having a well-balanced trade-off between security and performance.

**Part I, Chapter 3, Circular Dependency between Security and Time Synchronization.** This chapter points out the circular dependency between authentication and time synchronization. In fact, reliable time synchronization requires cryptographic materials that are valid only over designated time intervals, but time intervals can be only enforced when participating

servers and clients are reliably synchronized. Then, we describe some lightweight protocols proposed to provide rough time synchronization to break the circular dependency mentioned before.

**Part II, Chapter 4, STS: Secure Time Synchronization Protocol.** In the first chapter of the contributions, we propose STS , a new secure authenticated time synchronization protocol suitable for widespread deployments. First, we describe the operation of STS. Second, we prove our design secure with a formal analysis using security protocol verification tools: Proverif [18] and Tamarin [19]. Third, we present the implementation of STS by extending OpenNTPd [20], and evaluate its performance by comparing the STS precision with unauthenticated NTP. This chapter also presents our solution for bootstrapping time synchronization based on the Bitcoin blockchain to solve the problem of the circular dependency of time synchronization and public key authentication.

**Part II, Chapter 5, Calibrating NTP.** This chapter presents our second contribution, a method of improving the accuracy of the NTP protocol by taking into account asymmetric transmission delays due to different bandwidth or routing on the forward and backward paths. These asymmetric paths usually degrade the accuracy of NTP because NTP assumes that paths are symmetric and relies on this symmetric link assumption to estimate the one-way transmission time as half of Round Trip Time (RTT). However, this is a naive assumption because Internet traffic is often routed asymmetrically.

We conclude this work with a conclusion. It summarizes the contributions from the individual chapters and provides future research directions.

# Part I

# Time Synchronization: State of the Art

# Chapter 1

# Time Synchronization Concept

*"The only reason for time is so that everything doesn't happen at once."*

Albert Einstein

## Contents

## 1.1 Fundamental Concepts of Time Synchronization

### 1.1.1 Time and Clocks

We denote true time, measured in seconds from some origin, by $t$. Any real world clock inevitably tends to drift and suffers from an error or offset $\theta$ given by:

$$\theta(t) = C(t) - t \tag{1.1}$$

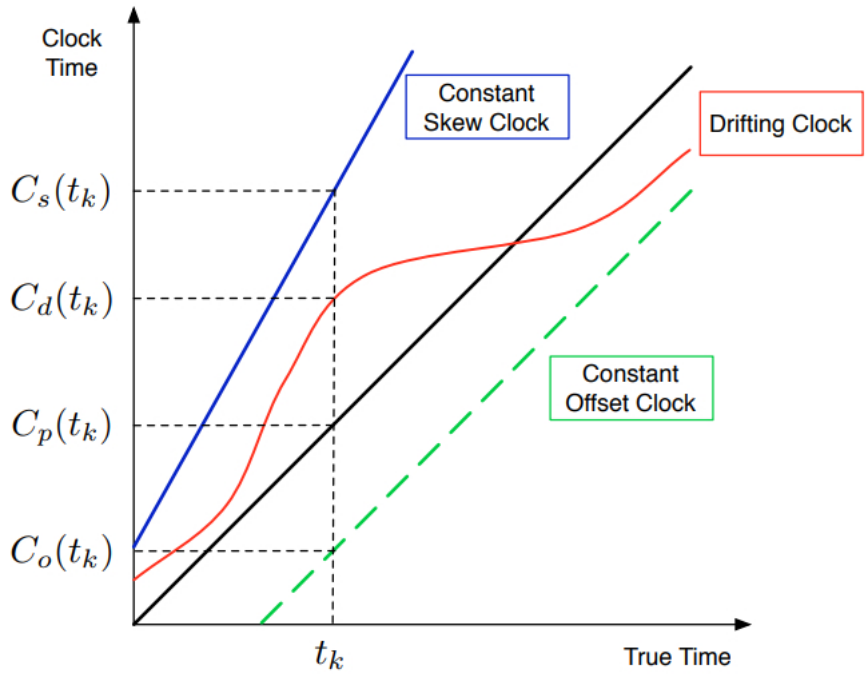



Figure 1.1 – Clock readings as a function of true time for four clocks. The realistic case is $C_d(t)$ with non-linear drift[1]

Figure 1.1 shows what 4 different clocks read, as a function of true time, for example at time $t = t_k$. The diagonal line corresponds to a perfect clock, represented by $C(t) = t$ with a clock drift (rate) $\frac{dC(t)}{dT} = 1$ and an offset $\theta = 0$. Clock $C_s(t)$ shows a clock that runs at a wrong rate that we call *skew* or frequency offset. It corresponds to parameter $\gamma$ in the model called *Simple Skew Model* that assumes that

$$\theta(t) = C(t) - \gamma t \tag{1.2}$$

This model can be valid over small time scales. Clock $C_o(t)$ has zero skew but constant offset. Over longer timescales, clock $C_d(t)$ shows a model for real clocks where the offset is a non-linear function and cannot be described by a simple skew. In this case, clock $C_d(t)$ has the following clock model:

$$C_d(t) = \theta_0 + \gamma t + \Delta\frac{t^2}{2} + \epsilon(t) \tag{1.3}$$

where:

$\theta_0$ is the initial offset

$\gamma$ is the frequency offset or skew

$\Delta$ is the frequency drift

$\epsilon(t)$ is a a stochastic or random part

An imperfect clock has a given bounded drift $\Delta$ that satisfies $|\frac{dC(t)}{dT} - 1| \leqslant \Delta$

The drift is strongly influenced by the temperature in real clocks. In fact, the temperature fluctuations can affect the oscillator and the precision of the clock. The stability of a clock is a measure of the variability of the drift and is usually measured by means of the Allan Variance.

### 1.1.2 Clocks and Oscillators

#### 1.1.2.1 Clocks

A clock consists essentially of a pulse counter and an oscillator. The oscillator (e.g., a quartz crystal, or an atomic resonator) issues periodic pulses that constitute the input to the pulse counter. The oscillator frequency $f$ is the inverse of $T$, the time interval between pulses, $f = \frac{1}{T}$. The output of the counter represents the clock $C(t)$.

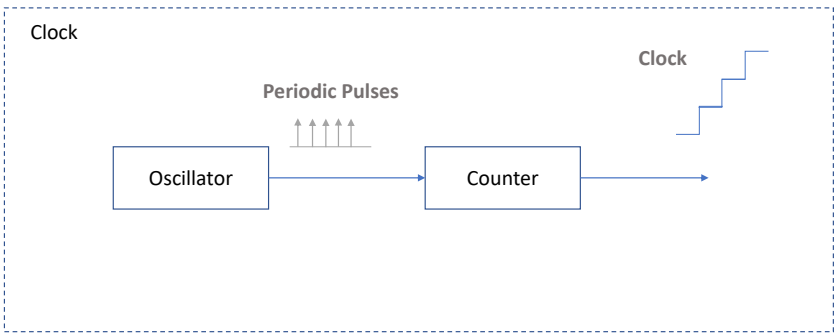



Figure 1.2 – Oscillator and counter are the components of a clock device

#### 1.1.2.2 Oscillators

An oscillator is a circuit that generates a continuous, repeated, alternating waveform without any input. Oscillators basically convert a unidirectional current flow from a Direct Current (DC) source

into an alternating waveform of the desired frequency, as decided by its circuit components.

There are many types of oscillators, the main ones are:

**Crystal oscillators ("XO")** The most commonly found in computer clocks are Quartz Crystal Oscillators (QCO). Quartz crystals are attractive because they are inexpensive, small, use little power, and perform surprisingly well despite their low resource requirements. Many factors can impact the QCO stability: the cut of the crystal, the quality and purity of it, the temperature, humidity, even radiation. Shocks can also change the frequency permanently. Over time, the frequency of crystals will also drift due to aging. To improve the performance of the quartz oscillator, a temperature compensation circuit is used to limit the variations in the output frequency that result from variations in the operating temperature. Crystal oscillators with such compensation are referred to as Temperature-compensated Crystal Oscillators (TCXO). We find also Microcomputer-compensated Crystal Oscillators (MCXO).

An Oven-controlled oscillator (OCXO) generates heat to keep the system constantly at the particular temperature at which it exhibits the greatest frequency stability. The generator is more stable, but uses much more energy, requires longer startup times, and costs more.

**Atomic oscillators** atomic oscillators are based on the observation of atomic properties of elements such as cesium or rubidium. We can use different atoms but the basic principle is the same: a Voltage-Controlled crystal Oscillator (VCXO) is locked to a highly-stable frequency reference generated by a microwave transition in the atom of interest. The stability and environmental insensitivity of this atomic reference frequency is thereby transferred to the VCXO. Although atoms can absorb and emit electromagnetic energy at many different frequencies, the hyperfine transitions are selected because they are highly stable, are relatively insensitive to environmental effects, and occur in a reasonably convenient region of the spectrum.

The following Figure 1.3 gives some characteristics of different types of oscillators.

### 1.1.3 Accuracy, Precision, and Stability

The terms accuracy, precision, and stability are often used in describing the quality of an oscillator. We propose to define each term and illustrate the difference between them (see Figure 1.4 and 1.5):

**Accuracy** is the closeness of the agreement between the result of a measurement and a true value of the measurand.

| | Quartz Oscillators | | | Atomic Oscillators | | |
|---|---|---|---|---|---|---|
| | TCXO | MCXO | OCXO | Rubidium | RbXO | Cesium |
| Accuracy (per year) | $2 \times 10^{-6}$ | $5 \times 10^{-8}$ | $1 \times 10^{-8}$ | $5 \times 10^{-10}$ | $7 \times 10^{-10}$ | $2 \times 10^{-11}$ |
| Aging/Year | $5 \times 10^{-7}$ | $2 \times 10^{-8}$ | $5 \times 10^{-9}$ | $2 \times 10^{-10}$ | $2 \times 10^{-10}$ | 0 |
| Temperature Stability (range, °C) | $5 \times 10^{-7}$ (-55 to +85) | $3 \times 10^{-8}$ (-55 to +85) | $1 \times 10^{-9}$ (-55 to +85) | $3 \times 10^{-10}$ (-55 to +68) | $5 \times 10^{-10}$ (-55 to +85) | $2 \times 10^{-11}$ (-28 to +65) |
| Stability, $\sigma_y(\tau)$ ($\tau$=1s) | $1 \times 10^{-9}$ | $3 \times 10^{-10}$ | $1 \times 10^{-12}$ | $3 \times 10^{-12}$ | $5 \times 10^{-12}$ | $5 \times 10^{-11}$ |

Figure 1.3 – Comparison of Quartz and Atomic Oscillators[2]

**Precision** is the random uncertainty of a measured value, expressed by the standard deviation or by a multiple of the standard deviation.

**Stability** it represents the change over a given observation time interval. It is expressed with some statistical dispersion metrics as a function of the observation interval (e.g., Time Deviation (TDEV), Maximum Time Interval Error (MTIE)).

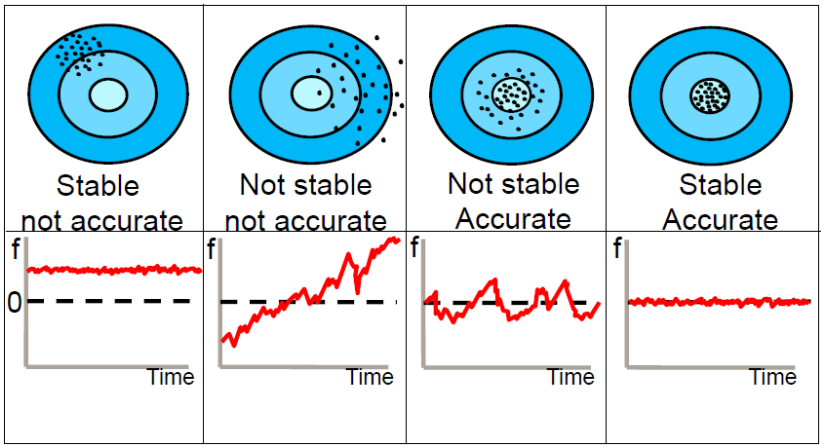



Figure 1.4 – Accuracy vs Stability [3]

### 1.1.4 Time, Frequency, and Phase Synchronization

Considering physically distinct systems, it exists three fundamental types of synchronization, based on the time, frequency, and phase. They are characterized as follow as described in International Telecommunication Union–Telecommunication Standardization Sector (ITU-T) G 8260 [21] (see Figure 1.6)

**Time Synchronization** Time synchronization refers to the distribution of an absolute time reference to a set of real-time clocks. The synchronized nodes share a common epoch and

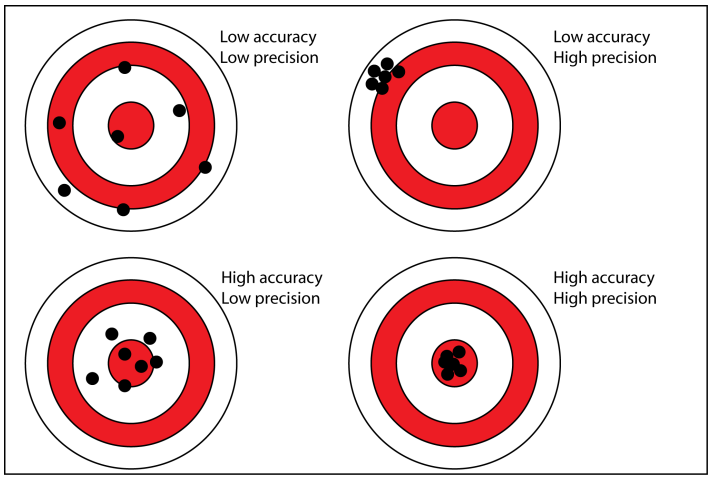



Figure 1.5 – Accuracy vs Precision [3]

time-scale.

**Frequency Synchronization** In frequency synchronized systems, the significant instants occur at the same rate for all synchronized nodes.

**Phase Synchronization** In systems synchronized based on the phase, the timing signals occur at the same instant.

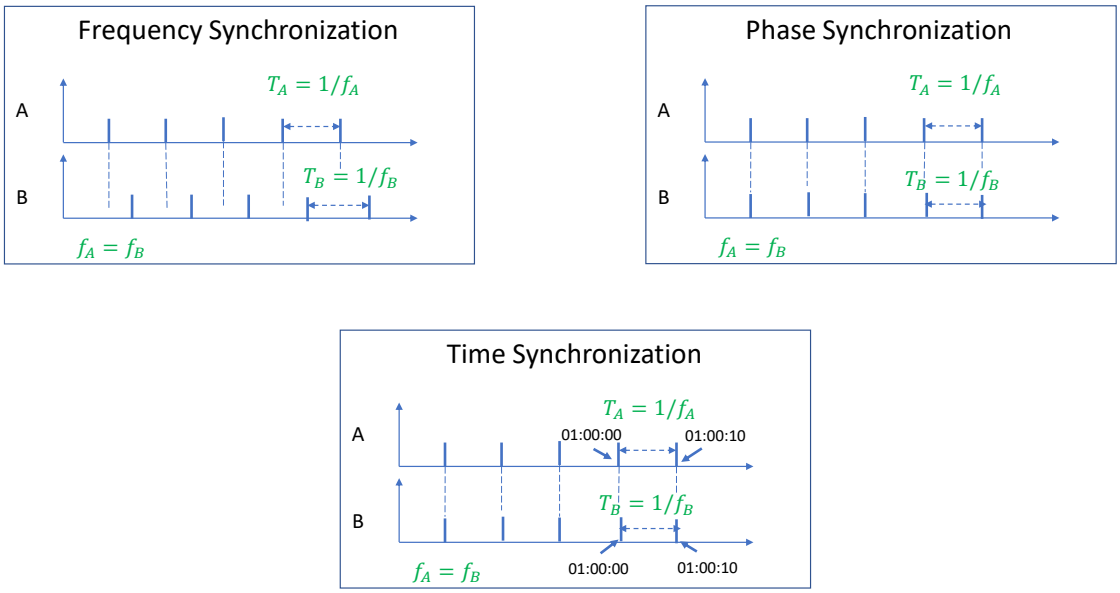



Figure 1.6 – Three fundamental types of synchronization

Some clock synchronization applications require only frequency synchronization like circuit based communication (telephone time-division multiplexing). Other clock synchronization applications require time synchronization. Packet-based communication channels, lacking a continuous frequency source, must use time synchronization even if the application needs only fre-

quency synchronization.

### 1.1.5 Synchronization and Syntonization

**Synchronization** It refers to the time difference between two clocks within some level of uncertainty after accounting for all of the necessary corrections. Two clocks are said to be synchronized if the time difference between them, after accounting for some level of uncertainty, is less than the required value.

**Syntonization** It refers to the frequency difference between two oscillators within some level of uncertainty after accounting for all of the necessary corrections. Two oscillators are said to be syntonized if the frequency difference between them, after accounting for some level of uncertainty, is less than the required value.

## 1.2 Protocols for Time Synchronization

The first public network time transfer protocols were published in 1983: **time** [22] and **daytime** protocol [23]. Those protocols provided limited accuracy and did not try to compensate the network delay. NTP was proposed in 1985 and significantly improved the time synchronization accuracy.

At the beginning, the reference implementation of NTP was designed to run under Unix-like systems. Windows systems rely on dedicated packets of the NETBIOS protocol (later extended and renamed to NETBEUI protocol) for synchronization. However, this kind of time synchronization had some limitations. So, current Windows systems also use Simple Network Time Protocol (SNTP) provided by the Windows Time service (w32time) by default.

In 2002, Precise Time Protocol IEEE1588 (PTP) has been introduced to improve accuracy beyond the level of accuracy provided by NTP. However, PTP requires special hardware support to yield the highest level of accuracy.

In order to meet sub-nanosecond accuracy, White Rabbit (WR) was developed by CERN and other scientific laboratories to be used for the CERN accelerators. WR is a combination of PTP and Synchronous Ethernet (SyncE).

The following sections give more details and characteristics about these protocols.

### 1.2.1 Network Time Protocol

#### 1.2.1.1 NTP Time Synchronization

Fig. 1.7 presents the principle of NTP time synchronization in client/server mode. We adopt the standard NTP assumptions: the server has a perfect clock $C_s = t$ and the client wants to synchro-

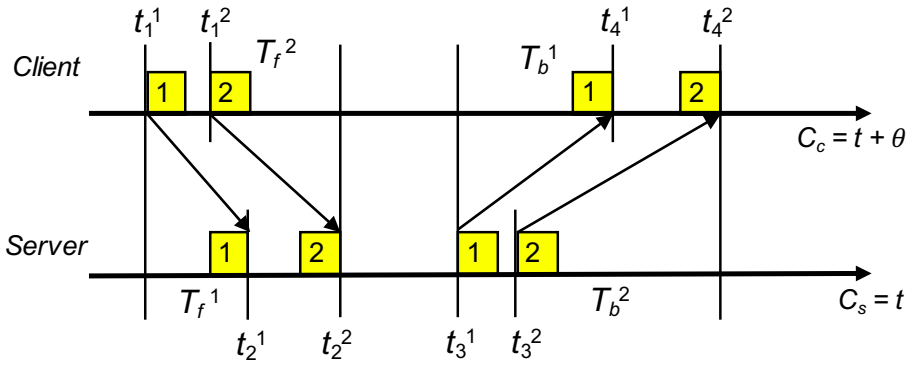



Figure 1.7 – Principles of NTP

nize its clock $C_c = t + \theta$ with the server, $\theta$ being the time offset between the client and the server.

Time synchronization relies on the two-way packet exchange. In general, the client can send $n$ NTP requests to the server that responds with $n$ responses ($n = 2$ in Figure 1.7).

The instants of sending and receiving packets are recorded and included in NTP packets: with packet $j$, the client learns timestamps $t_2^j, t_3^j$ and knows timestamps $t_1^j, t_4^j$. Note that the client records timestamps according to its clock $C_c$.

We denote forward (from the client to the server) and backward (from the server to the client) one-way transmission times: $T_f^j$ and $T_b^j$. If we assume that the clock drift during the exchange is constant, we have the following relations:

$$t_2^j = t_1^j + T_f^j - \theta^j, j = 1, ..., n, \tag{1.4}$$

$$t_4^j = t_3^j + T_b^j + \theta^j, j = 1, ..., n. \tag{1.5}$$

If one-way transmission times are symmetric ($T_f^j = T_b^j$), the time offset becomes:

$$\theta^j = \frac{(t_2^j - t_1^j) + (t_3^j - t_4^j)}{2}, j = 1, ..., n. \tag{1.6}$$

NTP uses this equation to compute the time offset. In general, one-way transmission times are asymmetric ($T_f^j \neq T_b^j$) and vary in time ($T_f^j \neq T_f^{j+1}$). In this case, the time offset becomes:

$$\theta^j = \frac{(t_2^j - t_1^j) + (t_3^j - t_4^j)}{2} + \frac{T_b^j - T_f^j}{2}, j = 1, ..., n \tag{1.7}$$

We can observe that if one-way transmission times are symmetric ($T_b^j = T_f^j$), the expression is the same as Eq. 1.6.

Eq. 1.7 shows that the accuracy of NTP time synchronization depends on the difference of one-way transmission times so the assumption of symmetric one-way transmission times is the main

source of accuracy errors. If we can estimate one-way transmission times in a more precise way, we can improve the accuracy of time synchronization.

Note that most NTP implementations send several NTP requests and compute round trip network delay $\delta^j$ as follows:

$$\delta^j = (t_4^j - t_1^j) - (t_3^j - t_2^j), j = 1, ..., n \tag{1.8}$$

They can then choose the best set $j$ of timestamps for computing the time offset $\theta^j$ based on the smallest $\delta^j$.

#### 1.2.1.2 NTP Development

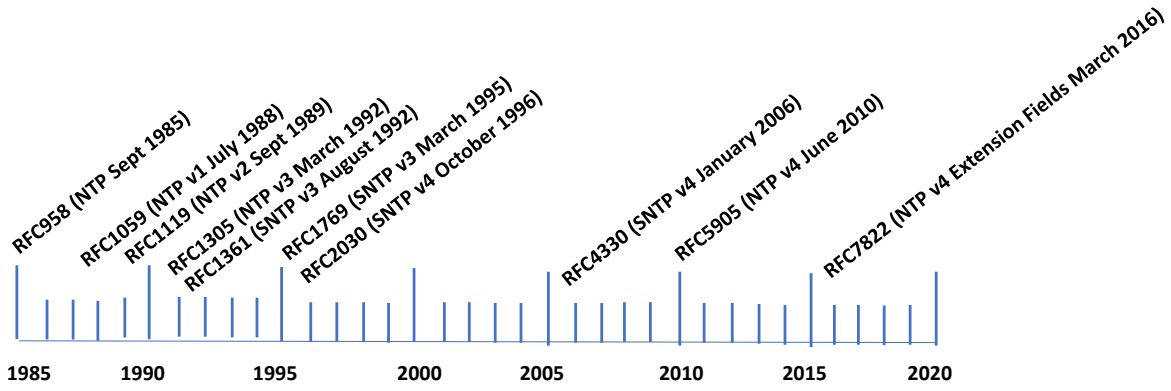



Figure 1.8 – Timeline of NTP Development

NTP is one of the most widely used protocols for clock synchronization developed by David L. Mills in 1985 at the University of Delaware. Figure 1.8 describes the timeline of the NTP development. As depicted in this figure, the protocol and related algorithms have been specified in several Requests For Comments (RFC). Its current version 4 (NTPv4) has been described in RFC 5905 [24]. In NTPv4, the basic format of the network packets is compatible with earlier NTP versions, so the current NTP implementations can be used together with older versions, unless specific NTPv4 features are being used.

There is a simplified version called SNTP [25], intended for servers and clients that do not require the degree of accuracy that NTP provides. Because the network packet format of SNTP and NTP are identical, the two protocols are interoperable. The main difference between those protocols is that SNTP client only queries a single server when requesting time synchronization and does not implement the sophisticated algorithms provided by NTP.

#### 1.2.1.3 NTP Algorithms

Besides the basic algorithm of computing the offset and estimating the delay, NTP also relies on some sophisticated techniques to improve the precision of the protocol. These techniques include:

- **Clock Filter Algorithm** processes the offset and delay samples produced by each peer process separately. It uses a sliding window of eight samples and picks out the sample with the smallest delay, which generally represents the most accurate data.
- **Clock Select Algorithm** distinguishes servers that provide correct time called "truechimers" from servers that provide incorrect time called "falsetickers" that should be discarded. Truechimers are servers which offset is located in some interval called intersection interval. Servers that are beyond this intersection interval are falsetickers.
- **Clock Cluster Algorithm** processes the truechimers produced by the clock select algorithm to produce a list of survivors. These survivors are used by the mitigation algorithms to discipline the system clock.
- **Clock Combine Algorithm** uses the survivor list to produce a weighted average of both offset and jitter. The combined offset is used to discipline the system clock, while the combined jitter is augmented with other components to produce the system jitter statistic.
- **Clock Discipline Algorithm** it is an hybrid phase/frequency-lock (NHPFL) feedback loop, used to compute the clock adjustment with an optimum averaging interval depending on prevailing network jitter and oscillator wander.

#### 1.2.1.4 NTP Operations

NTP was designed to operate over packet-switched networks and uses UDP (port 123) to send and receive timestamps. NTP uses a hierarchical system of time sources. For each level, NTP assigns a number called stratum. The Stratum represents the synchronization distance of a clock to the reference clock. Figure 1.9 presents the NTP stratum hierarchy.

- Stratum 0: atomic, GPS or radio clock.
- Stratum 1: primary time servers attached to Stratum 0 clocks and provide time services via NTP for Stratum 2 clocks.
- Stratum 2: servers that request time from stratum 1 servers. Stratum 2 servers also peer with other Stratum 2 clocks to further increase the stability of the time information. Stratum 2 clocks act as servers for Stratum 3 clocks.

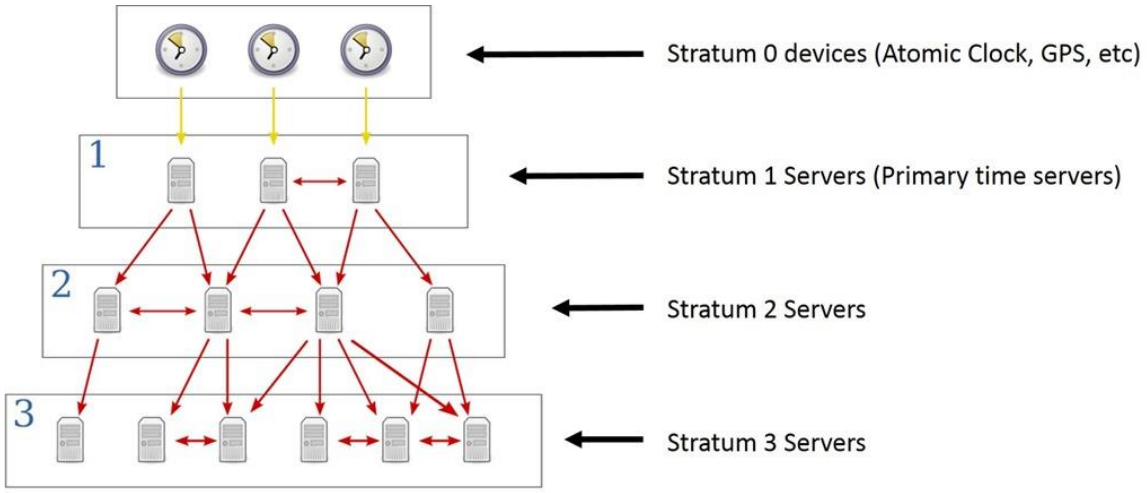



Figure 1.9 – NTP stratum hierarchy [4]

- Stratum 3: stratum 3 clocks behave like Stratum 2 clocks and provide time for stratum 4 clocks.

#### 1.2.1.5 NTP modes

NTP can operate in three different modes: symmetric, client/server, and broadcast. Each mode has an association mode, which is a description of the relationship between two NTP speakers. NTP associations have two types : persistent and ephemeral. Persistent associations are mobilized upon startup and are never demobilized. Ephemeral associations are mobilized upon the arrival of a packet and are demobilized upon error or timeout. Table 1.1 gives the values of possible NTP association modes.

| Value | Meaning |
|---|---|
| 0 | reserved |
| 1 | symmetric active |
| 2 | symmetric passive |
| 3 | client |
| 4 | server |
| 5 | broadcast |
| 6 | NTP control message |
| 7 | reserved for private use |

Table 1.1 – NTP Association Modes

**Symmetric active.** The host is willing to synchronize and be synchronized. The symmetric active association sends symmetric active packets (mode 1) to another symmetric active association.

**Symmetric passive.** If the symmetric active packet arrives with no matching association, an ephemeral symmetric passive association is mobilized and sends symmetric passive (mode 2) packets and persists until error or timeout.

**Client.** A client sends mode 4 packets to a server. In the NTP terminology, we say it pulls synchronization from a server.

**Server.** The server receives mode 4 packets from clients and responds with mode 3 packets.

**Broadcast.** The broadcast server association sends periodic broadcast server (mode 5) packets that can be received by multiple clients.

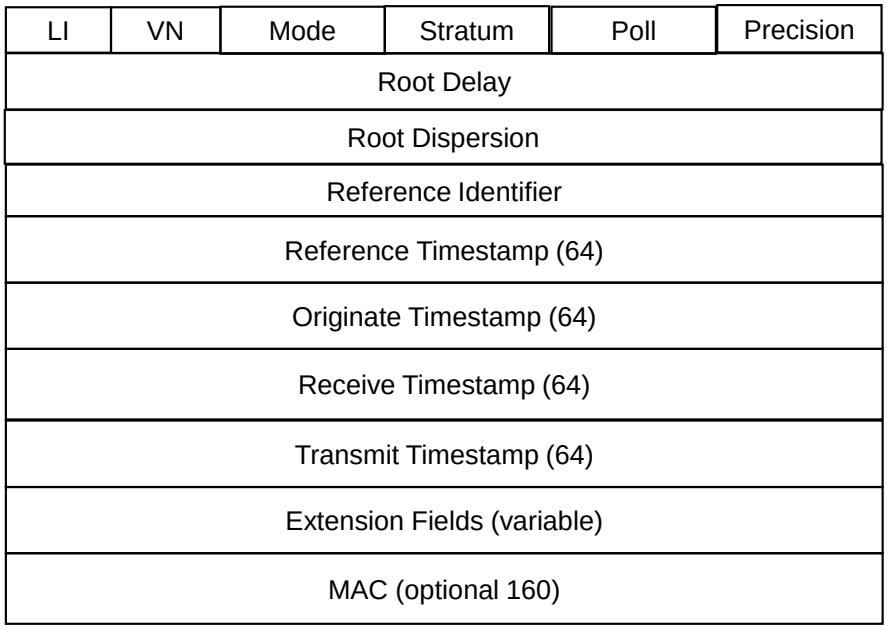



Figure 1.10 – NTP packet header format

Figure 1.10 shows NTP packet header that contains the following fields:

- **Leap Indicator (LI):** 2-bit integer used to warn of an impending leap second to insert or delete in the last minute of the month of June or December.
- **Version (VN):** 3-bit integer used to give the NTP version, currently, it is version 4.
- **Mode:** 3-bit integer that gives the mode of the association, with values defined in Table 1.1.
- **Stratum:** 8-bit integer that gives the stratum number, possible values are defined in Table 1.2.

| Value | Meaning |
|---|---|
| 0 | unspecified or invalid |
| 1 | primary server (e.g., equipped with a GPS receiver) |
| 2-15 | secondary server (via NTP) |
| 16 | unsynchronized |
| 17-255 | reserved |

Table 1.2 – NTP packet stratum

- **Poll:** 8-bit signed integer that represents the maximum interval between successive messages, in log2 seconds. The default values are 6 and 10 for minimum and maximum poll intervals, respectively.
- **Precision:** 8-bit signed integer that gives the precision of the clock, in log2 seconds. For example, a value of -18 corresponds to a precision of about one microsecond.
- **Root delay:** this field gives the total round trip delay to the primary reference source, in seconds.
- **Root dispersion:** this field gives the maximum error relative to the primary reference source, in seconds
- **Reference identifier:** 32-bit code that identifies the reference clock. It is mainly used to detect and avoid synchronization loops.
- **Reference Timestamp:** time when the system clock was last set or corrected.
- **Originate Timestamp ($t_1$):** time at the client when the request departed for the server.
- **Receive Timestamp ($t_2$:)** time at the server when the request arrived from the client.
- **Transmit Timestamp ($t_3$):** time at the server when the response left for the client.
- **Extension Fields:** the Extension fields can be used to add optional capabilities or additional information that is not conveyed in the standard NTP header.
- **MAC:** the Message Authentication Code consists of the key Identifier followed by the message digest computed over the whole NTP packet.

#### 1.2.1.6 Efforts to Improve NTP Accuracy

NTP is certainly not suitable for all applications where high timing accuracy is required, especially, when it operates over asymmetric operating conditions. Much related work tried to improve the accuracy of NTP using different methods. Schmid et al. [26] used the temperature-compensated

method to improve the accuracy. Johannessen et al. [27] used the data-filtering method to improve the NTP accuracy. Li et al. [28] proposed an improved DF-NTP, a double filter NTP algorithm to deal with uncertain delays. Quan et al. [29] established a drift compensating synchronization model to improve accuracy. Zhu et al. [30] proposed a clock offset correcting scheme for the structural characteristics of a ring network. Zhang et al. [31] adapted the path weighted feedback method to correct the ring network time.

Another idea to improve the accuracy of NTP inside a datacenter was proposed in the HUYGENS project [32]. It proposes a software clock synchronization system that achieves clock synchronization to an accuracy of 10s of nanoseconds. The important feature of HUYGENS is that it processes the transmit and receive timestamps of probed packets exchanged by a pair of clocks in bulk and simultaneously from multiple servers. The purpose of setting a synchronization network of probes between multiple servers is to identify the probes that encounter no queuing delays and no noise on the path because they naturally convey the most accurate one-way delays. To automatically identify such probes, HUYGENS introduced *coded probes*, a pair of probe packets going from server $i$ to $j$ with a small inter-probe transmission time spacing of $s$. If the spacing between the probe pair when they are received at server $j$ is very close to $s$, they are considered as "pure" and are kept both. Else, they are rejected.

Next, using the purified data, HUYGENS takes advantage of Support Vector Machines, a widely-used and powerful classifier in supervised learning to accurately estimate both the "instantaneous time offset" ($\theta(t)$) between a pair of clocks and their "relative frequency offset" ($\gamma$).

Finally, HUYGENS exploits a natural network effect: the idea that a group of pair-wise synchronized clocks must be transitively synchronized to detect and correct synchronization errors even further.

### 1.2.2 Precision Time Protocol (IEEE-1588)

Many applications require high accuracy that NTP cannot provide. For those applications, PTP, also known as the IEEE-1588 standard, was developed. Its first version PTPv1 was released in 2002 but it is not widely deployed. The second PTP version denoted as PTPv2 was released in 2008, and it is totally different from the first version and widely used. The PTP version 2.1 will be released soon and will be compatible with PTPv2.

#### 1.2.2.1 PTP Time Synchronization

A typical PTP packet exchange is shown in Figure 1.11. At the beginning, each host in the PTP master state periodically broadcasts the *Announce* message to give the sender clock quality description and to inform that it currently operates in the master mode. The synchronization is

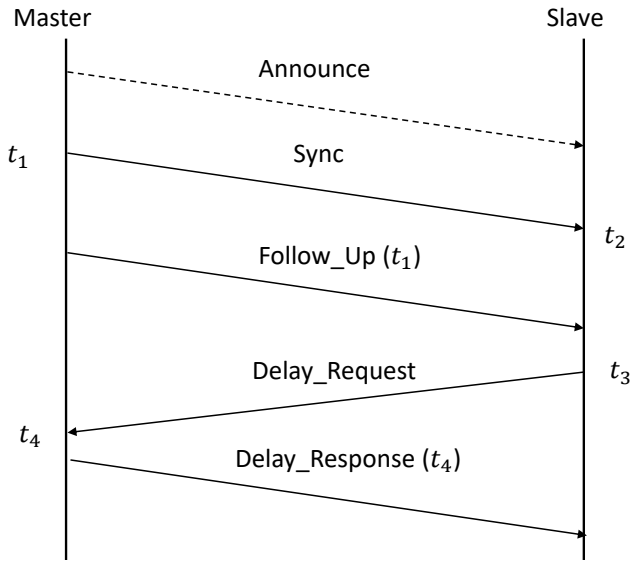



Figure 1.11 – PTPv2 messages exchange process

performed based on four hardware-generated timestamps ($t_1$,$t_2$,$t_3$, $t_4$) associated with *Sync* and *Delay Req* messages. However, to calculate delay and offset values, $t_1$ and $t_4$ have to be sent to the slave node. There are two possible variations of the PTPv2 protocol: *one-step* and *two-step*. The former incorporates $t_1$ inside *Sync* message, while the latter carries it inside a separate packet (*Follow Up*) sent just after *Sync* message. $t_4$ timestamp is always sent in *Delay_Response* message.

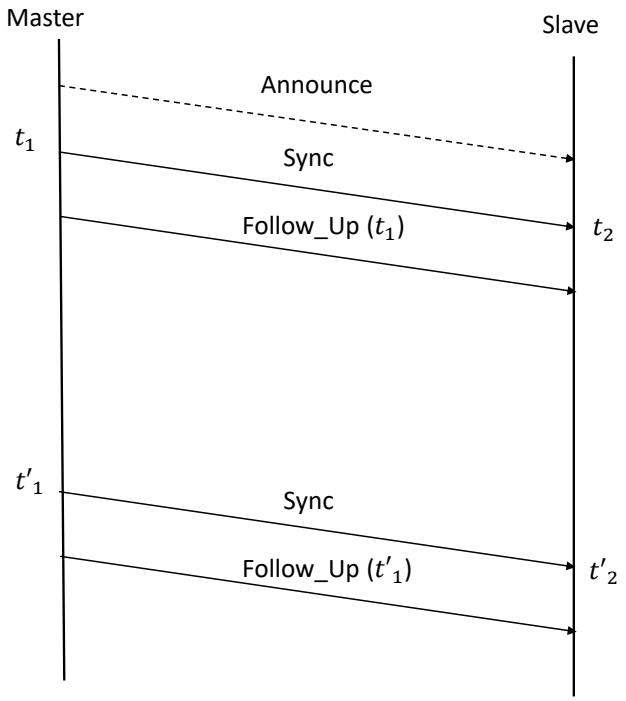



Figure 1.12 – Frequency Adjustment in PTP

The PTP protocol adjusts the frequency. As depicted in Figure 1.12, the master repeats the sending of *Sync* messages and after two *Sync* messages, the slave can compute the frequency difference to its master, which is called drift $\Delta$. The calculation is as follows:

$$\Delta = \frac{(t_2' - t_2) - ((t_1' - t_1))}{(t_1' - t_1)} \tag{1.9}$$

The drift can then be adjusted to align the frequency of the slave with the one of the master.

After obtaining the four timestamps, the slave node computes the delay and offset to correct its local clock. There are two modes to measure the delay: End-to-End (E2E) and Peer-to-Peer (P2P). The E2E delay mechanism measures the delay from the slave to the master. The P2P delay mechanism measures the delay between two nodes only independent of their states.

Equations 1.10 and 1.11 are used to compute the E2E delay $\delta$ and the offset $\theta$ that will be used for adjusting the phase of the slave.

$$\delta = \frac{(t_4 - t_1) - (t_3 - t_2)}{2} \tag{1.10}$$

$$\theta = (t_2 - t_1) - \delta \tag{1.11}$$

The delay equation assumes that the time it takes for messages to go from the master to the slave is the same as the time it takes for messages to go from the slave to the master and equal half of the round-trip delay. So, like NTP, any difference in the forward and backward delay results in an error in determining the difference between the master clock and the slave clock.

#### 1.2.2.2 PTP Clocks

The PTP clocks are categorized into ordinary clocks, boundary clocks, and transparent clocks. The master clock and the slave clock are known as Ordinary Clock (OC) that have a single network interface. The boundary clock and transparent clock have multiple network interfaces: one of them acts as the slave clock and the other acts as the master. The following list explains these clocks in detail (see Figure 1.13):

**Master clock** It gets its time from a primary reference source, typically a GPS satellite signal and transmits the messages to the PTP clients. This allows the clients to establish their offset from the master clock which is the reference point for synchronization. The root timing source is called the Grandmaster.

**Slave clock** Located in the PTP client, also called slave node, the slave clock performs clock and time recovery operations based on the received and requested timestamps from the master

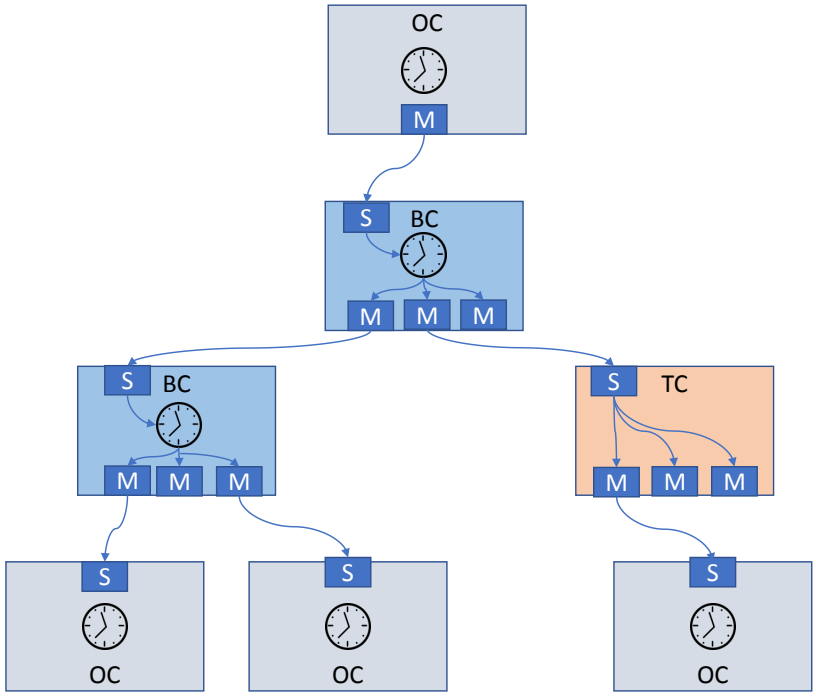



Figure 1.13 – PTP tree-structured synchronization network

clock.

**Boundary Clock (BC)** It operates as a combination of the master and slave clocks. The boundary clock endpoint acts as a slave clock to the master clock, and also acts as the master to all the slaves reporting to the boundary endpoint.

**Transparent Clock (TC)** It is used by switches or routers to assist clocks in measuring and adjusting for packet delay. It computes the variable delay as the PTP packets pass through the switch or the router. In other terms, it measures "the residence time", the time taken for PTP message to transit the device, and provides this information to clocks receiving this PTP message.

#### 1.2.2.3 PTP Messages

PTP distinguishes between two types of messages: event and general PTP messages.

**Event messages** generate and transport timestamps needed for the synchronization. The PTP event messages are: *Sync* and *Delay_Request*.

**General messages** used to measure the link delay between two clocks. The PTP general messages are: *Announce, Follow Up*, and *Delay_Response*.

We are going to explain in details the use of each message:

- *Annonce*: A general message used in establishing the synchronization hierarchy. It distributes information about the grandmaster clock time source.
- *Sync*: An event message sent by the master clock and used to communicate master clock time information to downstream clocks.
- *Delay_Request*: An event message sent by the slave and used to determine the end-to-end path delay between a master clock and a slave clock.
- *Delay_Response*: A general message sent by the master clock used to determine the end-to-end path delay between a master clock and a slave clock.
- *Management*: Management messages are used to query and update PTP data sets (attributes) and to generate specific events.
- *Signaling*: Signaling messages carry information, requests, and commands between clocks. Example: signaling messages may be used for negotiation of the rate of unicast messages between a master clock and slave clocks.

#### 1.2.2.4 PTP Algorithms

The PTP protocol consists of two main layers:

1. **Grandmaster election:** the Best Master Clock algorithm (BMC), it is a continuously active election executed by all master-candidate PTP nodes. Each candidate node sends an *Announce* message in which it declares its data sets (e.g., the stratum number, clock variance, distance from a grandmaster clock). These informations about the quality of clocks are used by the Best Master Clock algorithm (BMC) algorithm to choose a leader (a Grandmaster) that synchronizes the rest of PTP nodes.
2. **Time synchronization:** It is based on exchanging timestamped packets and estimating the delay and offset as explained before.

#### 1.2.2.5 PTP improvements

The main source of the inaccuracy in time protocols is the software generation of timestamps that causes random latencies that can not be neither estimated nor compensated. So, hardware-based timestamps used by PTP eliminate the inaccuracy caused by an operating system and result in much better link latency estimation.

To compensate the receive delay which is the delay between the moment when a packet comes in from the wire until it arrives at the application, packets can be timestamped when they come in

from the wire. This is done by a timestamp unit (TSU) that identifies incoming PTP packets in the bit stream from the wire, and takes a timestamp if such packet is detected. Both the network card driver and the application have to provide an API call to let the application retrieve that timestamp from the NIC driver and assign it to the associated network packet.

To compensate the delay for outgoing packet which is the delay between the moment when the packet is sent by the application until it really goes onto the wire, PTP relies on the "follow-up" packet that contains the timestamp of the previous packet. The receiver then gets the original packet timestamped when coming in, plus the follow-up packet that contains the transmission delay and can thus account for both delays.

The introduction of Transparent Clocks also improved the accuracy of PTP, because they can measure the residence time in switches or routers and can then provide this information to clocks receiving the PTP packet to adjust the packet delay.

#### 1.2.2.6 PTP Limits

PTP can achieve tens of nanoseconds accuracy in a local area network. However, achieving this accuracy does not only depend on the capacity of endpoints to provide a way for hardware timestamping but it also depends on the propagation delay across intermediate nodes like switches and routers which also depends on the type of switch, the network load, and the queue depth. All these factors can degrade the accuracy of PTP. That is why PTP is considered as a suitable solution only for synchronizing devices inside a local network with switches aware of PTP packets and that handle them in a special way. Besides, PTP assumes that the underlying networking hardware runs asynchronously, so the syntonization of the slave oscillator is performed by periodically sending Sync messages. This frequent exchange can generate significant network traffic that some applications do not accept.

Like NTP, PTP assumes that the transmission delays are symmetrical. This poor assumption can degrade the accuracy of the PTP protocol.

### 1.2.3 Synchronous Ethernet

Ethernet started as a LAN technology used in the vast majority of customer premises and service provider installations. Now, Ethernet is being used in base station backhaul and aggregation networks. Many access network technologies such as Passive Optical Network (PON) require synchronization, this is also the case for cellular mobile networks that require their base stations to be synchronized. However, Ethernet was not designed to transport synchronization. The solution was Synchronous Ethernet (SyncE), a traditional Ethernet plus an embedded synchronization system similar to that already used in Synchronous Digital Hierarchy (SDH)/Synchronous Optical

Network (SONET). SyncE has been standardized by the International Telecommunication Union (ITU) in cooperation with IEEE, in ITU-T G.8262 [33]. It provides mechanisms to transfer frequency over the Ethernet physical layer, which can then be made traceable to an external source such as a network clock. Three recommendations have been published:

1. ITU-T Rec. G.8261 [33] that defines frequency synchronization aspects in packet networks. It specifies the maximum network limits of jitter and wander that shall not be exceeded.

2. ITU-T Rec. G.8262 [34] that specifies an equipment clock for SyncE.

3. ITU-T Rec. G.8264 [35] that describes the specification of a synchronization signaling channel or ESMC (Ethernet Synchronization Messaging Channel).

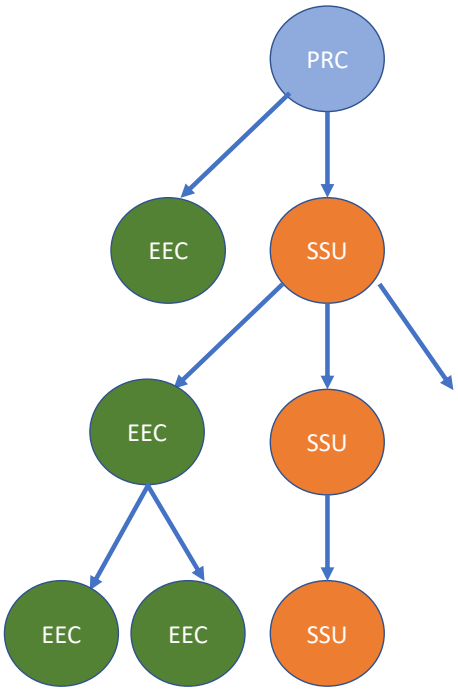



Figure 1.14 – Synchronization Network Model for Synchronous Ethernet [5]

SyncE imposes a hierarchical network structure (see Figure 1.14) with the highest accuracy clock at the top called Primary Reference Clock (PRC). PRC is the primary clock, such as an atomic clock or a GPS receiver. At the next level of hierarchy is the Synchronization Supply Unit (SSU) that uses Phase Locked Loop (PLL) to recover the reference clock from the incoming data. The recovered clock is used to encode the data streams propagated to nodes being lower in the system hierarchy. The clock supporting synchronous Ethernet networks is called synchronous Ethernet Equipment Clock (EEC).

As depicted in Figure 1.15, in SyncE, all internal clocks should be synchonized by a unique timing signal. The new Ethernet cards have an internal clock that is being synchronized by an external signal and transports the time signal.

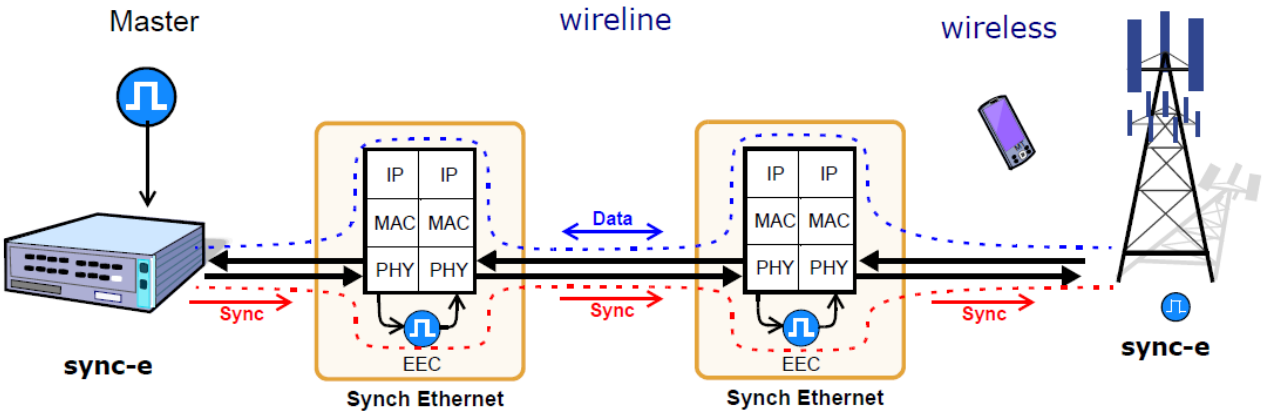



Figure 1.15 – Synchronization Ethernet exchange [5]

### 1.2.4 White Rabbit

WR [36] was developed by CERN [37] and other scientific laboratories to be used for the CERN accelerators. WR is a synchronization technology combining the PTP protocol with two further improvements: precise knowledge of the link delay and clock syntonization over the physical layer.

WR uses SyncE to distribute a common notion of frequency in the entire network over the physical medium (syntonization). WR casts the problem of timestamping into a phase detection measurement using Digital Dual-Mixer Time Difference (DDMTD) (see Figures 1.16 and 1.17). The results of these precise measurements are used both during normal PTP operation and for quantifying physical link asymmetry during the calibration phase.

The improvements proposed by WR are:

1. *WR-PTP.* The number of PTP messages in a WR network is reduced, reducing the PTP-related throughput and allowing more bandwidth for mission-critical-data exchange,

2. *SyncE.* Typical PTP implementations use free-running oscillators in each node which causes time drifts between master and slaves. This issue is solved by SyncE that enables network nodes to beat at exactly the same rate. The WR switch uses the clock recovered by the data link to sample the incoming data then it uses an embedded PLL-based oscillator for transmission.

3. *Precise Phase Measurement.* The accumulation of phase noise can degrade the time synchronization performance. To this end, WR switch is equipped with a phase measurement

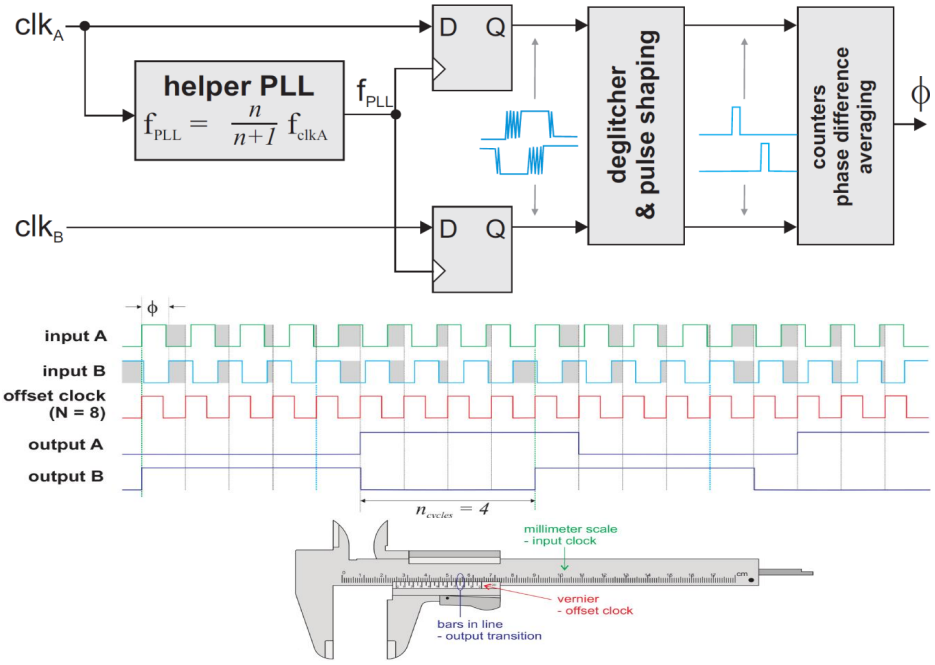



Figure 1.16 – Digital Dual-Mixer Time Difference [6]

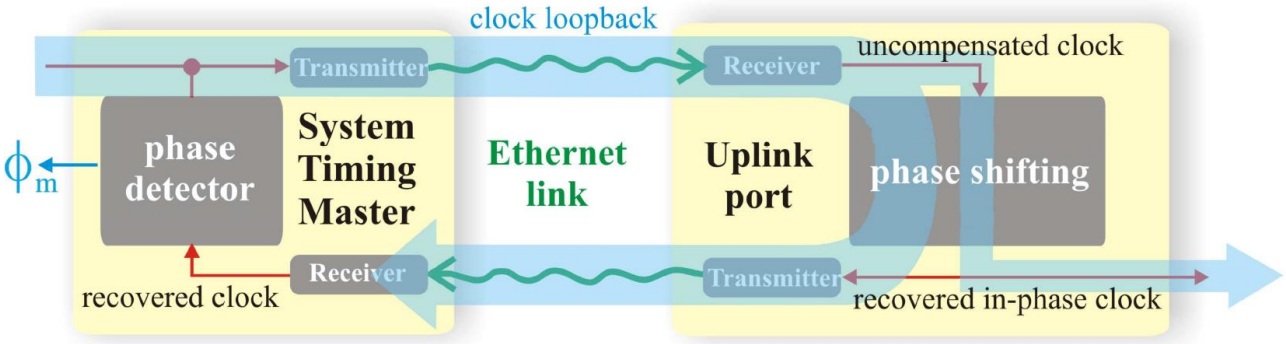



Figure 1.17 – Using DDMTD as a phase detector and phase shifter [6]

module based on phase/frequency detectors used to periodically measure the phase difference between the recovered clock and the master. This difference is transmitted to the slave for the compensation of the round-trip link delay with sub-nanosecond accuracy.

### 1.2.5 RADclock

The Robust Absolute Clock and Difference Clock (RADclock) project aimed to provide a new system for network timing that distinguishes between two types of clocks: **difference clocks** and **absolute clocks**. The difference clocks accurately measure the time elapsed between two events to under a microsecond and the absolute clock gives the time $C(t)$.

Unlike the feedback approach with PLL and Frequency Locked Loop (FLL) used by NTP to

adjust the client clock, RADclock uses a *feed-forward* approach where the correction parameters are not directly applied to the system clock. They are instead used only when a timestamp is required thus limiting the impact of frequent adjustments and inconsistencies. So, RADclock uses a bidirectional, minimum RTT-filtered, feed-forward-based absolute and difference synchronization algorithm.

Several studies showed the robustness and accuracy of RADclock compared to other solutions like NTP or PTP [38] [39] [40] [41]. In fact, the stability of approaches such as PLL and FLL cannot be guaranteed, they can lose their lock if conditions are not adequate, resulting in permanent error modes. Figure 1.18 shows that RADclock has better accuracy then ntpd over two weeks. In addition, difference clocks cannot even be defined in the feedback approach so we lose their benefit to measure delay, jitter, RTT or inter-arrival times in accurate way, which are examples of time differences.

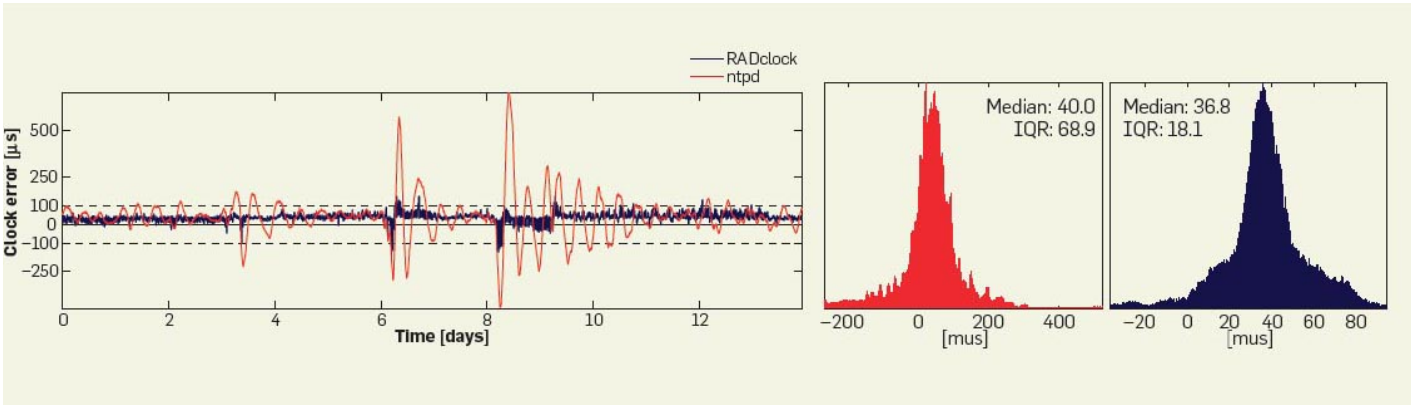



Figure 1.18 – Performance of RADclock and ntpd over 14 days synchronizing to a Stratum-1 server on the LAN [7]

## 1.3 Communication Model

Time synchronization requires communication. In the Internet, we usually have asymmetrical paths, which means that path from a source to a destination differs from the path from the destination back to the source. The difference in the transmission delays over forward and backward paths may come from two sources: first, the links between routers may have different capacity, e.g., an ADSL line [42]. Second, it may also come from *hot potato routing*: when two Autonomous System (AS) are competitors and have a peer-to-peer relationship in propagating Border Gateway Protocol (BGP) route advertisements, they have interest in getting rid of a given packet as soon as possible by forwarding them to the closest egress point in terms of the internal routing cost. Hot potato routing is a consequence of a rule in the BGP decision process stating that a router always

prefers to use a route learned over an eBGP session compared to a route learned over an iBGP session. Hot potato routing results in asymmetric paths because each AS sends packets through its favorable exit point, the points being different for different ASes.

Another cause of asymmetric traffic is link redundancy or alternative paths within networks. Since routing decisions occur independently for each packet, load-balancing algorithms may cause packets destined to the same endpoint to follow different paths. Other traffic engineering techniques, e.g., policy-based Shortest Path First (SPF), may also induce asymmetry in internal routing state of large provider networks.

### 1.3.1 Time Synchronization and Asymmetry

Much related work tried to study the asymmetry of transmission delays and find solution to deal with its impact to improve the time synchronization accuracy. Freris et al. [43] analyzed fundamental limits on synchronizing clocks over networks and showed that asymmetry cannot be measured only based on timestamps in a pairwise synchronization system even with an infinite number of round trip measurements.

Lévesque and Tipper [44] surveyed the state of the art in clock synchronization over packet-switched networks and presented different mechanisms proposed to improve the synchronization accuracy of NTP and PTP.

Several authors considered exploiting delay characteristics to improve the accuracy of time synchronization by trying to estimate the difference in paths to take into account in the offset calculation. Tsuru et al. [42] considered the case of asymmetric delays due to the difference of bandwidth on the paths. The authors validated the scheme over an Asymmetric Digital Subscriber Line (ADSL) link with asymmetric bandwidth.

PTP introduced some mechanisms to mitigate the negative effects of asymmetric links on synchronization accuracy [44]: residence time at intermediate nodes, $D_{asym}$, asymmetric delay parameter, and peer-to-peer path correction. In a network with non PTP switches, Zarick et al. [45] measured synchronization accuracy as low as 450 $\mu$s under the presence of asymmetric delays. Lee [46] proposed to take into account asymmetric bandwidth based on link speed measurements. A slave initiates a block burst transmission at the master by an `Asymm_Check_Req` message, it measures the time interval for receiving the message burst from the master, and sends the burst back. The scheme allows to estimate the link speed ratio between the forward and backward path.

Schriegel et al. [47] characterized the variable delays of wireless links and used the characteristics to compensate non-deterministic forwarding delays of PTP synchronization frames by using different send rates for `Sync` messages. They evaluated the method in a real setup consisting of a Real-Time Ethernet with a wireless extension. Murakami and Horiuchi [48] proposed to add prob-

ing messages before and after `Sync` and `Delay_Request` messages in PTP to detect link utilization, which improved synchronization accuracy.

Exel [49] analyzed various asymmetry mitigation approaches for software timestamping and proposed a timestamp correction-based asymmetry compensation scheme that takes into account bandwidth asymmetry. He showed precision improvement with measurements using Wireless Local Area Network (WLAN) synchronization hardware.

Lévesque and Tipper [50] considered a PTP set up with PTP support on only a subset of nodes on the path between a client and a server. They proposed a probing-based mechanism to estimate asymmetry and improve the synchronization performance. The protocol is similar to that by Lee, but is lightweight and takes into account the per-packet control delays.

Hajikhani et al. [51] considered PTP in asymmetric packet-based networks and proposed a method to estimate the asymmetric random parts in one-way delays, however, they assumed that the constant parts of asymmetric delays are equal in both directions.

WR defines a calibration procedure [52] to estimate the asymmetry in fiber propagation latencies by connecting the WR master and the WR slave with oscilloscopes.

# Chapter 2

# Security of Time Synchronization

*"Time is like the wind, it lifts the light and leaves the heavy."*

Domenico Cieri Estrada

## Contents

## 2.1 Introduction

At the beginning of the use of time synchronization protocols, their security was not seen as an immediate need, after all, time is not a secret. In addition, time synchronization protocols were designed to be lightweight and any attempts to authenticate the source of time add necessarily additional latency that had a negative impact on the objective of time synchronization. All these considerations resulted in time synchronization protocols that did not include robust security mechanisms in their initial designs. However, as time protocols are becoming increasingly common and widely deployed, concerns about the resulting exposure to various security threats were raised and security has become an integral part of network time synchronization.

Over the last few years, NTP has received some attention from security research due to software implementation flaws and its potential to act as an amplifier for DDoS attacks. In fact, an attacker can exploit some NTP server functionality like the *monlist* command, this command make the server responds with the last 600 received requests, so it can be used by an attacker to overwhelm the targeted network with an amplified amount of UDP traffic rendering the target unavailable.

The attacks against time synchronization protocols matter because the correctness of time underpins many other protocols and services. For example, validating a public key certificate requires confirming that the current time is within the certificate validity period. Validation with an inaccurate clock may cause expired or revoked certificates to be accepted as valid. We find the same issue for ticket verification in Kerberos. Another example is the HTTP Strict Transport Security (HSTS) policy [53] that specifies the duration of time that Secured Hypertext Transfer Protocol (HTTPS) must be used. A downgrade attack can be possible by the expiration of the HSTS policy due to a browser clock that jumps ahead. This attack on HSTS was demonstrated by Selvi [54].

So, many network security mechanisms rely on time as part of their operation. If an attacker can spoof the time, she may be able to bypass or neutralize other security elements. In practice, most NTP servers do not authenticate themselves to clients, so the attacker can intercept responses and set the clock arbitrarily. Malhotra et al. [55] presented a variety of attacks that rely on unauthenticated NTP, further emphasizing on the need for authenticated time synchronization.

## 2.2 Security Threats

Operating over the Internet, a widely deployed time service such as NTP can be vulnerable to different types of attacks, which might attempt to disrupt the protocol operation or the data it conveys. The most obvious goal of the intruder is to disrupt the protocol operations, to clog the network, the server or the client with a high volume of traffic, or force the protocol to consume

significant resources, e.g., in expensive cryptographic computations.

The IETF TICTOC Working Group identifies these threats and the RFC7384 [56] documents are the results of that analysis. It distinguishes the threat model in terms of an internal versus an external attacker, and in terms of Man-in-the-Middle (MiTM) versus packet injection types of attacks.

Several potential threats to network time synchronization protocols were identified:

- Manipulation of time synchronization packets,
- Masquerading as a legitimate participant in the time synchronization protocol,
- Replay of legitimate packets,
- Tricking nodes into believing time from the wrong master,
- Intercepting and removing valid synchronization packets,
- Delaying legitimate time synchronization packets on the network,
- Denial of service attacks on the network at layer 2 and layer 3,
- Denial of service by overloading the cryptographic processing components,
- Denial of service by overloading the time synchronization protocol,
- Corruption of the time source used by the grand master,
- Protocol design and implementation vulnerabilities, and
- Using the time synchronization protocol for broader network surveillance and fingerprinting types of activities.

## 2.3 Possible Attacks against Time Synchronization Protocols

Let us explain some of possible attacks in the context of the threat model determined above:

In **the MiTM attack**, the intruder can intercept a client and server packets and prevent their onward transmission, it can then alter and relay them to their destination to maliciously tamper with the protocol or simply drop them to prevent the destination from receiving the protocol packets. In **a replay attack**, the intruder intercept and resend previous NTP packets. An intruder can also attempt **a delay attack**, in which client or server packets are delayed a constant or variable time, but otherwise are unchanged. In **the masquerade attack**, the intruder assumes the identity of a legitimate server. In addition of all these attacks, one or more intruders can collaborate in **a**

**Denial Of Service (DoS) attack**, which attempts to deny service by flooding the network, clients, or servers with a high level of bogus traffic. A DoS attack may be effective if it forces needless and expensive cryptographic calculations causing high utilization of the cryptographic engine at the receiver, which attempts to verify the integrity of these fake packets.

## 2.4 Security Requirements

Based on the threat analysis, the IETF TICTOC Working Group specified in RFC7384 the security requirements for network time synchronization protocols [56] and analyzed them in terms of being required and being recommended/optional depending on the needs of the application.

These requirements include:

- Authentication and authorization of the clock identity,
- Integrity of the time synchronization protocol messages,
- Prevention of various spoofing techniques,
- Protection against Denial of Service (availability),
- Protection against packet replay,
- Timely refreshing of cryptographic keys,
- Support for both unicast and multicast security associations,
- Minimal impact on synchronization performance,
- Confidentiality of the data in the time synchronization messages,
- Protection against packet delay and interception, and
- Operation in a mixed secure and non-secure environment.

The NTP security model is based on these requirements and considers the data in an NTP packet to be public values, so there is no attempt to encrypt the data itself; only to confirm authenticity of the sources and avoid attacks. In our case, SCPTime project adds stronger requirements like proving the authenticity of clients and keeping the server lightweight and stateless.

Besides the most used client-server mode, NTP provides a mode for synchronization of symmetric peers, a mode for exchanging control messages, and a broadcast mode. These modes have different security and performance requirements. The symmetric and control modes have more

rigorous security requirements when compared to the client-server mode. However, the client-server mode requires more attention to resource utilization, since NTP servers may be contacted by a high number of clients and may not be able to maintain the state information for each client.

The distinction of the security requirements of each NTP mode is very important. Malhotra et al. [57] showed that NTP vulnerabilities arise because client/server mode and symmetric mode have conflicting security requirements while RFC5905 [24] suggests identical processing for incoming packets of both modes.

## 2.5 NTP Security

Early versions of NTP had no standardized authentication method. The first effort to secure NTP was proposed in its version 3. NTPv3 offered symmetric cryptographic authentication by appending an MD5 hash keyed with a symmetric key to NTP packets. However, the pre-shared key approach did not scale enough for large scale network deployments. Therefore, NTPv4 introduced a public key authentication mechanism called *Autokey*, which has not seen widespread adoption because a security analysis has demonstrated a number of security issues with Autokey. It uses small 32-bit seeds that can be easily brute forced to then forge packets. Because of the shortcomings of the preshared key and Autokey, there is an ongoing effort in the Internet Engineering Task Force to propose Network Time Security (NTS).

Details about these different approaches are presented in the following sections.

### 2.5.1 NTPv3 Symmetric Key Authentication

NTPv3 applies a symmetric key for the calculation of the digest, which guarantees authenticity and integrity of the exchanged packets. The calculation of the digest may be based on a Message Digest 5 (MD5) hash. If the NTP daemon is built on top of an OpenSSL library, NTP can also base the calculation of Message Authentication Code (MAC) upon SHA-1 or any other digest algorithm supported by OpenSSL library. To use this approach, participants have to exchange the key, which consists of a keyid with a value between 1 and 65534 and a label which indicates the chosen digest algorithm. The NTP process has to explicitly add each key it wants to trust to a list of trusted keys in its configuration file.

NTP does not provide a mechanism for the exchange of the keys between the associated nodes nor a mechanism to automatically refresh the keys. Therefore, symmetric keys must be preconfigured manually or exchanged securely by external means. For instance, NIST distributes symmetric keys once per year via mail to users for some important public stratum 1 NTP servers. The US Naval Office proceeds in a similar way [58]. In conclusion, such a shared key scheme is not

scalable to large environments.

### 2.5.2 NTPv4 Autokey Public Key Authentication

NTPv4 introduced the Autokey protocol [59] as an authentication method based on public key cryptography and digital signatures (PKI). Published in 2010, Autokey is designed to work on top of authenticated NTPv3. Autokey uses MD5 and a identity scheme to prevent malicious attacks. However, recent research done by Rottger revealed several weaknesses inherent in the Autokey protocol [60]. For instance, the seed value on 32 bits used to compute the cookie can be easily brute-forced by a MiTM adversary with sufficient computational power to generate all possible seed values and use the cookie to authenticate chosen adversarial NTP packets. The server does not perform authenticity verification of a client and relies on the client IP address to compute the cookie, which can be exploited by an adversary to obtain the client cookie (Cookie Snatcher Attack) [60].

### 2.5.3 ANTP: Authenticated NTP

ANTP was originally intended as a means to address the vulnerabilities in the Autokey protocol. ANTP [61, 62, 63] supports authentication of NTP servers based on certificates and guarantees message integrity via MAC 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 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.

### 2.5.4 Network Time Security (NTS)

The NTS [64] protocol is an in-progress alternative security protocol proposed by the IETF NTP Working Group as an enhanced replacement for Autokey. The main objectives of NTS are to authenticate NTP participants, to ensure authenticity and integrity of the exchanged time synchronization packets, and to provide replay protection. NTS proposes an additional goal of providing "unlinkability", which ensures that NTS does not leak any data that would allow an attacker to track mobile NTP clients when they move between different networks.

The security of NTS is based on Transport Layer Security (TLS) and Authenticated Encryption with Associated Data (AEAD) [64]. The NTS protocol is structured as two coupled sub-protocols :

- The first protocol (NTS-KE) handles initial authentication and key establishment over TLS.
- The second one handles encryption and authentication during NTP time synchronization via extension fields in the NTP packets, and holds all required state only on the client via opaque cookies.

As depicted in Figure 2.1, the typical protocol flow is as follows:

1. The client connects to an NTS-KE server on the NTS TCP port and the two parties perform a TLS handshake. Via the TLS channel, the parties negotiate some additional protocol parameters and the server sends the client a supply of cookies along with a list of one or more IP addresses to NTP servers for which the cookies are valid. The parties use TLS key export [65] to extract key material (two AEAD keys: a client-to-server (C2S) key and a server-to-client (S2C) key, which will be used in the next phase of the protocol. Then, the server closes the connection and discards the associated state.

2. Time synchronization proceeds with one of the indicated NTP servers over the NTP UDP port.The client sends an NTP client packet to the server, which includes several extension fields. Included among these fields are a cookie (previously provided by the key exchange server), a Unique Identifier Extension field that provides the client with a cryptographically strong means of detecting replayed packets, an authentication tag, computed using the key material extracted from the NTS-KE handshake. The NTP server uses the cookie to recover

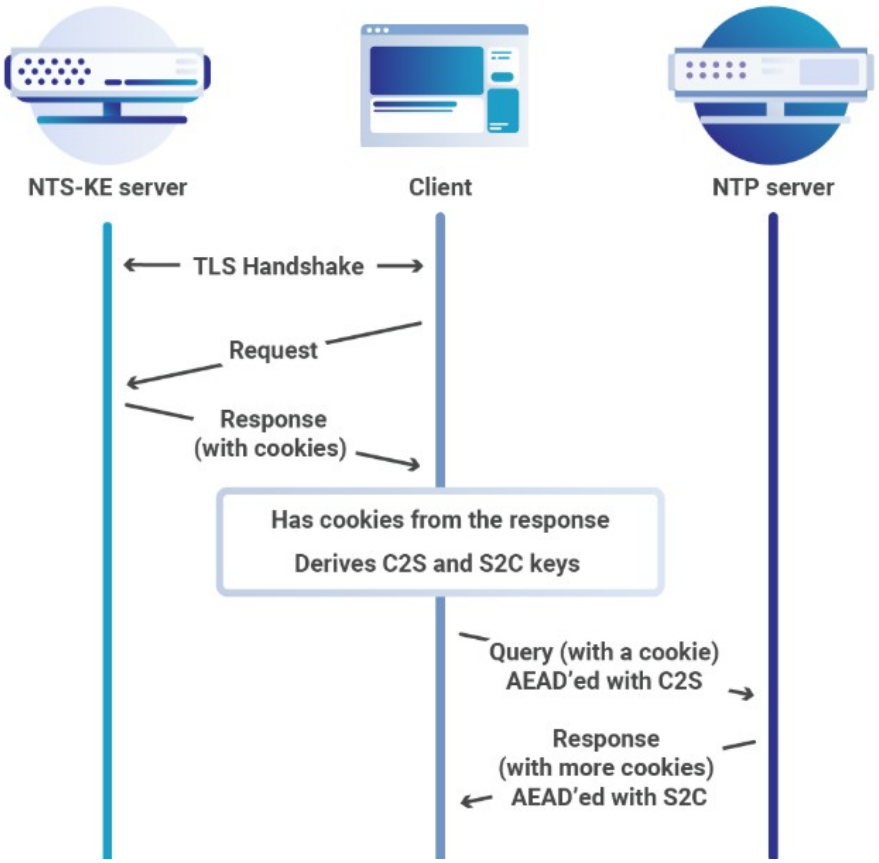



Figure 2.1 – NTS protocol exchanges [8]

this key material and send back an authenticated response. The response includes a fresh, encrypted cookie, which the client then sends back in the clear subsequent request. This constant refreshing of cookies is necessary to achieve the NTS unlinkability goal.

## 2.6 Security and Performance

The trade-off between security and performance is an important topic studied by many researchers. Aldini et al. [66] studied the trade-off between performance and security of an adaptive protocol for the secure transmission of real-time audio over the Internet. Haijin et al.[67] analyzed trade-off between security and performance of Networked Control System (NCS), related to the delay added by the data encryption and decryption procedure. Zeng and Chow [68] proposed an optimal trade-off between performance and security using coevolutionary algorithm. Haleem et al. [69] introduced the trade-off between security and throughput in wireless networks. Yau et al. [70] presented an adaptive trade-off model based on the definition of a trade-off objective function that can be used to adjust security configurations of services to provide sufficient protection and satisfy service performance requirements for Service Oriented Architecture-based systems.

Time synchronization protocols need also this kind of trade-off between security and performance. In fact, the security mechanisms must be designed in a way that does not downgrade the quality of the time transfer [56], which implies that they minimize the bandwidth overhead-/latency required for the security protocol (e.g., execution of cryptographic operation). Since the performance constraints on time synchronization protocols are driven by the fact that time servers are heavily loaded, these servers must be always available to provide responses to time clients. The time server should stay stateless, so the security mechanism should not have storage requirements of the client state on the server.

The ANTP protocol [61, 62, 63] achieves high performance while maintaining high security by relying only on symmetric cryptography during the frequent time synchronization phase, which makes it only slightly more expensive than an unauthenticated NTP. ANTP also leaves the server stateless by offloading state to clients.

## 2.7 Circular Dependency between Time Synchronization and Security

Many systems have increasing dependency on digital certificates and public-key infrastructure for authentication and ensuring confidentiality of communications. TLS, Secure Shell (SSH), and Internet Protocol Security (IPsec) protocols use certificates for this purpose. Many of these certificate schemes (in particular, X.509 certificates) specify the periods of validity and provide revocation lists to confirm that a certificate is valid at the time of use. Thus, certificates in turn depend on the synchronization of time between the issuer and the verifier. A malicious party that has the ability to offset the verifier system clock from the issuer can replay previously revoked or compromised certificates.

In fact, the operation of the authentication mechanism and the time synchronization mechanism are inextricably intertwined. It is a vicious circle: reliable time synchronization requires cryptographic materials that are valid only over designated time intervals, but time intervals can only be enforced when participating servers and clients are time synchronized.

NTP combats this problem by querying multiple parties and combining the received time samples using Byzantine Agreement mechanisms to find a majority offset to the local clock. However, this approach assumes that most implementations query multiple parties, which in typical deployments may not be the case as evidenced by the implementation of the Simple Network Time Protocol that queries a single server when requesting time synchronization. NTP also assumes the attacker can only affect some minority of the queried parties, which is not a realistic assumption for a man-in-the-middle attacker in control of the network.

On the other hand, 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. Mizrahi also points out the problem without suggesting a possible solution [71].

To solve this issue of circular dependency between time synchronization and security, some protocols were proposed, the following section presents these protocols.

### 2.7.1 RoughTime Protocol

RoughTime [72] is a Google project that provides a simple and flexible protocol used to achieve rough time synchronization. It lacks the precision of NTP, but its accuracy is enough for cryptographic applications like certificate validation. Clients use this protocol to synchronize their clocks with one or more authenticated servers.

As depicted in Figure 2.2, RoughTime is a multiple-round protocol in which client generates a random nonce and sends it to a Roughtime server. The RoughTime server reply includes the current time , the client nonce (*Nonce 1*), and a signature of both ($signature_1$). The current time is presented by timestamp ($Timestamp_1$) and a radius ($radius_1$), in microseconds, used to indicate the server certainty about the reported time). The client knows the server reply is fresh because it includes its previously generated nonce and it can also prove the authenticity of the server reply by verifying the received signature using the long term server public key, known to the client through an external mean.

If the client does not completely trust the server, it can request another one. For its subsequent request to the second server, client generates its nonce ($Nonce_2$) by hashing the reply from the first server with a random value. This proves that the nonce was created after the reply from the first server. It sends it to the second server and receives signature ($signature_2$) from it including that nonce and the time from the second server presented by timestamp ($Timestamp_2$) and radius ($radius_2$).

In case of different received time, if the time from the second server is before the first, the client has a proof of misbehavior of the second server because the reply from the second server implicitly shows that it was created later because of the way that the client constructed the nonce. In this way, clients can end up with a publicly verifiable, cryptographic proof of this misbehavior. However, with only two servers, the client has no idea of the correct time, so it needs to request multiple servers to get a chain of proof of any server misbehavior and also to get enough accurate replies to establish the correct time. If a server receives many requests, it can batch-sign a number of client requests by constructing a Merkle tree from the nonces. So, the server only signs the root of the tree and sends in its reply the information that proves to the client that its request is in the tree.

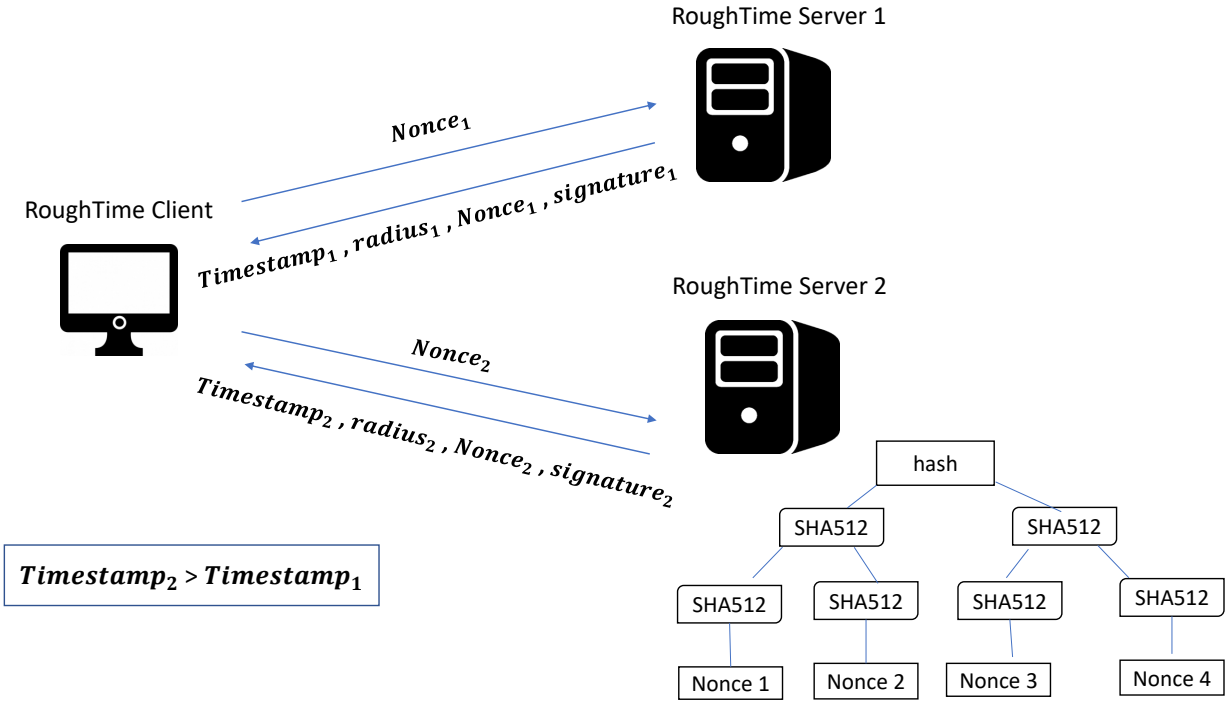



Figure 2.2 – RoughTime Protocol

### 2.7.2 Lightweight Authentication Time Synchronization Protocol

A Lightweight Authentication Time Synchronization Protocol (LATe) [73] is a protocol designed for Constrained Environments (ACE). It requires the least possible messages at the synchronizing node to minimize the cryptographic operations to execute and optimize the traffic sent on the network.

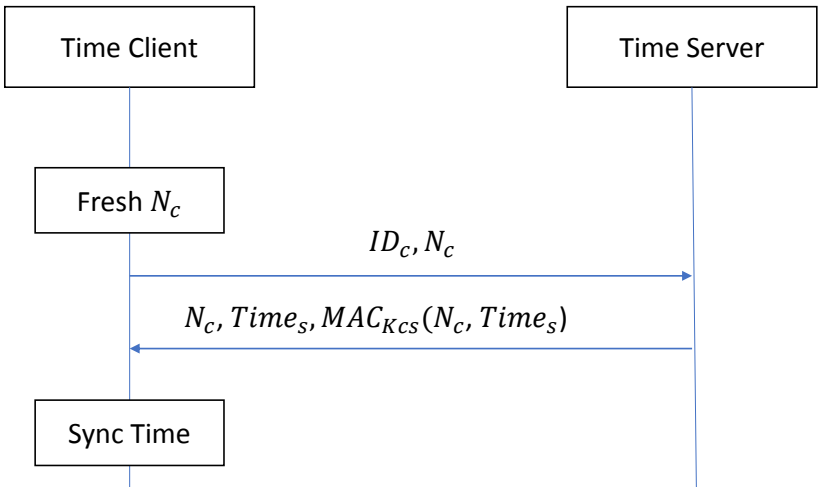



Figure 2.3 – Lightweight Authentication Time Synchronization Protocol

The protocol involves two entities: a Time Client (TC), the entity that attempts to update its

local clock and a Time Server (TS), the entity that provides its local time. The two entities TC and TS have a preshared cryptographic material $K_{cs}$.

The protocol consists of two messages exchanged between TC and TS (see Figure 2.3). TC generates a random nonce $N_c$ and the identity of TC then TS sends back to TC the nonce $N_c$, its local time representation $Time_s$ and a message authentication code $MAC_s$, computed on $N_c$ and $Time_s$ using the preshared key $K_{cs}$.

In the LATe protocol, fine grained time synchronization is not a goal since its precision depends on the one-way delay from the server to the client.

# Part II

# Contributions

# Chapter 3

# STS: Secure Time Synchronization Protocol

*"Time is the wisest of all things that are, for it brings everything to light."*

Thales

## Contents

## 3.1 Introduction

Most NTP servers do not authenticate themselves to clients, so a network attacker can intercept responses and set the timestamps arbitrarily. Malhotra et al. [55] present a variety of attacks that rely on NTP being unauthenticated, further emphasizing on the need for authenticated time synchronization.

An ubiquitous network time synchronization service such as NTP requires some provision to prevent accidental or malicious attacks on servers or clients. Clients should be able to determine that received messages are authentic, which means that the messages were actually sent by the intended server and not created or modified by an intruder.

We propose to go further in the support of NTP authentication with STS, a new secure authenticated time synchronization protocol suitable for widespread deployments. First, we describe the operation of STS. Second, we prove our design secure with a formal analysis using security protocol verification tools: Proverif [18] and Tamarin [19]. Third, we present the implementation of STS by extending OpenNTPD [20], and evaluate its performance by comparing the STS precision with unauthenticated NTP. This chapter also presents our solution for bootstrapping time synchronization based on the Bitcoin blockchain to solve the problem of the circular dependency of time synchronization and public key authentication.

## 3.2 STS Overview

STS was initially designed to satisfy the following SCPTime requirements:

- 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.

With STS, the client is capable of authenticating the server, and all messages from the server. Replay attacks are explicitly prevented for the client. STS also manages the authorization and authentication of clients, which is a specific SCPTime requirement because clients need to pay a subscription to get the SCPTime service providing the legal time of the country, it is the concept of "Time as a Service".

STS provides cryptographic assurance using symmetric cryptography that no modification of the packets has occurred in transit. STS operations have a little impact on the time synchronization precision thanks to a post-verification method. It also keeps the server stateless and lightweight.

### 3.2.1 STS Architecture

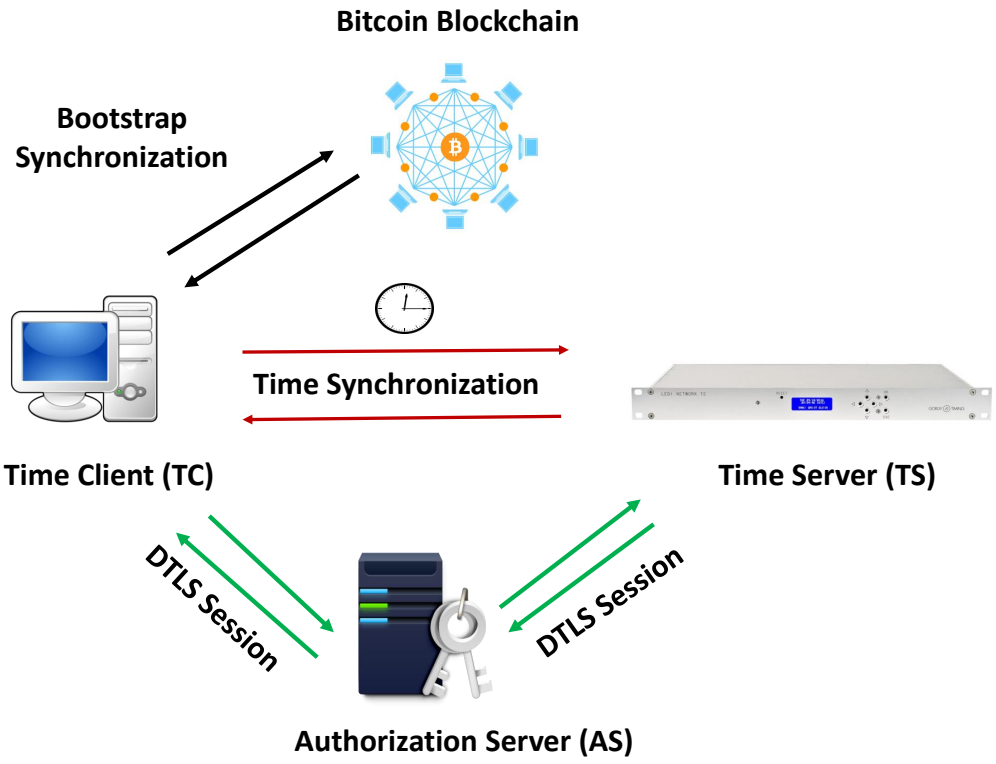



Figure 3.1 – STS architecture

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 3.1 presents the architecture of all the parties involved in the STS 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.

AS takes care of managing authorizations and storing the algorithms supported by servers. TS establishes a Datagram Transport Layer Security (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 provide its supported algorithms and 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 Bitcoin blockchain to validate the AS certificate.

| **Time Client** | **Authorization Server** | **Time Server** |
|---|---|---|
| | Setup Phase | |
| DTLS Session | | DTLS Session |
| $TCID \parallel TSID \parallel algo_{TC} \longrightarrow$ | | $\longleftarrow algo_{TS}$ |
| $\longleftarrow K \parallel K_e \parallel C \parallel algo_{TC-TS}$ | | $S \parallel K_e \parallel K_d \longrightarrow$ |
| | $algo_{TC-TS} \leftarrow negotiate(algo_{TC}, algo_{TS})$ | |
| | $C \leftarrow Enc_S(algo_{TC-TS} \parallel K)$ | |
| | Synchronization Phase | |
| $\alpha \leftarrow$ `in-progress` | | |
| $n \leftarrow \{0,1\}^{256}$ | | |
| $T_1 \leftarrow$ Now() | | |
| $m_1 \leftarrow T_1 \parallel n \parallel C$ | | |
| | $\xrightarrow{m_1}$ | |
| | | $T_2 \leftarrow$ Now() |
| $\tau_1 \leftarrow MAC_K(m_1)$ | | $algo_{TC-TS} \parallel K \leftarrow Dec_S(C)$ |
| $m_2 \leftarrow \tau_1$ | $\xrightarrow{m_2}$ | |
| | | Verify $MAC_K(m_1)$ |
| | | If Verify fails then $\alpha \leftarrow$ `reject` *and abort* |
| | | $T_3 \leftarrow$ Now() |
| | | $m_3 \leftarrow T_1 \parallel T_2 \parallel T_3$ |
| | $\xleftarrow{m_3}$ | |
| $T_4 \leftarrow$ Now() | | |
| RTT $\leftarrow (T_4 - T_1) - (T_3 - T_2)$ | | $\tau_2 \leftarrow MAC_K(m_1 \parallel m_3)$ |
| If RTT $> \Delta$, then $\alpha \leftarrow$ `reject` *and abort* | | or $\tau_2 \leftarrow DS_{K_d}(m_1 \parallel m_3)$ |
| Offset $\leftarrow [(T_2 - T_1) - (T_4 - T_3)]/2$ | | $m_4 \leftarrow \tau_2$ |
| | $\xleftarrow{m_4}$ | |
| Verify $MAC_K(m_1 \parallel m_3)$ or Verify $DS_{K_e}(\tau_2)$ | | |
| If Verify fails then $\alpha \leftarrow$ `reject` *and abort* else $\alpha \leftarrow$ `accept` | | |
| Time $\leftarrow$ Now() + Offset | | |

Figure 3.2 – 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 $K_d$, $\alpha$ - session state (`in-progress`, `accept`, `reject`), $\Delta$ - bound on RTT.

### 3.2.2 Protocol Description

We present below the details of the STS protocol operation presented in Figure 3.2.

1. **Bootstrap time synchronization:** In this initial phase, a client obtains approximate time in a secured way so that it can validate the authorization 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 Bitcoin blockchain as the basis for the approximate time [17]. More details about our proposal are in the end of this chapter.

2. **Client/Server Setup:** TS establishes a DTLS session with mutual authentication with AS to provide its supported Message Authentication Code (MAC) algorithms and obtains long-term secret $S$ as well as a pair of public/private keys $(K_e, K_d)$. Similarly, TC opens a DTLS session with AS and validates the AS certificate against the approximate time. Then, TC 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). AS negotiates the MAC algorithms to find the ones supported by both TC and TS. TC obtains the following parameters: the MAC 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 (the negotiated MAC algorithms and the session key $K$) 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:** During this phase, the usual time synchronization process is executed but with guarantee of authenticity and integrity of all messages exchanged between TC and TS:

   (a) TC sends a time synchronization request along with a nonce $n$ and opaque state $C$ to chosen TS. Then, it sends a second message with MAC computed with symmetric key $K$ over the first message.

   (b) Upon receiving the client request, TS timestamps the arrival of the request, then it verifies the freshness of nonce $n$ by searching if it already exists in its Nonce cache.

   (c) TS uses secret $S$ and the decryption algorithm (we suppose that AS and TS uses the same encryption/decryption algorithm) to decrypt symmetric key $K$ and the MAC algorithms from opaque state $C$, and verifies MAC of the received message with key $K$.

   (d) TS sends the reply message (unauthenticated NTP response) and generates another one with MAC tag computed with symmetric key $K$ over the request and reply mes-

sages. If TC requires non-repudiation, TS generates DS, a digital signature over the request and reply messages computed with TS private key $K_d$.

(e) 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 $\Delta$ (the bound on RTT to eliminate outliers), TC rejects the message and aborts time synchronization, which prevents TC from delay attacks.

The MAC algorithms are used to provide data integrity and authentication but they don't provide the property of non-repudiation because the client and the server share the same key. To solve this issue, Digital Signature (DS) are used. The public key is available to everyone. The private key is known only by the owner and can not be derived from the public one. When something is encrypted with the public key, only the corresponding private key can decrypt it. Moreover, when something is encrypted with the private key, then anyone can verify it with the corresponding public key and can know who the sender of the message really is and exactly which message was sent. So, the property of non-repudiation is achieved by using a digital signature.

## 3.3 STS Security Analysis

After the specification of the STS protocol, we wanted to perform a formal analysis. At that stage, our specification efforts can benefit from the results of the formal analysis: if the analysis unveils security vulnerabilities in the specification we can consider them for the redrafting of the specification.

To successfully perform the analysis, an important criterion is that the methods and tools used for the analysis can find existing weaknesses quickly and point them out clearly.

### 3.3.1 Choice of a Verification Tool

Several approaches are available to perform a formal analysis of a security protocol, the most established being theorem proving and model checking [74]. The model checking approach seemed a good approach because it provides an easy and fast way to unveil vulnerabilities in protocol specifications.

#### 3.3.1.1 Proverif Verification Tool

ProVerif [75] is tailored to be a verification tool for security protocols that offers help in the detection of attack scenarios. ProVerif supports a wide range of cryptographic primitives defined by *rewrite rules* [18] or by equations [75].

It takes as input a description of the protocol in a dialect of the applied $\pi$-calculus with support of types [76, 77], translates it into Horn clauses, and determines whether the desired security properties hold by resolution on these clauses [75]. Although there are time and clock related extensions of the $\pi$-calculus [78], none of them are applicable within ProVerif that does not support consideration of time and clocks for modeling.

ProVerif can prove various security properties, such as secrecy and authentication. It can also prove correspondence assertions of the form: "if some event is executed, then some events must have been executed before". In addition, ProVerif addresses injective correspondences, which require that "if an event is executed m times, then the corresponding events must have been executed at least m times".

ProVerif allows the user to specify so-called "queries", which can be used to specify the required protocol goals. When a goal has been specified in this way, ProVerif can look for a protocol state, which violates the condition corresponding to that goal. It can then return one of three possible results:

- "query" is **True**: There is no state that violates the goal condition,
- "query" is **False**: A state could be constructed that violates the given goal condition. In this case, ProVerif shows a trace of events leading to that state just before the result,
- "query" **cannot be proved**: ProVerif cannot prove that the goal is correct but it cannot construct a state violating the goal condition either. Since the problem of verifying protocols for an unbounded number of sessions is undecidable, this situation is unavoidable. However, ProVerif displays an attack derivation that can be useful to determine whether the query is true.

See Proverif User Manual [79] Sect. 3.3.1 for some more details on the different possible results and how to interpret them.

#### 3.3.1.2 Tamarin Verification Tool

The Tamarin prover [19] is a tool for symbolic modeling and analysis of security protocols. Like ProVerif, it takes as input a security protocol specification and its security requirements. It automatically outputs a proof whether no attack exists or a proof of concrete attack trace violating a security requirement.

Tamarin analyzes protocols with respect to a symbolic model of cryptography. It uses a term algebra with an equational theory to model cryptographic primitives and their properties.

### 3.3.2 Basic Assumptions

**Assumption 1** the adversary is a Dolev-Yao adversary [80] that "controls the network", the attacker has the following capabilities:

- It overhears and intercepts any message sent on the network. In particular, it can choose to prevent any message from being delivered in its origin form,
- It sends messages to any agent on the network, claiming to possess any identity it chooses,
- it can synthesize messages by inventing new values, assembling multiple values known to it into a tuple value, disassembling any tuple value that it knows into its single component values and applying any operator to any value known to it using any keys as long as they are in its knowledge.

**Assumption 2** cryptographic primitives are considered as perfect blackboxes, modeled by function symbols possibly with equations. The adversary can compute only using these primitives. Note that the Dolev-Yao model assumes cryptographic operations to be unbreakable.

### 3.3.3 Scope of the Analysis

In the STS protocol, the setup phase is done over a secure channel established with a DTLS end-to-end connection between the time server or the time client and the authorization server. Since, TLS is a standard protocol that is proven safe, the analysis of its vulnerabilities is unnecessary. So, we decide to only analyze formally with Proverif the synchronization phase that relies on a new protocol design.

The STS protocol employs only standard cryptographic primitives, which are AES-GCM, HMAC-SHA256, AES-CMAC, Ed25519, and MQQ-SIG. Since these standard cryptographic primitives are proven safe and because ProVerif is a symbolic protocol verifier, the cryptographic primitives used in the STS protocol will not be analyzed.

Since it is difficult to perform an analysis on protocols depending on time [81, 82], we have limited its scope to the analysis of security properties: authentication, integrity protection, and secrecy of keys that can be evaluated without considering time or clocks as part of the model.

## 3.4 Proverif-based Protocol Modeling and Results

We started the modeling with a basic protocol description then, we presented an advanced protocol description that describes all the aspects of the protocol.

### 3.4.1 Basic Protocol Description

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 security properties, as mentioned before, 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 a trace of an attack against the desired security property. The choice of queries was based on the security properties specified in the context of SCPTime.

**Goal: Secrecy property of keys $K$ and $S$** If client accepts keys from a server as being legitimate, then these keys are unknown to the adversary.

To evaluate the secrecy property of keys $K$ and $S$, the following line is added to the protocol description:

```
query attacker(K);
      attacker(S).
```

**Goal: Time synchronization authenticity** If the client accepts the data from a time response message as authentic from the server, then the server has indeed sent a time response message with the same time data and the same nonce, secured with the correct cookie $C$ for the association between client and server.

The authentication and integrity properties of the protocol are evaluated by means of correspondence assertions that capture the relationships between events added to the protocol description in ProVerif.

As mentioned before, ProVerif cannot handle time-related information, i.e., $T_1$,$T_2$, $T_3$, $T_4$, due to a lack of time related formal specification and verification techniques. Moreover, these time information is considered not secret nor unguessable. So, abstracting these information as new local variables unreasonably restricts the attacker's power. For this reason, we let the attacker input these time informations:

```
in(c, time);
```

### 3.4.2 Results of the Basic Analysis

As mentioned above, to evaluate the authentication and integrity properties, we added four events as depicted in Figure 3.3.


Figure 3.3 – Four events added in the protocol description in ProVerif to evaluate authentication and integrity properties

The results is as follows:

```
RESULT not attacker(K[]) is true.
RESULT not attacker(S[]) is true.
RESULT event(termClient(x_84,y,z,w)) ==> event(acceptsServer(x_84,y,z,w)) is true.
RESULT inj-event(termServer(x_85,y_86,z_87,w_88))
==> inj-event(acceptsClient(x_85,y_86,z_87,w_88)) is false.
RESULT (even event(termServer(x_4805,y_4806,z_4807,w_4808))
 ==> event(acceptsClient(x_4805,y_4806,z_4807,w_4808)) is false.)
```

#### 3.4.2.1 Replay attack

Since the used nonces in the protocol are not included in the basic protocol description in ProVerif, it means that nonces are not checked by the time server. Thus, an attacker can replay the messages of honest time clients. As depicted in the attack trace in Figure 3.4, an attacker replays the same message of an honest time client. Therefore, the assertion **"event termServer ==> event acceptsClient"** is not valid. So, in the next STS version, we should add the use of nonces in Proverif description.

#### 3.4.2.2 DDOS on the Server

In STS specification, the client sends an unauthenticated request and then sends a signed one, Proverif found a trace attack of a MITM attack that causes a DDOS attack on the server side that's

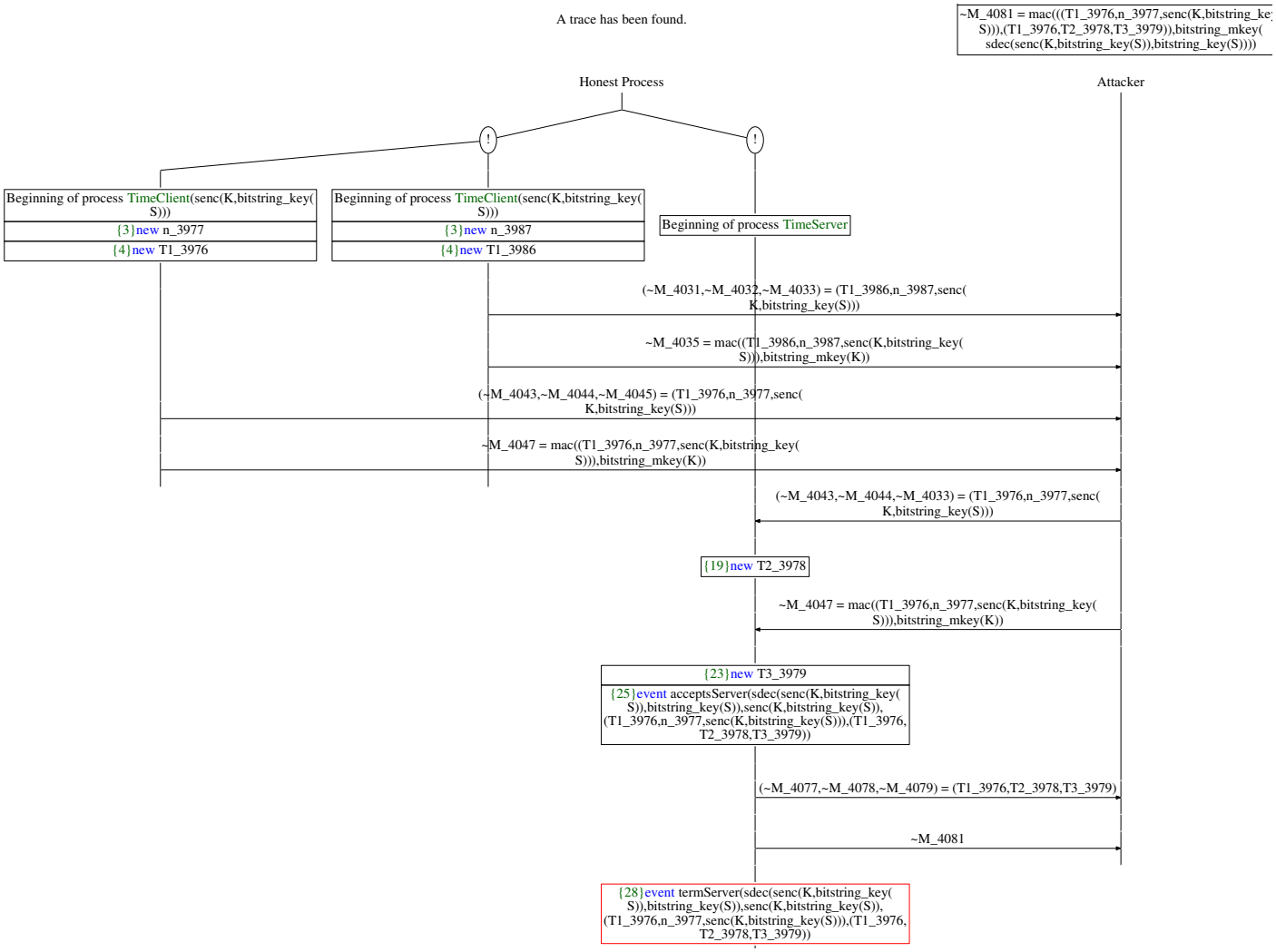



Figure 3.4 – Attack trace: replay attack in the basic protocol description in ProVerif

why the assertion **"event received1Server ==> event sent1Client"** does not hold. An attack trace is depicted in Figure 3.5 that shows that an attacker can send an arbitrary message to the time server, and the server engages in a session with the attacker.

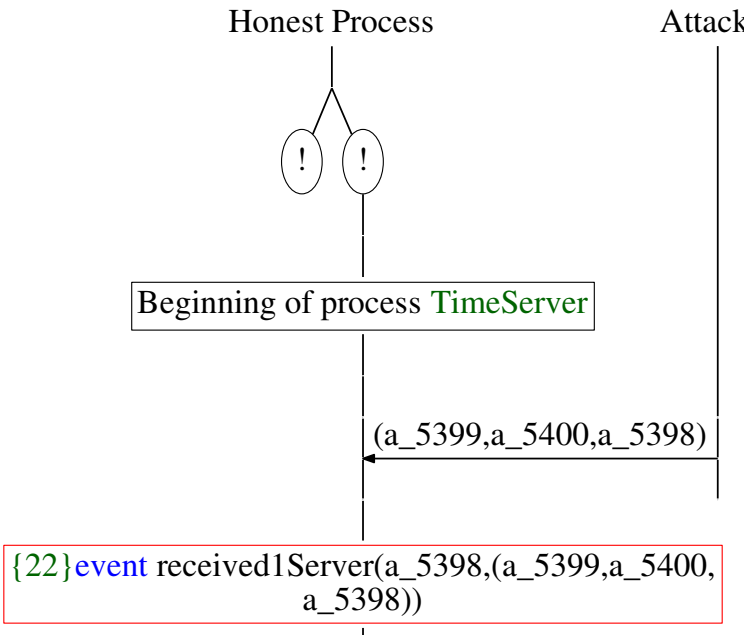



Figure 3.5 – Attack trace : unauthenticated message $m_1$

So, we decided to change the STS specification accordingly. In the final STS version as depicted in Figure 3.6, the client sends only the authenticated request that contains the Nonce $n$, the opaque state $C$ and the MAC authentication tag. This modification has no impact on the precision of timestamp $T_1$ because the client prepares its request and compute the MAC. Just before the moment of sending the request, it adds $T_1$, which is not the real value of $T_1$ but a random representation. The client stores locally the correspondence between the real value $T_1$ and the random representation sent in its request.

### 3.4.3 Advanced Protocol Description

We performed an advanced protocol description in ProVerif that addresses a number of considerations for the evaluation of the security properties.

#### 3.4.3.1 Nonce Handling

To take into account the use of nonces in the STS protocol and prevent the discovered replay attack, a cache of nonces is modeled in the description of the protocol. The nonces cache is modeled as a table defined as a cons-list with a membership predicate.

```
pred mem(bitstring, bset).
```

| Time Client | Authorization Server | Time Server |
|---|---|---|
| | **Setup Phase** | |
| DTLS Session | | DTLS Session |
| $\xrightarrow{TCID \parallel TSID \parallel algo_{TC}}$ | | $\xleftarrow{algo_{TS}}$ |
| $\xleftarrow{K \parallel K_e \parallel C \parallel algo_{TC-TS}}$ | | $\xrightarrow{S \parallel K_e \parallel K_d}$ |
| | $algo_{TC-TS} \leftarrow negotiate(algo_{TC}, algo_{TS})$ | |
| | $C \leftarrow Enc_S(algo_{TC-TS} \parallel K)$ | |
| | **Synchronization Phase** | |
| $\alpha \leftarrow$ `in-progress` | | |
| $n \leftarrow \{0,1\}^{256}$ | | |
| $T_1 \leftarrow$ Now() | | |
| $m_1 \leftarrow T_1 \parallel n \parallel C$ | | |
| $\tau_1 \leftarrow MAC_K(m_1)$ | | |
| $m_2 \leftarrow m_1 \parallel \tau_1$ | $\xrightarrow{m_2}$ | |
| | | $T_2 \leftarrow$ Now() |
| | | $algo_{TC-TS} \parallel K \leftarrow Dec_S(C)$ |
| | | Verify $MAC_K(m_1)$ |
| | | If Verify fails then $\alpha \leftarrow$ `reject` *and abort* |
| | | $T_3 \leftarrow$ Now() |
| | | $m_3 \leftarrow T_1 \parallel T_2 \parallel T_3$ |
| | $\xleftarrow{m_3}$ | |
| $T_4 \leftarrow$ Now() | | |
| RTT $\leftarrow (T_4 - T_1) - (T_3 - T_2)$ | | $\tau_2 \leftarrow MAC_K(m_1 \parallel m_3)$ |
| If RTT $> \Delta$, then $\alpha \leftarrow$ `reject` *and abort* | | or $\tau_2 \leftarrow DS_{K_d}(m_1 \parallel m_3)$ |
| Offset $\leftarrow \frac{(T_2-T_1)-(T_4-T_3)}{2}$ | | $m_4 \leftarrow \tau_2$ |
| | $\xleftarrow{m_4}$ | |
| Verify $MAC_K(m_1 \parallel m_3)$ or Verify $DS_{K_e}(\tau_2)$ | | |
| If Verify fails then $\alpha \leftarrow$ `reject` *and abort* else $\alpha \leftarrow$ `accept` | | |
| Time $\leftarrow$ Now() + Offset | | |

Figure 3.6 – Modification of STS design to prevent against the discovered replay attacks

```
clauses
  forall x:bitstring, y:bset; mem(x, consset(x, y));
  forall x:bitstring, y:bset, z:bitstring; mem(x, y) -> mem(x, consset(z, y)).
```

The nonces cache stores used nonces per a time client:

```
table cache_nonce(bitstring, bset).
```

The verification of a received nonce consists of, first of all, checking if the cache exists, and if it exists, of checking whether or not the nonce belongs to the cache using the membership predicate.

```
new cache:bset;
get cache_nonce(=C_s, cache) in (
if mem(n, cache) then
  (* replayed nonce, TimeServer process terminates*)
else (
  (* fresh nonce added to cache associated with C_s*)
  insert cache_nonce(C_s, consset(n, cache)); )
else (
    (* a cache is created for C_s with first nonce included*)
    insert cache_nonce(C_s, consset(n, emptyset)); )
```

#### 3.4.3.2 Authenticated Encryption

The opaque information $C$ transports key $K$ authenticated and encrypted using AES-GCM. In the protocol description, the cipher is modeled using the encrypt-then-mac paradigm.

```
(* Encrypt_then_MAC*)
  let C1 = senc(K, bitstring_key(S)) in
  let C = (C1, mac(C1, bitstring_mkey(S))) in}
```

#### 3.4.3.3 Per-message Evaluation

The advanced protocol description included new events that exhaustively evaluate not just the integrity of all messages, but also enables a closer look per message by evaluating the relationship between these messages, which allows to detect attacks against message freshness and interleaving session attacks (e.g., parallel session attacks). The added events are illustrated in Figure 3.7.

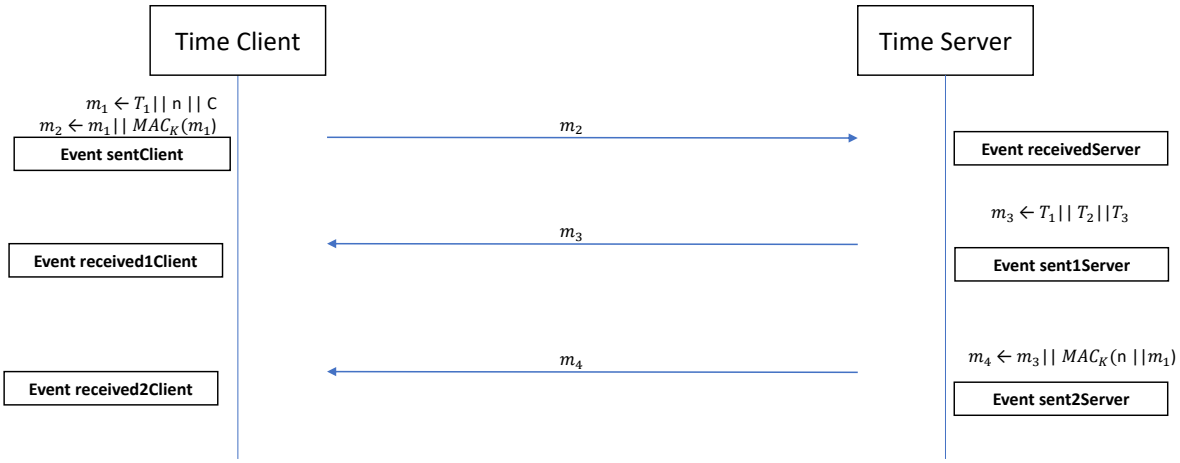



Figure 3.7 – Six events added in the protocol description in ProVerif to evaluate authentication and integrity properties.

### 3.4.4 Results of the Advanced Analysis

The ProVerif source code of the final STS version is given in Appendix A.

The results of the advanced Proverif analysis is as follows:

```
RESULT not attacker(K[]) is true.
RESULT not attacker(S[]) is true.
RESULT inj-event(received1Server(x_85,y_86))
 ==> inj-event(sent1Client(x_85,y_86)) is false.
RESULT (even event(received1Server(x_5376,y_5377))
 ==> event(sent1Client(x_5376,y_5377)) is false.)
RESULT inj-event(sent2Client(x_87,y_88,z_89))
==> inj-event(sent1Client(x_87,y_88)) is true.
RESULT inj-event(received2Server(x_90,y_91,z_92))
 ==> inj-event(sent2Client(x_90,y_91,z_92)) is false.
RESULT (but event(received2Server(x_10917,y_10918,z_10919))
 ==> event(sent2Client(x_10917,y_10918,z_10919)) is true.)
RESULT inj-event(received2Server(x_93,y_94,z_95))
 ==> inj-event(received1Server(x_93,y_94)) is true.
RESULT inj-event(sent1Server(x_96,y_97,z_98,w))
 ==> inj-event(received2Server(x_96,y_97,z_98)) is true.
RESULT inj-event(received1Client(x_99,y_100,z_101,w_102))
==> inj-event(sent1Server(x_99,y_100,z_101,w_102)) is false.
RESULT (even event(received1Client(x_24440,y_24441,z_24442,w_24443))
```

```
 ==> event(sent1Server(x_24440,y_24441,z_24442,w_24443)) is false.)
RESULT inj-event(sent2Server(x_103,y_104,z_105,w_106,v_107))
 ==> inj-event(sent1Server(x_103,y_104,z_105,w_106)) is true.
RESULT inj-event(received2Client(x_108,y_109,z_110,w_111,v_112))
 ==> inj-event(sent2Server(x_108,y_109,z_110,w_111,v_112)) is true.
RESULT inj-event(received2Client(x_113,y_114,z_115,w_116,v_117))
 ==> inj-event(received1Client(x_113,y_114,z_115,w_116)) is true.
```

#### 3.4.4.1 Race condition Attack

The Proverif trace as depicted in Figure 3.8 shows that the attacker can perform "a race condition" attack against the cache_nonce, whereby the attacker replayed the same message with the same nonce in two instances of the time server and the nonce checking in each instance is performed before inserting the new nonce in the cache. So, the message can be replayed in different instances of the time server. The assertion **"event receivedServer ==> event sentClient"** is valid, whereas its injective form **"inj-event receivedServer ==> inj-event sentClient"** is not valid because of the "race condition" attack.

To avoid "race condition" attacks against the cache of nonces, the "write access" to the cache must be restricted to only authorized processes, and the cache must be "reader-writer" locked using mutex (mutual exclusion). So this attack can easily prevented during the STS implementation.

#### 3.4.4.2 DOS Client Attack

The message $m_3$ sent by the time client is not authenticated, therefore the assertion **"event received1Client ==> event sent1Server"** is not valid. As depicted in the attack trace in Figure 3.9, an attacker can impersonate a time server by sending bogus information: $T_2$ and $T_3$.

In our STS design, the server sends two response packets, the first being the unauthenticated NTP packet $m_3$, and the second $m_4$ being the same NTP packet along with an extension field providing the authentication tag ensuring that $m_3$ was not changed in transit. We propose this "post-verification" method to avoid the degradation of time synchronization accuracy due to the time required to compute the authentication tag over the outgoing timestamp. In fact, client measures the roundtrip time based on the unauthenticated response, but does not update its clock until authenticating the response.

To satisfy the requirement of performance, we decided to not change our design. To prevent denial of service (DoS) attacks at the client side, common measures based on intrusion detection (IDS) and prevention systems (IPS) should be deployed (packet filtering in routers and firewalls).

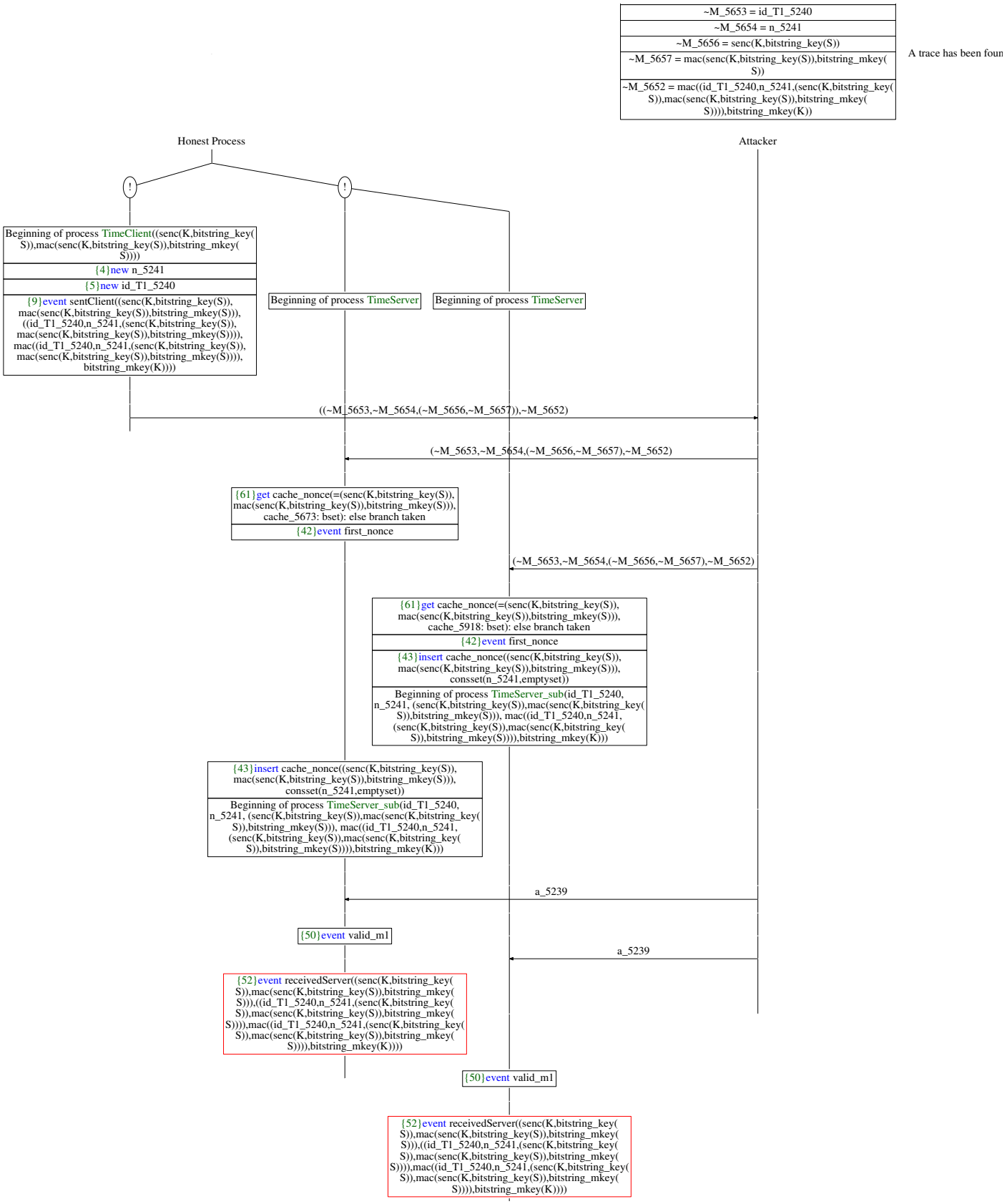



Figure 3.8 – Attack trace: time client messages replayed using race condition attack against the nonce cache

| Abbreviations |
| --- |
| ~M_13755 = id_T1_13644 |
| ~M_13756 = n_13645 |
| ~M_13758 = senc(K,bitstring_key(S)) |
| ~M_13759 = mac(senc(K,bitstring_key(S)),bitstring_mkey(S)) |
| ~M_13754 = mac((id_T1_13644,n_13645,(senc(K,bitstring_key(S)),mac(senc(K,bitstring_key(S)),bitstring_mkey(S)))),bitstring_mkey(K)) |

A trace has been found.

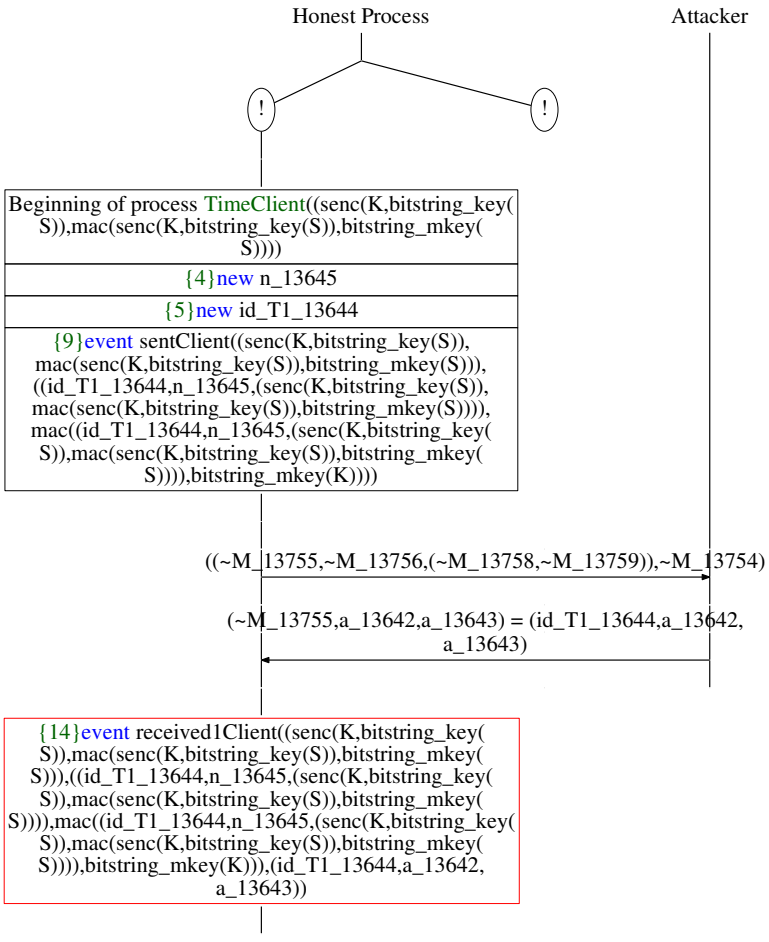



Figure 3.9 – Attack trace: unauthenticated message $m_3$

## 3.5 Tamarin-based Protocol Modeling and Results

The code of the modeling of the final STS version with Tamarin is presented in Appendix. We decided to use Tamarin to confirm the results obtained with Proverif.

In Tamarin, the protocol is modeled using multiset rewriting rules that have the following form:

```
rule name: premise --[ actions ]-> conclusion
```

The desired properties to be evaluated are denoted by lemmas:

```
lemma my_secret_key:
 "Forall tid key #i.
 Accepted( tid, key )@i =>
 ( not Ex #j. K(key)@j ) "
```

Like Proverif, the adversary in Tamarin is a Dolev-Yao adversary that controls the network and it is possible to specify its capabilities:

```
rule SessionKeyReveal:
 [ State( ThreadID, ... , Key ) ]
 --[ SessionKeyReveal( ThreadID, Key ) ]->
 [ Out( Key ) ]
```

### 3.5.1 Results of the Analysis

Like the Proverif analysis, the synchronization phase protocol is evaluated with respect to the security properties: secrecy of keys $K$ and $S$, and authentication and integrity of messages. The results of the analysis with Tamarin are the following:

```
summary of summaries:
analyzed: STS.spthy
distinct_nonces (all-traces): verified (2 steps)
S_secrecy (all-traces): \textcolor{green}{verified} (4 steps)
K_secrecy (all-traces): \textcolor{green}{verified} (6 steps)
Client_Auth (all-traces): \textcolor{green}{verified} (6 steps)
Server_Auth (all-traces): \textcolor{green}{verified} (14 steps)
m1_integrity (all-traces): \textcolor{green}{verified} (6 steps)
m3_integrity (all-traces): \textcolor{red}{falsified}- found trace (4 steps)
```

```
m4_integrity (all-traces): verified (14 steps)
```

Tamarin confirms the results obtained with Proverif. In fact, the Tamarin evaluation results prove that keys $K$ and $S$ are not disclosed during the STS protocol operation, the time client and the time server are authenticated, and that all messages, except for message $m_3$ sent without MAC, are protected in terms of integrity. For the message $m_3$ integrity, Figure 3.10 illustrates the attack trace output by Tamarin

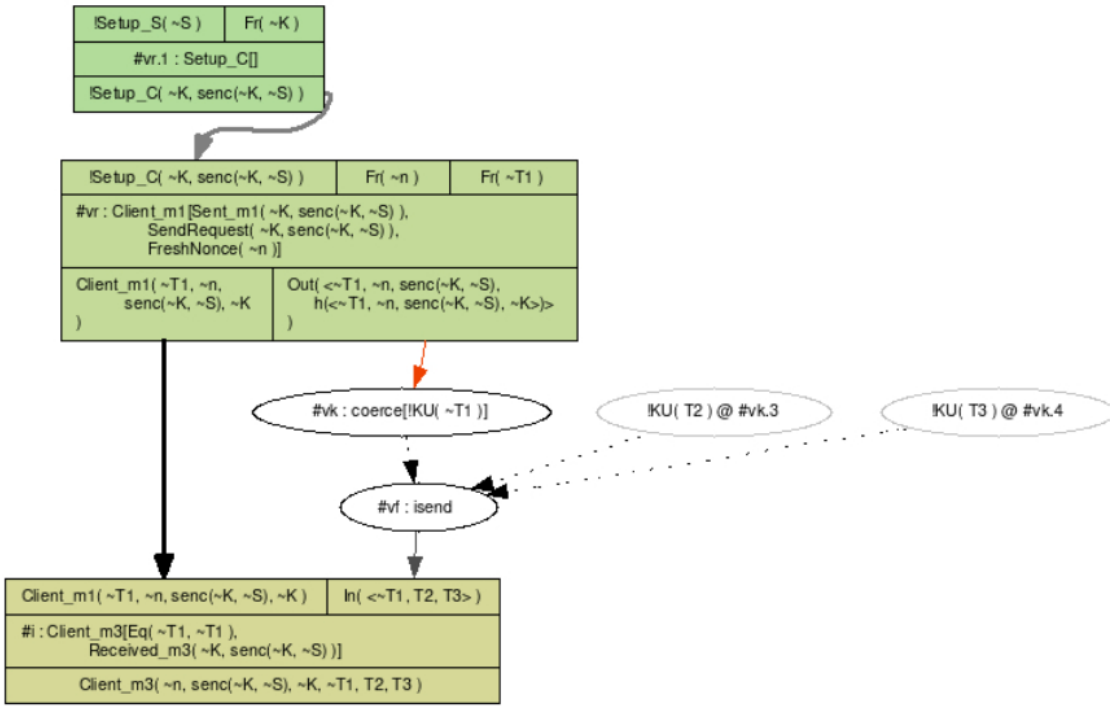



Figure 3.10 – Attack trace: message $m_3$ received by the time client may contain arbitrary data from an attacker.

As mentioned before, STS sends an unauthenticated message $m_3$ to avoid the degradation of time synchronization accuracy due to cryptographic operations to compute the MAC.

## 3.6 STS Implementation and Performance

We take advantage of the NTP extension fields for transporting the additional STS informations. The STS implementation extends OpenNTPd [20], under a BSD License, with the operation described in Figure 3.6 and we use the OpenSSL libcrypto library for cryptographic operations [83].

### 3.6.1 Cryptographic Primitives

We have chosen the following algorithms for cryptographic primitives:

Table 3.1 – Execution Time of MAC Primitives

| MAC Algorithms | Execution Time |
|---|---|
| HMAC-MD5 (generation) | 4 $\mu$s |
| HMAC-SHA256 (generation) | 4 $\mu$s |
| AES-CMAC (generation) | 3 $\mu$s |

Table 3.2 – Execution Time of DS Primitives [9]

| DS Scheme | Execution Time |
|---|---|
| Ed25519 (generation and verification) | 75 $\mu$s |
| MQQ-SIG (generation and verification) | 28 $\mu$s |

- AES-GCM [84] for symmetric encryption the server uses to decrypt the opaque value sent by the client,
- HMAC-SHA256 [85] and AES-CMAC [86] for the MAC algorithms,
- Ed25519 [87, 88] and MQQ-SIG [89, 90] for the DS algorithms.

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) [91]. 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 3.1 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. [9]: Table 3.2 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) [9].

### 3.6.2 STS Performance

The STS performance in the time synchronization phase was evaluated based on the initial design (see Figure 3.2). We have measured its precision on a LAN testbed presented in Figure 3.11: we run two PCs as TC and TS synchronized with the Gorgy Timing LEDI Network ITS v2m (GPS time reference) as the time source reference over the PPS (Pulse-Per-Second) interface. TC runs STS

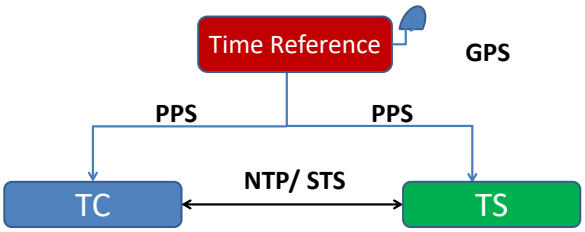



Figure 3.11 – LAN testbed for measurements.

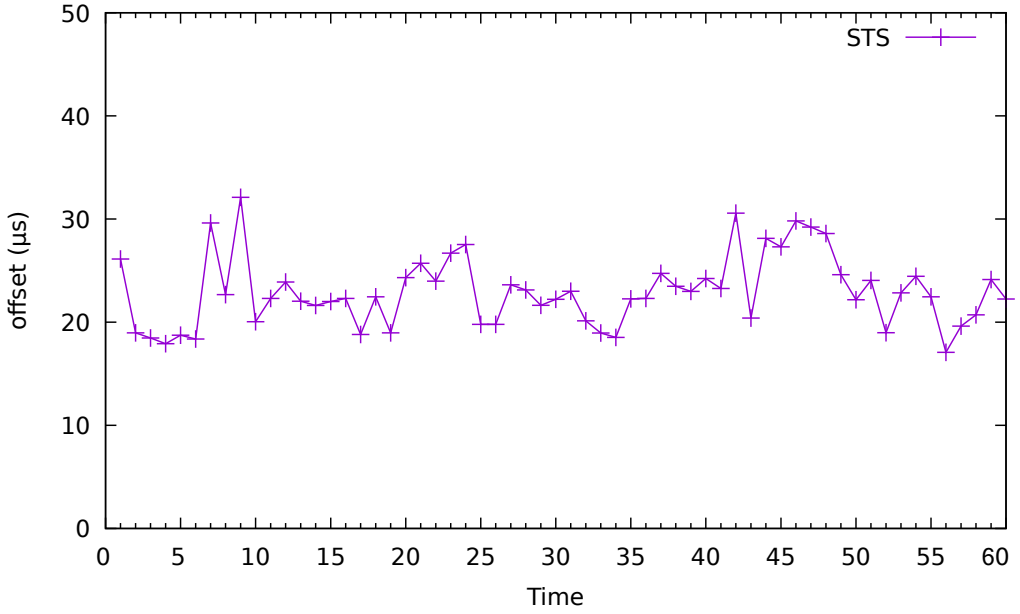
and unauthenticated NTP for a comparison based on the time reference at TC also obtained from ITS through the PPS interface.

Figure 3.12 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).

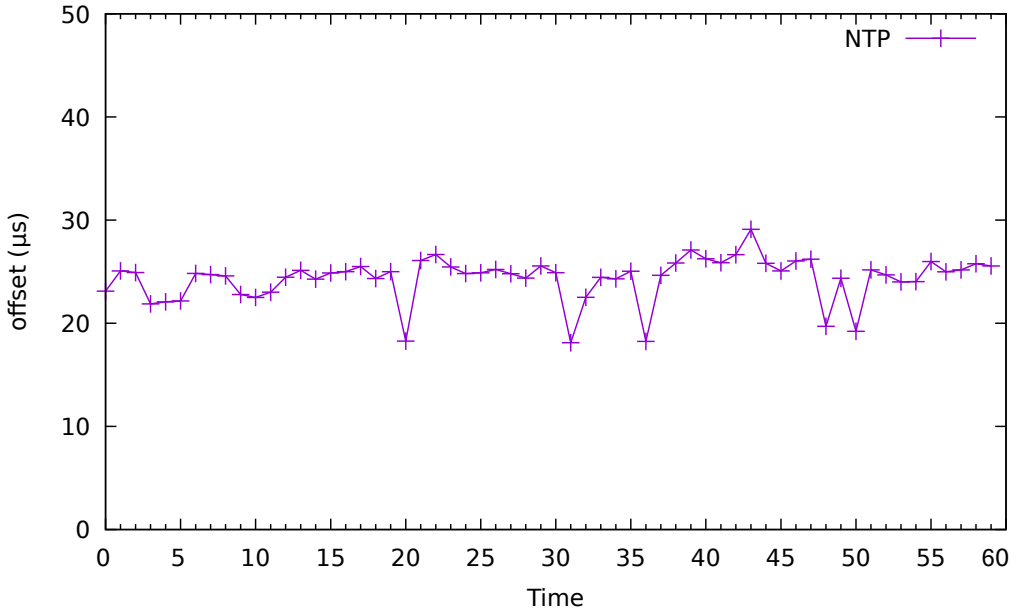
We have also measured the latency of client-server interactions on the LAN testbed: STS - 600 $\mu$s, NTP - 510 $\mu$s (RTT estimated with ping is 400 $\mu$s). The results show that STS only introduces a small overhead to time synchronization.

We wanted to evaluate STS performance over Internet. Figure **??** and Figure **??** show the precision of the offset estimation by NTP and STS over a period of 24 hours. We can observe that the precision of both protocols is comparable and we cannot really distinguish the overhead of the STS operation.

We have also measured the RTT between the client and the server over the Internet as depicted in Figure **??** and Figure **??**.




(a) STS




(b) Unauthenticated NTP

Figure 3.12 – Offset estimation precision during a 1 min. period.

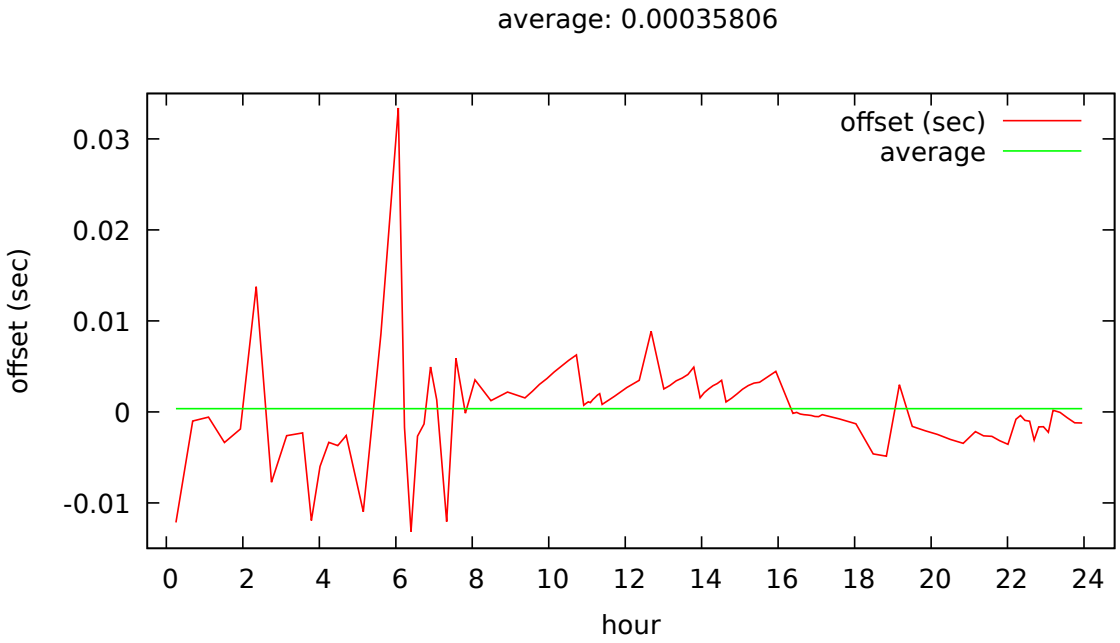



Figure 3.13 – NTP Offset estimation during 24 hours over Internet.

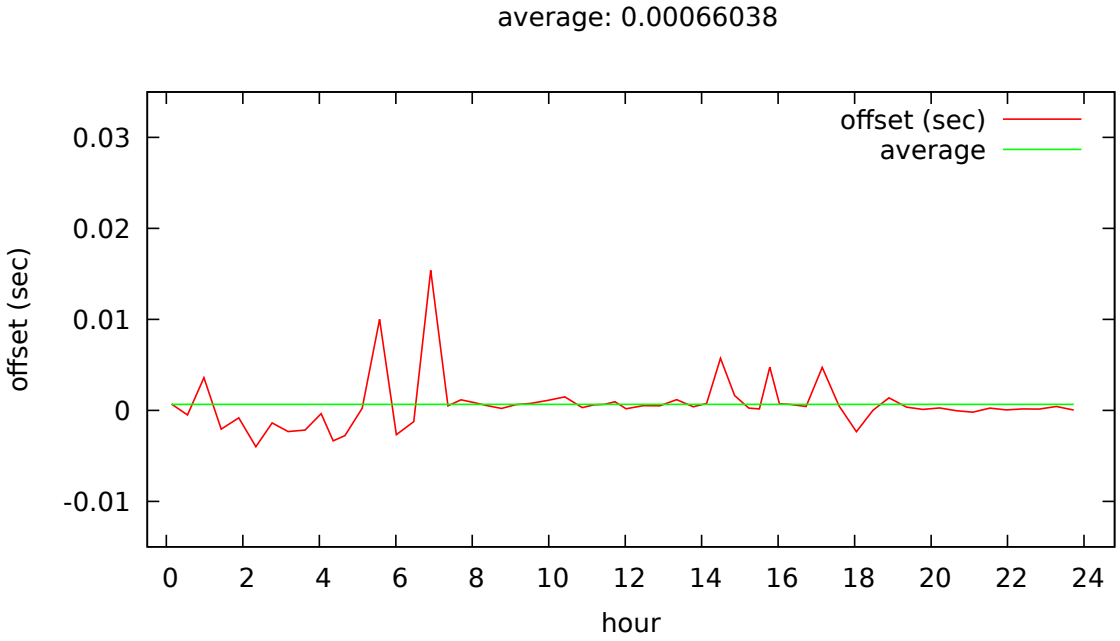



Figure 3.14 – STS Offset estimation during 24 hours over Internet.

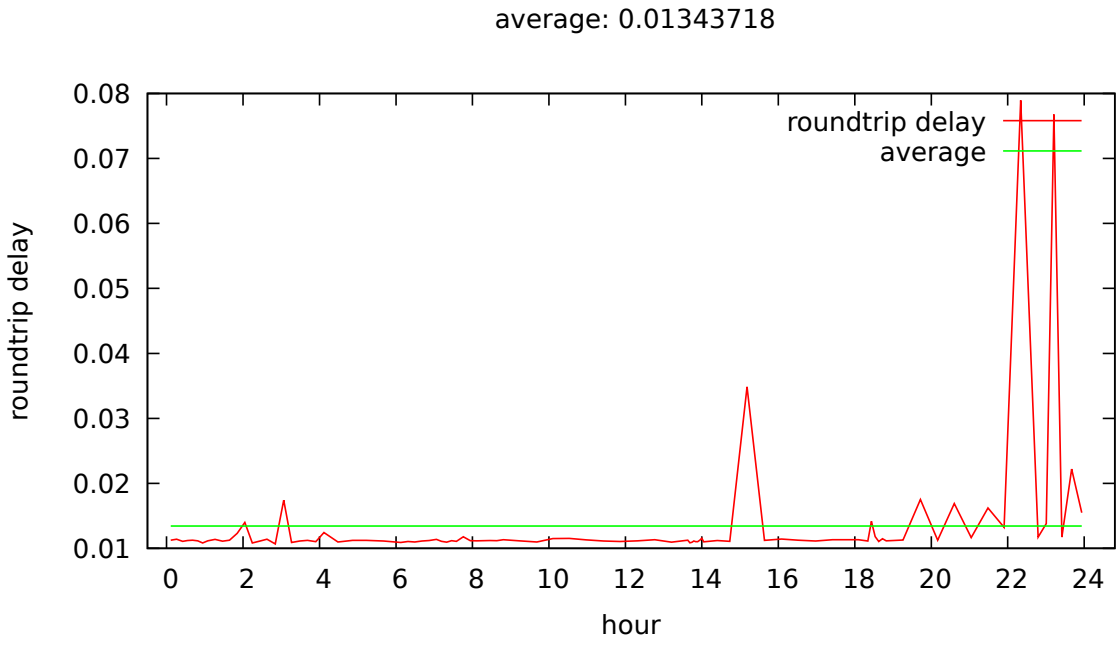



Figure 3.15 – Measured RTT using NTP protocol.

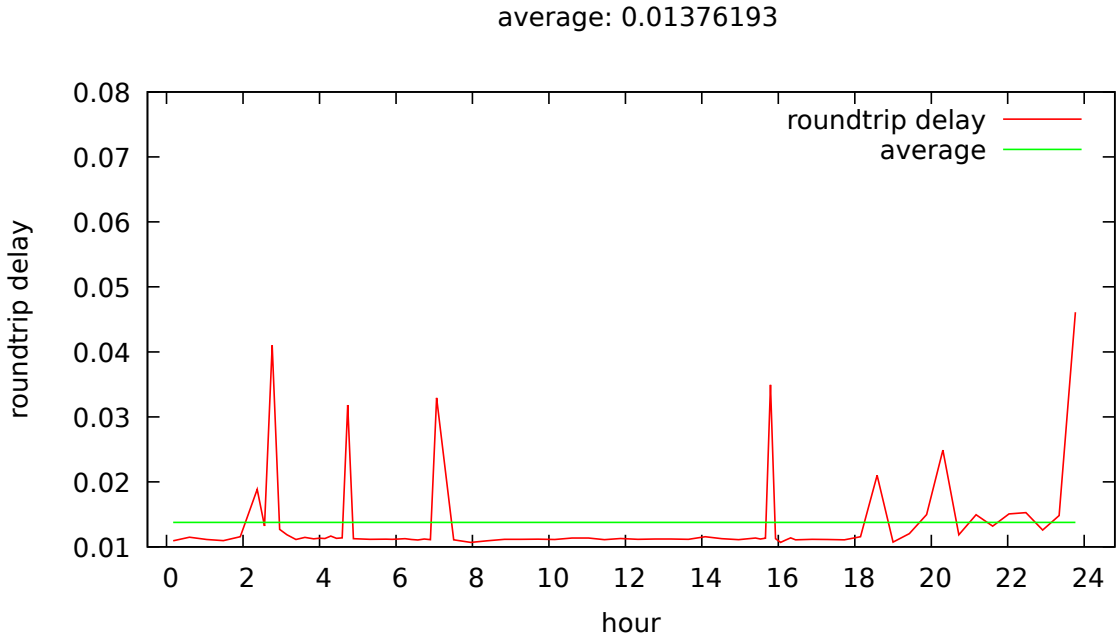



Figure 3.16 – Measured RTT using STS protocol.

## 3.7 Secure Bootstrap Synchronization Using the Bitcoin Blockchain

### 3.7.1 Bitcoin Blockchain

Bitcoin [17] is an open cryptocurrency and transaction system. To add a block of transactions into the Bitcoin network, nodes, called miners, solve a cryptographic proof-of-work puzzle. The difficulty of the puzzle is set in such a way that the creation of a block takes about 10 minutes.

Each block has a header that contains a field with a hash of the previous header to link the blocks together. Transactions are represented as leaves of a Merkle tree whose root is also included in the block header. So, using the header, we can prove that a transaction is part of a given block.

In addition, the block header includes a Unix timestamp corresponding to the time when the block was mined.

### 3.7.2 Block Timestamps

As mentioned before, each block has an associated timestamp. These timestamps are the following [92]:

- **decentralized:** no entity controls the database of timestamps, and all miners in the network validate the timestamp,
- **immutable:** once a timestamp has been verified and recorded, no one can delete it,
- **public:** the timestamps are publicly visible, and
- **programmable:** you can write code against the blockchain.

Timestamps are validated in a special way. A node considers a new block timestamp $T$ as valid if:

1. $T >$ the median timestamp of previous eleven blocks, and
2. $T$ - 2h < network time (defined as the median of the timestamps returned by all nodes connected to the node).

### 3.7.3 Simplified Payment Verification Mode

It is possible to verify payments without running a full Bitcoin node. In the SPV mode [93], a user only needs to keep a copy of the block headers of the longest proof-of-work chain, which she can get by querying Bitcoin nodes until she is convinced to have the longest chain, and obtain the Merkle branch linking the transaction to its block (Figure 3.17). By linking the block to a place in the chain, she can see that a Bitcoin node has accepted it, and blocks added after it further confirm the validity of the transaction.

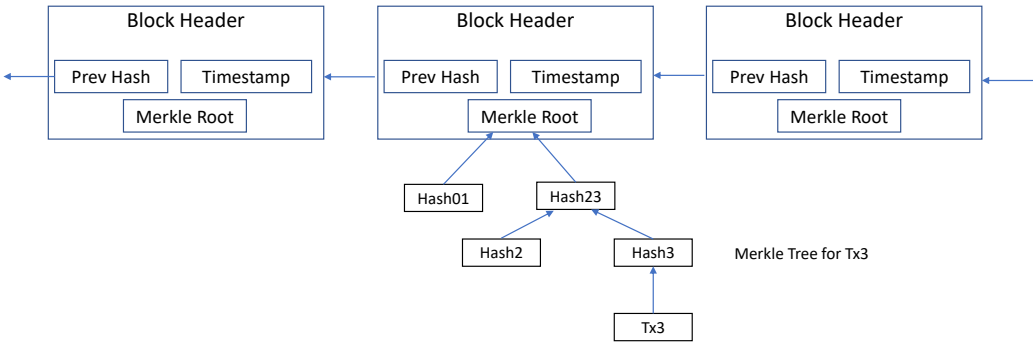



Figure 3.17 – Structure of the Bitcoin blockchain

### 3.7.4 Blockchain and Timestamping

As a consequence of the success and properties of Bitcoin, developers and researchers try to reuse the Bitcoin infrastructure to build new or enhance existing systems. One class of such systems is a decentralized timestamping service. For instance, the OpenTimestamps project [94] aims to standardize blockchain timestamping, where a timestamp authority, known from previous proposals [95], is replaced by a blockchain. More focused applications that rely on the blockchain timestamps include trusted record-keeping service [96], [97], decentralized audit systems [98], document signing infrastructures [99], timestamped commitments [100].

#### 3.7.4.1 Rough Time Synchronization using the Bitcoin Blockchain Timestamps

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 Bitcoin blockchain as the basis for the approximate time.

Figure 3.18 represents the steps of bootstrap time synchronization using the Bitcoin blockchain.

1. 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 $N-m$.

2. When TC starts to operate, it behaves like a Simplified Payment Verification (SPV) lightweight blockchain client [93]: it discovers peers in the Bitcoin P2P network and synchronizes the

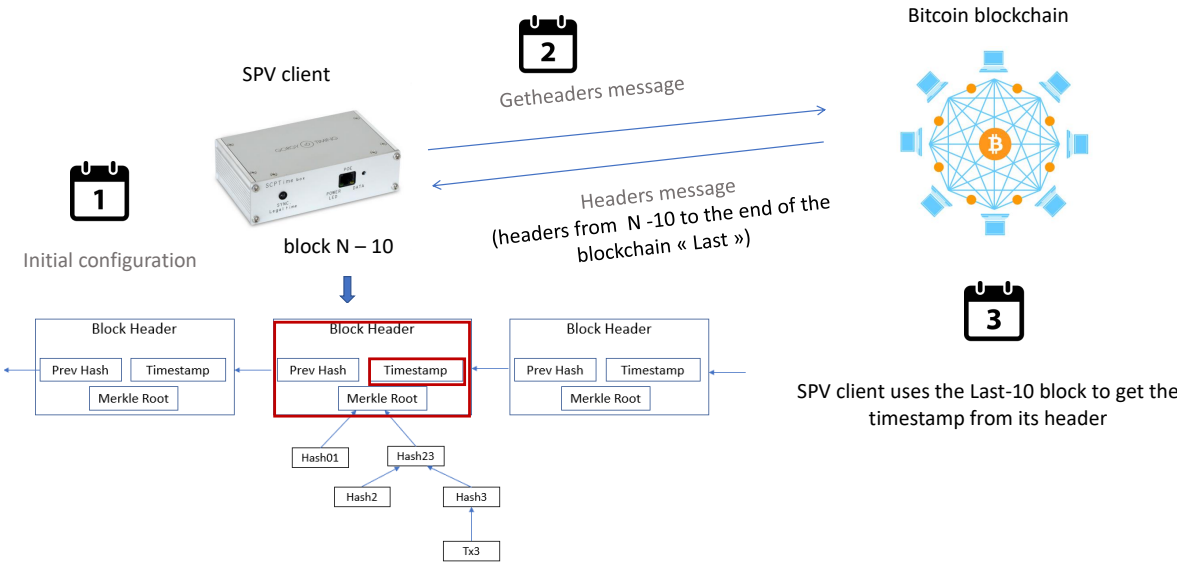



Figure 3.18 – Bootstrap Time Synchronization

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.

3. TC 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.

## 3.8 Conclusion

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. Finally, we have presented our solution for bootstrap-

ping time synchronization based on the Bitcoin blockchain to solve the problem of the circular dependency of time synchronization and public key authentication.

# Chapter 4

# Calibrating NTP

*"The bad news is time flies. The good news is you're the pilot"*

Michael Altshuler

## Contents

## 4.1 Introduction

In this chapter, we propose a method of improving the accuracy of NTP time synchronization by taking into account asymmetric transmission delays due to different bandwidth or routing on the forward and backward paths.

A common approach for clock offset estimation is the exchange of packets between hosts as performed by NTP. The two-way packet exchange with timestamps allows estimating the time offset between the client and the server. However, the accuracy of NTP time synchronization depends on the validity of the assumption related to symmetric transmission delays between the client and the server. If this assumption does not hold, which is a common case in the current Internet, the NTP synchronization scheme results in significant errors.

Our proposed method consists of:

- deploying a time box with a GPS clock at given client premises (we assume that the server has an accurate time source),
- calibrating—measuring the one-way transmission delay on the forward and backward path and finding the minimal delays,
- using the minimal delays in the estimation of the clock offset at the client to take into account path asymmetry,
- detecting changes in operating conditions to re-calibrate.

After calibration, the client does not longer use the time box and relies on NTP synchronization with a modified expression for the time offset. The proposed method requires periodical calibration—when the operating conditions (e.g., routing) change between the client and the server, which we can detect with the `ping` and `traceroute` tools, we need to redo calibration to find the new parameters of one-way transmission delays.

Unlike much work that validated their proposed schemes to improve the accuracy of time protocols with simulations, we perform measurement experiments to compare the clock offsets computed by standard NTP and calibrated NTP based on the GPS time reference.

## 4.2 NTP Assumptions and Notations

In this chapter, we adopt the standard NTP assumptions: the server has a perfect clock $C_s = t$ and the client wants to synchronize its clock $C_c = t + \theta$ with the server, $\theta$ being the time offset between the client and the server.

We denote forward (from the client to the server) and backward (from the server to the client) one-way transmission times: $T_f^j$ and $T_b^j$.

If we assume that the clock drift during the exchange is constant, we have the following relations:

$$t_2 = t_1 + T_f - \theta \tag{4.1}$$

$$t_4 = t_3 + T_b + \theta \tag{4.2}$$

If one-way transmission times are symmetric ($T_f = T_b$), the time offset becomes:

$$\theta = \frac{(t_2 - t_1) + (t_3 - t_4)}{2} \tag{4.3}$$

In general, one-way transmission times are asymmetric ($T_f \neq T_b$) and vary in time ($T_f^j \neq T_f^{j+1}$). In this case, the time offset becomes:

$$\theta = \frac{(t_2 - t_1) + (t_3 - t_4)}{2} + \frac{T_b - T_f}{2}, \tag{4.4}$$

Eq. 4.4 shows that the accuracy of NTP time synchronization depends on the difference of one-way transmission times so the assumption of symmetric one-way transmission times is the main source of accuracy errors. If we can estimate one-way transmission times in a more precise way, we can improve the accuracy of time synchronization.

## 4.3 Estimation of One-Way Transmission Times

As computer networks become more complex and larger, measuring methods and tools become essential to gather information on computer network performance. Many protocols and systems use these measurements to adjust their behavior to network conditions.

The most common delay metric is RTT defined by the IP Performance Metrics (IPPM) IETF group [101] as the time interval between the injection of the first bit of the packet into the network and the reception of the last bit of the packet, supposing the receiver resent the packet immediately after its reception. RTT measurements does not require synchronized clocks at the hosts since the sender computes RTT only based on its clock. RTT is an appropriate measure for many protocols and applications. For example, RTT is adequate for understanding network proximity in order to select the closest server.

However, RTT cannot capture *path asymmetry*, the fact that the path from a source to a destination (forward path) may differ from the path from the destination back to the source (backward path). Freris et al. [43] showed that asymmetry cannot be measured in a pairwise synchroniza-

tion system only based on recorded timestamps, even with an infinite number of round trip measurements. In addition, Pathak et al. [102] studied one-way transmission delays on the PlanetLab testbed and showed that asymmetry is quite prevalent. They also presented conclusive evidence that delay asymmetry is a dynamic property that varies depending on routing dynamics.

To solve this issue of path asymmetry, metrics such as One-Way Delay (OWD) [103] are the most suitable. OWD is the time interval between the injection of the first bit of the packet in the link and the reception of the last bit of the packet in the other measuring point. Since OWD depends on clocks on the two points, its measurement requires synchronized clocks at the sender and the receiver.

### 4.3.1 Measurements

We have set up an experiment to measure the one-way delays between a client and a server connected to different Autonomous Systems. The client is at the Gorgy Timing premises in Grenoble, France, and the server is at the Observatory of Paris, France. Both the client and the server locally connect to 100 Mb/s Ethernets. Gorgy Timing uses Orange as an ISP over a 1 Gb/s link and Observatory of Paris connects to Renater also over a 1 Gb/s link. Traceroute between the two end-points shows 14 intermediate routers on the forward and 11 routers on the backward path with an average RTT of 27 ms. The Internet connectivity between the client and the server has several interesting characteristics: there are several ASes on the path (3215 Orange, 5511 Opentransit, 3257 GTT, 2200 Renater), the end-points are connected to different types of ISPs (Orange: commercial provider, Renater: public network with large capacity), and there is an asymmetric number of routers.

The client runs Ledi Network ATS with ARM System On Chip (SOC) card that provides NTP timestamps over SNMP. The server is Meinberg M1000/MRS at the Observatory of Paris. The client and the server are synchronized with GPS with a precision of 50 ns in respect to UTC.

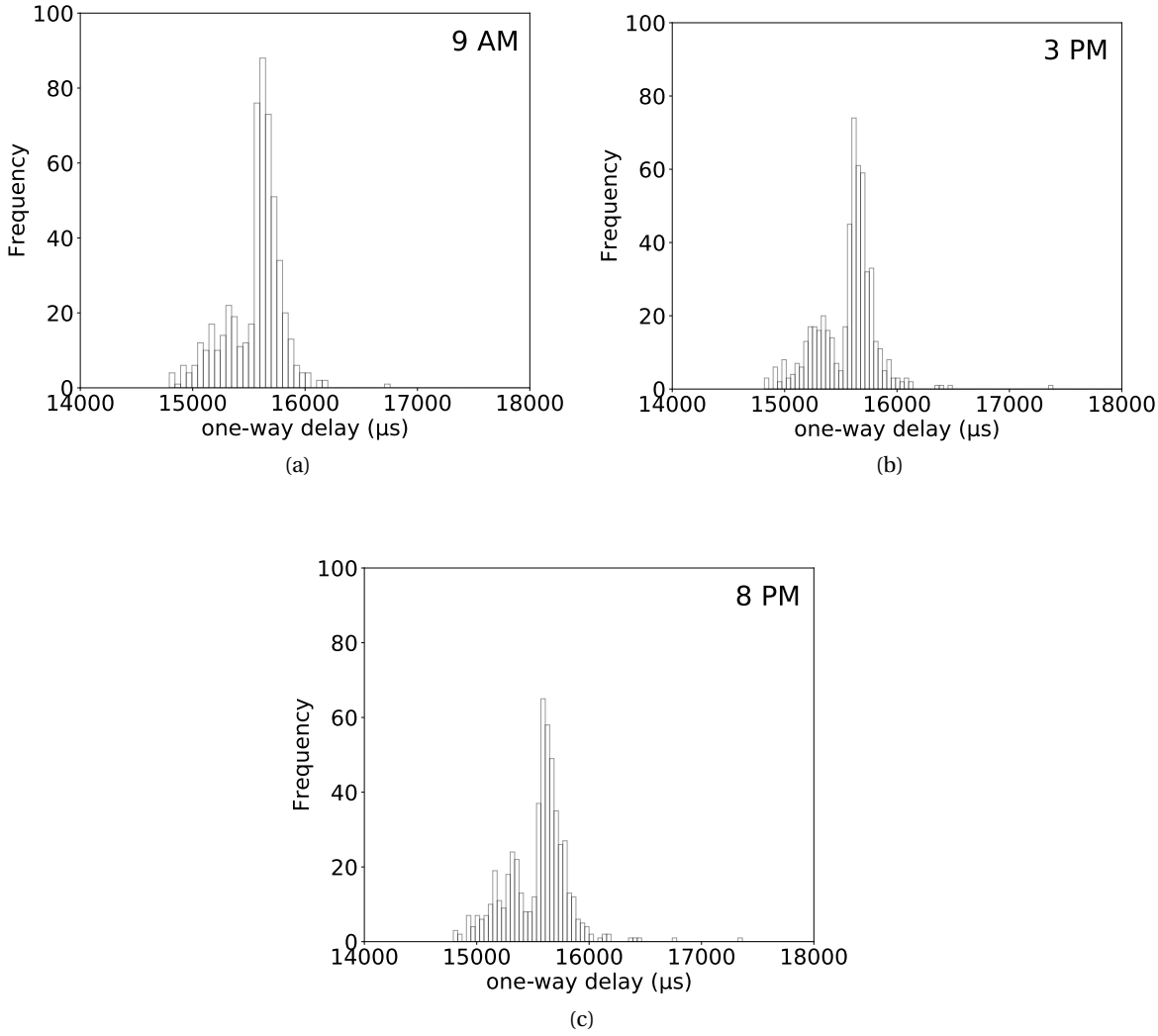
We have measured $T_f$ and $T_b$ (index $j$ skipped) at different hours of a day: 9AM, 3PM, and 8PM. Figures 4.1 and 4.2 show the histograms for each direction. We can notice that the one-way delays are highly asymmetric with the difference of around 3 ms. The shape of the distributions also varies: the variance of the forward distribution is much greater than that of the backward one.

We can observe that one-way delays $T_f$ and $T_b$ include a constant and a variable random part:

$$T_f = d_f^{min} + d_f, \tag{4.5}$$

$$T_b = d_b^{min} + d_b, \tag{4.6}$$

where $d_f^{min}$, $d_b^{min}$ are constant and $d_f$, $d_b$ are random variables. We denote the average values of the distributions as $\hat{T}_f$ and $\hat{T}_b$.




(a) (b) (c)

Figure 4.1 – Histograms of $T_f$

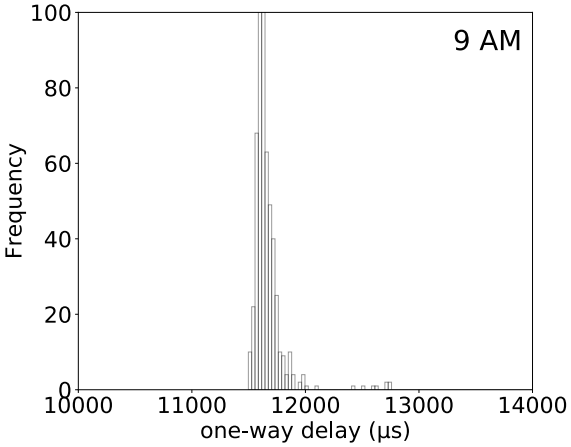
We can notice that the shape of the histograms is significantly different for forward and backward paths and the distributions do not vary much in time. To find the best fitting distributions of $T_f$ and $T_b$, we have tested several most important distributions: Gamma, Weibull, Normal, and Log Gamma. Figure 4.3 shows the results of fitting distributions at 9AM using the Maximum Likelihood Estimation (MLE) algorithm. For $T_f$, the best fitting distribution is a mixture of two Normal distributions:

$$p(x) = \lambda_1 \mathcal{N}(\mu_1, \sigma_1) + \lambda_2 \mathcal{N}(\mu_2, \sigma_2) \tag{4.7}$$

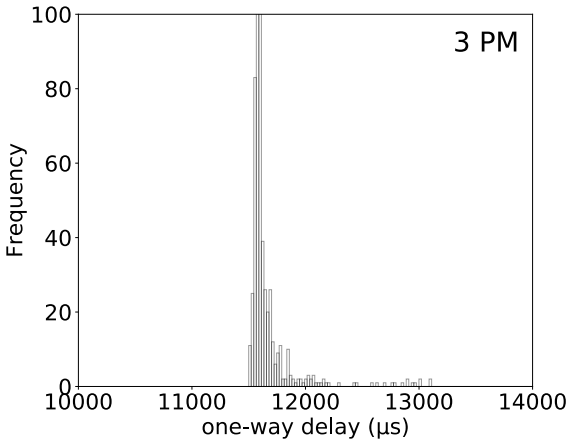
with $\lambda_1 = 0.5540935, \lambda_2 = 0.4459065$, the mean values are $\mu_1 = 15659.26, \mu 2 = 15465.70$, with standard variation of $\sigma_1 = 76.42764, \sigma_2 = 315.51818$. For $T_b$, we have fitted a Gamma distribution:

$$f(x; \alpha, \beta) = \frac{\beta^\alpha x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)}, \; x, \alpha, \beta > 0, \tag{4.8}$$

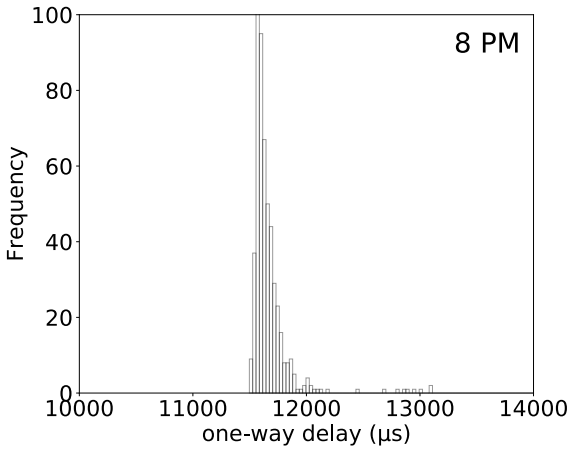
with the following parameters: shape $\alpha = 38521$ and rate $\beta = 3.3$, where $\Gamma(\alpha)$ denotes the Gamma




(a)

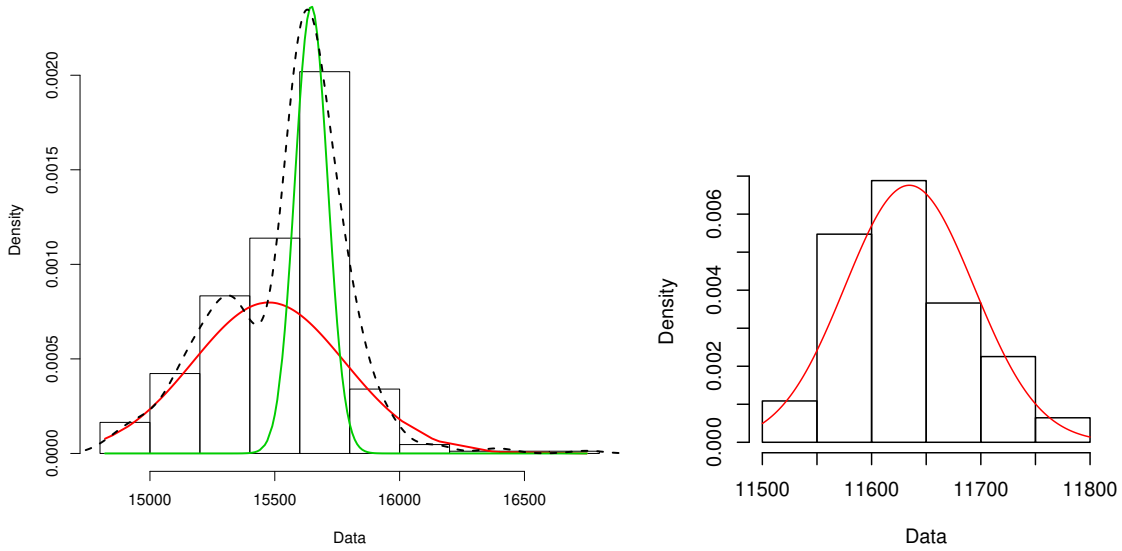



(b)




(c)

Figure 4.2 – Histograms of $T_b$




(a) $T_f$ – mixture of two Normal distributions

(b) $T_b$ – Gamma distribution

Figure 4.3 – Fitting the distribution of $T_f$ and $T_b$, 9AM

function.

## 4.4 Calibrating NTP

The goal of the calibration is to take advantage of the one-way delay measurements to mitigate the systematic error introduced by asymmetric paths in NTP time synchronization. Thus, we propose the following method for calibrating NTP:

1. *Calibration* – deploy the time box synchronized with GPS (like Ledi Network ATS) on given client premises. Measure the distributions of $T_f$ and $T_b$ for a given NTP server and find their $d_f^{min}$ and $d_b^{min}$.

2. *Regular operation* – replace the time box with an NTP client without GPS for time synchronization. The client will operate according to the NTP protocol with a modified way of computing the time offset.

3. *Change detection* – detect changes in operating conditions with the `ping` and `traceroute` tools and redo calibration.

We assume that the NTP client sends $n$ NTP requests and obtains $n$ responses with corresponding timestamps. We find packets that experience the shortest transmission delays (called *lucky packets*) presented by 4.9 and 4.10, and use their timestamps for computing the time offset.

$$j_f = \arg\min_j (t_2^j - t_1^j), \tag{4.9}$$

$$j_b = \arg\min_j (t_4^j - t_3^j), j = 1, ..., n, \tag{4.10}$$

Finally, we use the estimates of minimum transmission times $d_f^{min}$ and $d_b^{min}$ in Eq. 4.4 for the time offset with the timestamps of packets $j_f$ and $j_b$:

$$\theta = \frac{(t_2^{j_f} - d_f^{min} - t_1^{j_f}) + (t_3^{j_b} + d_b^{min} - t_4^{j_b})}{2}. \tag{4.11}$$

This expression corresponds to Eq. 4.4 with $T_f^j$ and $T_b^j$ replaced by $d_f^{min}$ and $d_b^{min}$.

## 4.5 Validation

The validation of our method is done in the same operating conditions as the estimation of OWD. To evaluate the improvement in accuracy of the proposed method, we compare the accuracy of time synchronization obtained by standard NTP given by Eq. 4.3 with proposed calibrated NTP measured with respect to the GPS time reference—we deploy the NTP client with the modified way of computing the time offset (Eqs. 4.9, 4.10, and 4.11) and compare the accuracy of its estimation with GPS.

The experiments took place at a given time of a day (9AM, 3PM, 8PM) with the measurement phase of one-way delay for 15 minutes during which the client sends trains of 8 NTP requests every 10 s followed by the measurements of the modified client with calibrated NTP during 15 minutes. Finally, we measured the performance of standard NTP also during 15 minutes. We repeated the experiments for several days and observed similar results.

Figures 4.4, 4.5, and 4.6 show the histograms of the time offset for standard NTP (left) and calibrated NTP (right) at 9AM, 3PM, and 8PM. The figures present *accuracy* measured by $\mu$, the mean value of the distribution and *precision* measured by $\sigma$, the standard deviation. We can observe significant elimination of the accuracy error due to asymmetry of one-way delays and improved precision (lower $\sigma$).

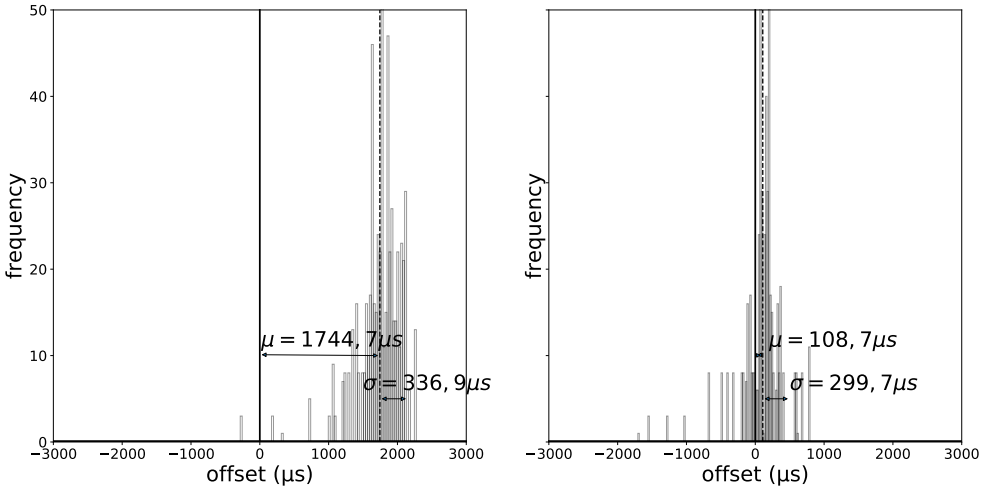



Figure 4.4 – Histograms of the time offset for standard NTP (left) and calibrated NTP (right), at 9AM.

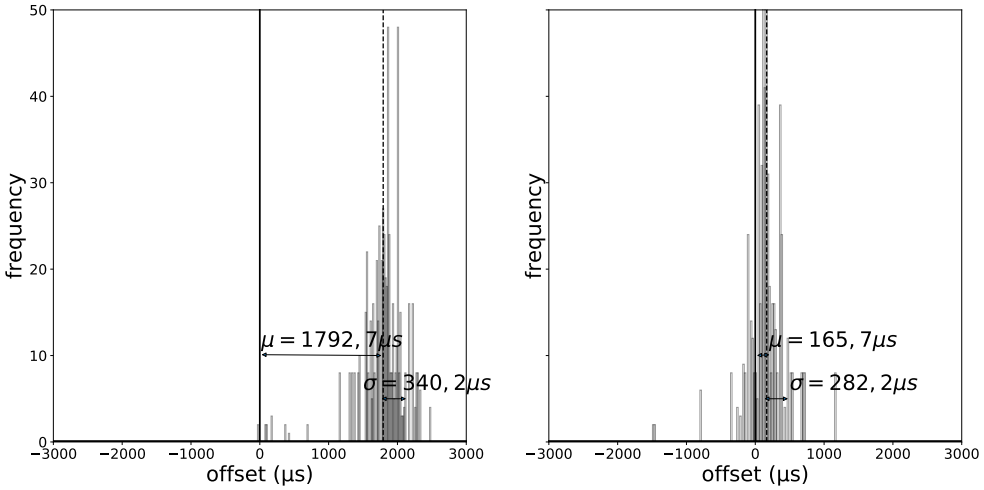



Figure 4.5 – Histograms of the time offset for standard NTP (left) and calibrated NTP (right), 3PM.

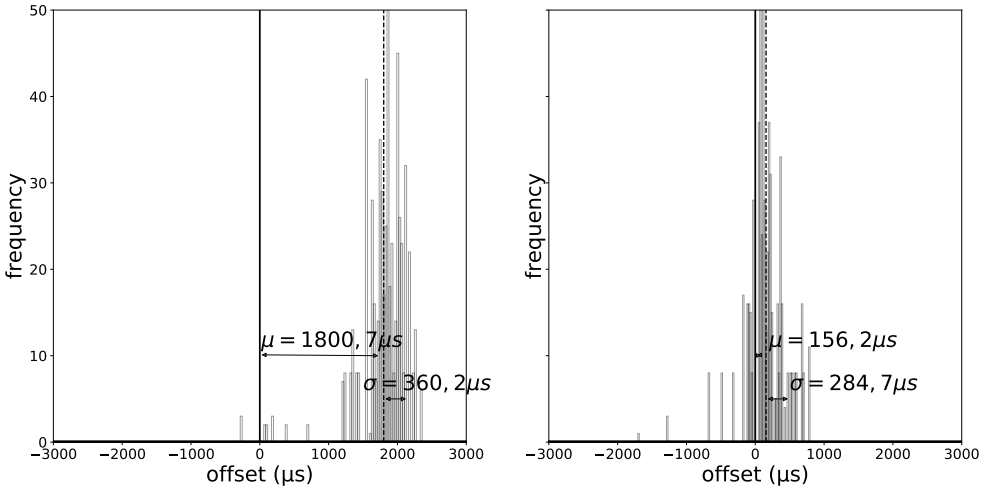



Figure 4.6 – Histograms of the time offset for standard NTP (left) and calibrated NTP (right), 8PM.

## 4.6 Conclusion

In this chapter, we propose to calibrate NTP with the measurements of one-way transmission delay on the forward and backward path. Then, in the set up without the precise clock at the client, the NTP expression for the time offset takes into account asymmetry, which results in improved accuracy and precision.

Unlike many papers that use simulations, we have validated the proposed method by measurements of the clock offsets computed by standard NTP and calibrated NTP based on the GPS time reference showing significant improvement in accuracy and precision.

# Chapter 5

# Conclusion

*"We must use time wisely and forever realize that the time is always ripe to do right."*

Nelson Mandela

The work presented in this thesis was conducted in the context of the SCPTime project. SCPTime requires a new secure and precise variant of NTP to securely disseminate the legal time of the country to final users.

In the first part of this thesis, we studied the state of the art of time synchronization. This study helped us to understand some fundamental concepts and notions of time synchronization. We identified the most common time synchronization protocols used in packet-switched networks. We also pointed out the impact of delay asymmetry that exists in current networks on the time synchronization accuracy. In fact, in the Internet, asymmetry is quite prevalent due to some traffic configuration practices to minimize resource consumption. We presented related work that tried to deal with the asymmetry issue.

Then, we studied the security of time synchronization by presenting the possible security threats against time synchronization protocols and the security requirements for these protocols. We particularly studied the security of NTP and noticed that NTP lacks robust security mechanisms. This study helped us to be aware of possible attacks against time synchronization protocols and to take into account the requirements in the design of new secure variant of NTP.

During the previous study, we noticed a circular dependency between certificate validation and time synchronization. In fact, reliable time synchronization requires cryptographic materials that are valid only over designated time intervals, but time intervals can be only enforced when participating servers and clients are reliably synchronized. We described some lightweight proto-

cols proposed to provide rough time synchronization to break this circular dependency.

## Summary of Contributions

The second part of the thesis presented our contributions. They aim to improve the security and accuracy of NTP.

**Chapter 4** presents our first contribution. We proposed the STS protocol that enables client and server authentication, supports the property of non-repudiation using digital signature, and offloads the negotiation and authorization phases to an authorization server to keep the time server lightweight and available to respond to clients. The authorization server checks for authorizations and provides the required cryptographic material to clients and servers over DTLS sessions. We decided to use DTLS instead of TLS because NTP is a datagram protocol and because DTLS requires minimum processing overhead.

During STS time synchronization phase, 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 have analyzed the main security properties of STS with two security protocol verification tools: Proverif and Tamarin. This formal analysis proved that our STS design is secure. In fact, we proved the secrecy property of the used keys $K$ and $S$. The time synchronization authenticity and integrity were proved for the messages $m_2$ (authenticated request) and $m_4$ (authenticated response). A "race condition" attack was discovered by Proverif tool, this attack is performed by an attacker that replayed the same message $m_2$ with the same nonce in two instances of the time server while the nonce checking in each instance is performed before inserting the new nonce in the cache. To avoid "race condition attack", we proposed to restrict the "write access" to the cache to only authorized processes and to protect the nonce cache using mutual exclusion.

As mentioned in STS operation, STS server sends two responses, the first being the unauthenticated NTP response $m_4$, and the second being the same NTP response along with an authentication tag (MAC) ensuring that $m_3$ was not modified in transit. This "post-verification" method aims to avoid the degradation of time synchronization accuracy by the computation of the MAC: the client measures the roundtrip time based on the unauthenticated response, but does not update its clock until authenticating the response. The formal analysis showed that sending an unauthenticated NTP response to the client can cause a DDOS attack on the client side. To prevent this kind of attack, the client should use common measures based in intrusion detection/prevention systems.

Then, we implemented the protocol based on OpenNTPd. The first evaluation of the overhead of the most time-critical operations shows that the chosen cryptographic primitives for MAC and DS generation introduce little overhead, which contributes to the capacity of the time server to accommodate a large number of clients.

Finally, we proposed 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 Bitcoin blockchain as the basis for the approximate time. The unsynchronized client acts as an SPV lightweight blockchain client that discovers peers in the Bitcoin P2P network to request the headers until reaching the end of the blockchain to get the timestamp of the last headers. This timestamp represents the rough time used by the client to get a rough synchronization. To avoid any risk of MITM or masquerade attacks, the client need to repeat this process with several peeres to check whether the returned headers correspond to the Bitocin blockchain.

**Chapter 5** details our second contribution. We showed that the accuracy of NTP time synchronization depends on the validity of the assumption related to symmetric transmission delays between the client and the server. In the current Internet, this assumption does not hold, which results in lower accuracy/precision in the NTP synchronization scheme. To solve this issue, we proposed to calibrate NTP with the measurements of one-way transmission delay on the forward and backward path. In fact, a client has a time box synchronized with GPS and exchanges NTP packets with an NTP server with a very precise clock. Using the NTP exchange, client and server measure precisely the OWD on the forward and backward path and find the minimum delays. The client uses these measured minimum delays in the computation of the time offset. In this way, it takes into account asymmetry, which results in improved accuracy and precision. This method requires re-calibration to find the new parameters when operating conditions change. This modification in operating conditions can be detected with the *ping* and *Traceroute* tools . We have validated the proposed method by measurements of the clock offsets computed by standard NTP and calibrated NTP based on the GPS time reference showing significant improvement in accuracy and precision.

## Future Work

This work also allowed us to identify compelling future research directions. We list these perspectives:

STS has been integrated in different Gorgy Timing devices used in SCPTime. In the future, we plan to measure the precision of STS over long distances and during longer time period because the presented results was done in a LAN condition and over 1 minute period. We also plan to measure how many simultaneous STS clients, the STS server can handle. To do this test, SCPTime collaborates with University of Grenoble. The idea is to synchronize the student's computers to a Biatime B provided by SCPTime to evaluate the load increase on the server.

In "calibrating NTP", we validate the method in short periods of time. In the future, we plan to measure the stability of paths between a client and a server over longer time intervals to evaluate how frequently routing changes impact the operation of calibrated NTP.

In our work, the calibration of NTP client is done with only one server. So, when the operating condition change between the client and the server, we must re-calibrate to estimate the new asymmetry. However, if initially the calibration is done with multiple servers, if routes change to the first server we can still rely on the other calibrated paths measured with the rest of servers, that did not experience changes on operating conditions. So, we can still use them as backups for calibration. In this way, we keep good NTP accuracy, and we have enough time to re-calibrate the paths with the first server if needed.

This work highlights the importance of having a secure and precise time synchronization in a growing digital world. Our work, focused on NTP but PTP protocol deserves also some improvements of its security because its initial design lacks efficient security solutions.

# Appendix A

# Appendix Example

---

## A.1 Proverif Code

```
free c:channel.

free S:bitstring [private].
free K:bitstring [private].

(* key secrecy *)
query attacker(K);
      attacker(S).

(* types - type converters*)
type int.
fun int_bitstring(int):bitstring.
fun bitstring_int(bitstring):int.
equation forall x:bitstring; int_bitstring(bitstring_int(x)) = x.
equation forall x:int; bitstring_int(int_bitstring(x)) = x.

type key.
fun bitstring_key(bitstring):key.
fun key_bitstring(key):bitstring.
equation forall x:bitstring; key_bitstring(bitstring_key(x)) = x.
```

```
equation forall x:key; bitstring_key(key_bitstring(x)) = x.

type mkey.
fun bitstring_mkey(bitstring):mkey.
fun mkey_bitstring(mkey):bitstring.
equation forall x:bitstring; mkey_bitstring(bitstring_mkey(x)) = x.
equation forall x:mkey; bitstring_mkey(mkey_bitstring(x)) = x.

(* cryptographic functions*)
fun senc(bitstring, key):bitstring.
fun sdec(bitstring, key):bitstring.
equation forall m:bitstring, k:key; sdec(senc(m, k), k) = m.
equation forall m:bitstring, k:key; senc(sdec(m, k), k) = m.

fun mac(bitstring, mkey):bitstring.

(* events *)
event sentClient(bitstring, bitstring).
event received1Client(bitstring, bitstring, bitstring).
event received2Client(bitstring, bitstring, bitstring, bitstring).
event sent1Server(bitstring, bitstring, bitstring).
event sent2Server(bitstring, bitstring, bitstring, bitstring).
event receivedServer(bitstring, bitstring).
```

(* all messages' authentication and integrity*)

query x:bitstring, y: bitstring; inj-event(receivedServer(x, y)) ==> inj-event(sentClient(x, y)).

query x:bitstring, y: bitstring, z: bitstring; inj-event(sent1Server(x, y, z)) ==> inj-event(receivedServer(x, y)).

query x:bitstring, y: bitstring, z: bitstring; inj-event(received1Client(x, y, z)) ==> inj-event(sent1Server(x, y, z)).

query x:bitstring, y: bitstring, z: bitstring, w:bitstring; inj-event(sent2Server(x, y, z, w)) ==> inj-event(sent1Server(x, y, z)).

query x:bitstring, y: bitstring, z: bitstring, w:bitstring; inj-event(received2Client(x, y, z, w)) ==> inj-event(sent2Server(x, y, z, w)).

query x:bitstring, y: bitstring, z: bitstring, w:bitstring; inj-event(received2Client(x, y, z, w)) ==>

inj-event(received1Client(x, y, z)) .

```
(* cache handling*)
type bset.

fun consset(bitstring, bset): bset [data].

const emptyset: bset [data].

pred mem(bitstring, bset).
clauses
  forall x:bitstring, y:bset; mem(x, consset(x, y));
  forall x:bitstring, y:bset, z:bitstring; mem(x, y) -> mem(x, consset(z, y)).

(* cache table stored by server*)
table cache_nonce(bitstring, bset).

(* events of nonce usage*)
event fresh_nonce.
event rogue_nonce.

event valid_m1.
event rogue_m1.

(* processes *)
let TimeClient(C_c:bitstring) =
  new n:bitstring;
  new id_T1: bitstring;
  let m1 = (id_T1, n, C_c) in
  let tag = mac(m1, bitstring_mkey(K)) in
  let m11 = (m1, tag) in
  out(c, m11);
  event sentClient(C_c, m11);
  in(c, (id_T1':bitstring, T2:bitstring, T3:bitstring));
  if id_T1' = id_T1 then
```

```
  let m3 = (id_T1', T2, T3) in
  event received1Client(C_c, m11, m3);
  new T4:bitstring;
  in(c, m4:bitstring);
  if m4 = mac((m1, m3), bitstring_mkey(K)) then (
  event received2Client(C_c, m11, m3, m4)).

let TimeServer() =
  in(c, (id_T1:bitstring, n:bitstring, C_s:bitstring, tag:bitstring));
  new cache:bset;
  get cache_nonce(C_s, =cache) in
  if mem(n, cache) then
    (* replayed nonce, TimeServer process terminates*)
    event rogue_nonce
  else (
    (* fresh nonce added to cache associated with C_s*)
    event fresh_nonce;
    insert cache_nonce(C_s, consset(n, cache));
    let m1 = (id_T1, n, C_s) in
    (* T2 guessable*)
    in(c, T2:bitstring);
    let K_s = sdec(C_s, bitstring_key(S)) in (
      if tag = mac(m1, bitstring_mkey(K_s)) then (
        event valid_m1;
        let m11 = (m1, tag) in
        event receivedServer(C_s, m11);
        (* T3 guessable*)
        in(c, T3:bitstring);
        let m3 = (id_T1, T2, T3) in
        out(c, m3);
        event sent1Server(C_s, m11, m3);
        let m4 = mac((n, m3), bitstring_mkey(K_s)) in
        out(c, m4);
        event sent2Server(C_s, m11, m3, m4)
      )
```

```
      else (
        event rogue_m1
      )
    )
    else (
      event rogue_m1
    )
  ).

process
  let C = senc(K, bitstring_key(S)) in
  ((!TimeClient(C)) | (!TimeServer()))
```

## A.2 Tamarin Code

```
theory STSv5
begin

builtins: symmetric-encryption, hashing

// setteing up S
rule Setup_S:
    [ Fr(~S) ]
  --[  ]->
    [ !Setup_S(~S) ]

// compromising key S
rule RevealS:
    [ !Setup_S(S) ]
  --[ S_Reveal(S) ]->
    [ Out(S) ]

// setting up C
rule Setup_C:
    [ !Setup_S(S)
    , Fr(~K) ]
  --[ ]->
    [ !Setup_C(~K, senc(~K, S)) ]

// compromising key K
rule RevealK:
    [ !Setup_C(K, C) ]
  --[ K_Reveal(K) ]->
    [ Out(K) ]

/* --------------synchronization phase - STS Protocol
   -------------------- */
```

```
// start a new thread sending m1
rule Client_m1:
    [ !Setup_C(K, C)
    , Fr(~n)
    , Fr(~T1)]
  --[ Sent_m1(~n, K, C)
    , SendRequest(~n, K, C)
    , FreshNonce(~n) ]->
    [ Client_m1(~T1, ~n, C, K)
    , Out(<~T1, ~n, C, h(<~T1, ~n, C, K>)>) ]

// start a new thread receiving m3
rule Client_m3:
    [ Client_m1(T1, n, C, K)
      , In(<T11, T2, T3>)]
  --[ Eq(T11, T1)
    , Received_m3(n, K, C) ]->
    [ Client_m3(n, C, K, T1, T2, T3) ]

// start a new thread receiving m4
rule Client_m4:
    [ Client_m3(n, C, K, T1, T2, T3)
    , In(m4)]
  --[ Eq(h(<n, T1, T2, T3, K>), m4)
      , Received_m4(n, K, C)
      , ServerAuth(K, C)
      , ClientTerm(n, K, C) ]->
    [ ]

// start a new thread receiving m1
rule Server_m1:
    [ !Setup_S(S)
    , In(<T1, n, C, tag>) ]
  --[ Eq(h(T1, n, C, sdec(C, S)), tag)
```

```
    , Received_m1(n, sdec(C, S), C)
    , ClientAuth(sdec(C, S), C) ]->
    [ Server_m1(T1, n, sdec(C, S), C) ]

// start a new thread sending m3
rule Server_m3:
    [ Server_m1(T1, n, K, C)
    , Fr(~T2)
    , Fr(~T3)]
  --[ Sent_m3(n, K, C)
    , SendReply(n, K, C) ]->
    [ Server_m3(n, T1, ~T2, ~T3, K, C)
    , Out(<T1, ~T2, ~T3>) ]

// start a new thread sending m4
rule Server_m4:
    [ Server_m3(n, T1, T2, T3, K, C) ]
  --[ Sent_m4(n, K, C)
    , SendReply_Auth(n, K, C)
    , ServerTerm(n, K, C) ]->
    [ Out(h(<n, T1, T2, T3, K>)) ]

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

/* -----------desired properties ------------- */

// nonce usage
lemma distinct_nonces:
  "All n #i #j. FreshNonce(n)@i & FreshNonce(n)@j ==> #i=#j"

// Secrecy of S and K

lemma S_secrecy:
```

```
  " /* It cannot be that a  */
    not(
      Ex K S n #i #j.
        /* client has set up a session key 'K' with a server
           with key 'S' */
        ClientTerm(n, K, senc(K, S)) @ #i
        /* and the adversary knows 'S' */
      & K(S) @ #j
        /* without having performed a long-term key reveal on '
           S'. */
      & not(Ex #r. S_Reveal(S) @ r)
    )
  "

lemma K_secrecy:
  " /* It cannot be that a  */
    not(
      Ex K S n #i #j.
        /* client has set up a session key 'K' with a server
           with key 'S' */
       ClientTerm(n, K, senc(K, S)) @ #i
        /* and the adversary knows 'K' */
      & K(K) @ #j
        /* without having performed a long-term key reveal on '
           K' and 'S'. */
      & not(Ex #r. S_Reveal(S) @ r)
      & not(Ex #b. K_Reveal(K) @ b)
    )
  "

// client and server authentication

lemma Client_inj_Auth:
  " /* For all session keys 'K' setup by clients with a server
     with key 'S' */
```

```
    ( All K S n #i. ClientTerm(n, K, senc(K, S)) @ #i
       ==>
         /* there is a server that answered the request */
       ( (Ex #a. SendReply(n, K, senc(K, S)) @ a
        /* and there is no other client that had the same
           request */
        & (All #j. ClientTerm(n, K, senc(K, S)) @ #j ==> #i = #
           j)
        )
         /* or the adversary performed a long-term key reveal
            on 'S' or 'K'
            before the key was setup. */
       | (Ex #r. K_Reveal(K) @ r & r < i)
       | (Ex #b. S_Reveal(S) @ b & b < i)
       )
    )
  "

lemma Server_Auth:
  " /* For all session keys 'K' setup by a server with a client
      */
    ( All K S n #i. ServerTerm(n, K, senc(K, S)) @ #i
       ==>
         /* there is a client that sent the request */
       ( (Ex #a. SendRequest(n, K, senc(K, S)) @ a
        /* and there is no other server that had the same
           request */
        //& (All #j. ServerTerm(n, K, senc(K, S)) @ #j ==> #i =
            #j)
        )
         /* or the adversary performed a long-term key reveal
            on 'S' or 'K'
            before the key was setup. */
       | (Ex #r. K_Reveal(K) @ r & r < i)
       | (Ex #b. S_Reveal(S) @ b & b < i)
```

```
        )
    )
  "

// messages' integrity evaluation

lemma m1_integrity:
  "
        ( All K S n #i. Received_m1(n, K, senc(K, S)) @ #i
       ==>
       ( (Ex #a. Sent_m1(n, K, senc(K, S)) @ a
        )
       | (Ex #r. K_Reveal(K) @ r & r < i)
       | (Ex #b. S_Reveal(S) @ b & b < i)
       )
    )
  "

lemma m3_integrity:
  "
        ( All K S n #i. Received_m3(n, K, senc(K, S)) @ #i
       ==>
       ( (Ex #a. Sent_m3(n, K, senc(K, S)) @ a
        )
       | (Ex #r. K_Reveal(K) @ r & r < i)
       | (Ex #b. S_Reveal(S) @ b & b < i)
       )
    )
  "

lemma m3_after_m1:
  "
        ( All K S n #i. Sent_m3(n, K, senc(K, S)) @ #i
       ==>
       ( (Ex #a. Received_m1(n, K, senc(K, S)) @ a
```

```
        )
      | (Ex #r. K_Reveal(K) @ r & r < i)
      | (Ex #b. S_Reveal(S) @ b & b < i)
      )
    )
  "

lemma m4_integrity:
  "
       ( All K S n #i. Received_m4(n, K, senc(K, S)) @ #i
      ==>
      ( (Ex #a. Sent_m4(n, K, senc(K, S)) @ a
       )
      | (Ex #r. K_Reveal(K) @ r & r < i)
      | (Ex #b. S_Reveal(S) @ b & b < i)
      )
    )
  "

lemma m4_after_m3:
  "
       ( All K S n #i. Sent_m4(n, K, senc(K, S)) @ #i
      ==>
      ( (Ex #a. Sent_m3(n, K, senc(K, S)) @ a
       )
      | (Ex #r. K_Reveal(K) @ r & r < i)
      | (Ex #b. S_Reveal(S) @ b & b < i)
      )
    )
  "

end
```

# Bibliography


[1] J. Ridoux and D. Veitch, "A Methodology for Clock Benchmarking," in *2007 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, pp. 1–10, May 2007. 9, 28

[2] J. Vig, "Introduction to Quartz Frequency Standards. Revision," *NASA STI/Recon Technical Report N*, vol. 93, p. 15330, 09 1992. 9, 31

[3] J. Vig, "Introduction to Quartz Frequency Standards. Revision," *NASA STI/Recon Technical Report N*, vol. 93, p. 15330, 09 1992. 9, 31, 32

[4] B. D. Esham, "Network Time Protocol Servers and Clients." [Online]. Available from: https://en.wikipedia.org/wiki/Network_Time_Protocol#/media/File:Network_Time_Protocol_servers_and_clients.svg. 9, 37

[5] Wikipédia, "SyncE for mobile Networks." [Online]. Available from: https://fr.wikipedia.org/wiki/Synchronous_Ethernet#/media/Fichier:Synce-network-wless.png. 9, 46, 47

[6] T. Włostowski, "Precise time and frequency transfer in a White Rabbit network," Master's thesis, Warsaw University of Technology, 2011. 9, 48

[7] J. Ridoux and D. Veitch, "Principles of Robust Timing over the Internet," *Commun. ACM*, vol. 53, pp. 54–61, 05 2010. 9, 49

[8] Cloudflare, "Cloudflare's implementation of NTS in Rust." [Online]. Available from: https://blog.cloudflare.com/announcing-cfnts/. 9, 60

[9] 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*, pp. 1–11, 2017. 11, 88

[10] Emissions-EUETS, "Timestamping and Business Clocks Synchronization under MiFID II." [Online]. Available from: https://www.emissions-euets.com/time-stamping-and-business-clocks-synchronisation. 17

[11] G. X. Xi Fang, Satyajayant Misra and D. Yang, "Smart Grid — The New and Improved Power Grid: A Survey," 2012. 17

[12] TSN-Working-Group, "Time-Sensitive Networking (TSN) Task Group." [Online]. Available from: https://1.ieee802.org/tsn/. 17

[13] NIST, "CPS Public Working Group." [Online]. Available from: https://pages.nist.gov/cpspwg/. 18

[14] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir, "Taming the 800 pound gorilla: The rise and decline of NTP DDoS attacks," *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, pp. 435–448, 11 2014. 18

[15] SCPTime, "SCPTime Project." [Online]. Available from: https://www.scptime.com/en/. 19

[16] GorgyTiming, "Gorgy Timing." [Online]. Available from: http://www.gorgy-timing.co.uk/. 19

[17] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," May 2011. 21, 72, 93

[18] ProVerif, "Automatic Cryptographic Protocol Verifier, User Manual and Tutorial." [Online]. Available from: http://prosecco.gforge.inria.fr/personal/bblanche/proverif/. 23, 69, 73

[19] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN Prover for the Symbolic Analysis of Security Protocols," in *Computer Aided Verification* (N. Sharygina and H. Veith, eds.), (Berlin, Heidelberg), pp. 696–701, Springer Berlin Heidelberg, 2013. 23, 69, 74

[20] B. Cook, "OpenNTPd Portable Implementation." [Online]. Available from: https://github.com/openntpd-portable/openntpd-portable. 23, 69, 87

[21] International-Telecommunication-Union, "Definitions and Terminology for Synchronization in Packet Networks." [Online]. Available from: https://www.itu.int/rec/T-REC-G.8260-201508-I/fr. 31

[22] J. Postel and K. Harrenstien, "Time Protocol," RFC 868, may 1983. 33

[23] J. Postel, "Daytime Protocol," RFC 867, May 1983. 33

[24] D. Mills, J. Martin, and J. Burbank, "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC 5905, 2010. 35, 57

[25] P. D. L. Mills, "Simple Network Time Protocol (SNTP)." RFC 1769, Mars 1995. 35

[26] T. Schmid, Z. Charbiwala, R. Shea, and M. B. Srivastava, "Temperature Compensated Time Synchronization," *IEEE Embedded Systems Letters*, vol. 1, pp. 37–41, Aug 2009. 39

[27] S. Johannessen, "Time Synchronization in a Local Area Network," *IEEE Control Systems Magazine*, vol. 24, pp. 61–69, April 2004. 40

[28] L. Chao, J. Han-Hong, and B. Kai, "Time Synchronization Method of Power Monitoring Networks Based on Filtrating Path Delay Filtration," in *2014 Fourth International Conference on Instrumentation and Measurement, Computer, Communication and Control*, pp. 341–345, Sep 2014. 40

[29] Y.-J. Quan, G.-X. Liu, B. Liu, and X.-B. Hong, "A RCR Clock Synchronization Model with Drift Compensation," vol. 38, pp. 76–79+85, 05 2010. 40

[30] H.-T. Zhu, W. Ding, and R.-R. Lin, "Network Clock Synchronization based on Ring Topology," vol. 36, pp. 80–85, 04 2008. 40

[31] H. Zhang, D.-Q. Feng, and J. Chu, "Industrial Ring Network Time Synchronization based on Waydelay Weighted Feedback Algorithm," vol. 44, pp. 849–853, 05 2010. 40

[32] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat, "Exploiting a Natural Network Effect for Scalable, Fine-grained Clock Synchronization," in *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, pp. 81–94, 2018. 40

[33] International-Telecommunication-Union, "Timing and Synchronization Aspects in Packet Networks." [Online]. Available from: https://www.itu.int/rec/T-REC-G.8261-201908-I/en. 46

[34] International-Telecommunication-Union, "Timing Characteristics of a Synchronous Equipment Slave Clock." [Online]. Available from: https://www.itu.int/rec/T-REC-G.8262-201811-I/en. 46

[35] International-Telecommunication-Union, " Distribution of Timing Information Through Packet Networks." [Online]. Available from: https://www.itu.int/rec/T-REC-G.8264-201708-I/en. 46

[36] CERN, "The White Rabbit Project." [Online]. Available from: https://white-rabbit.web.cern.ch/. 47

[37] CERN, "The White Rabbit Project." [Online]. Available from: https://home.cern/fr. 47

[38] J. Ridoux and D. Veitch, "The Cost of Variability," in *International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS 08)*, (Ann Arbor, Michigan, USA), pp. 29–32, Sep 22-26 2008. 49

[39] D. Veitch, J. Ridoux, and S. B. Korada, "Robust Synchronization of Absolute and Difference Clocks over Networks," *IEEE/ACM Transactions on Networking*, vol. 17, pp. 417–430, April 2009. 49

[40] J. Ridoux and D. Veitch, "Ten Microseconds Over LAN, for Free," in *International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS 07)*, (Vienna, Austria), pp. 105–109, Oct 1-3 2007. 49

[41] J. Ridoux, D. Veitch, and T. Broomhead, "The Case for Feed-Forward Clock Synchronization," *IEEE/ACM Transactions on Networking*, vol. 20, pp. 231–242, February 2012. 49

[42] M. Tsuru *et al.*, "Estimation of Clock Offset from One-Way Delay Measurement on Asymmetric Paths," in *Proceedings of the 2002 Symposium on Applications and the Internet (SAINT) Workshops*, 2002. 49, 50

[43] N. M. Freris, S. R. Graham, and P. R. Kumar, "Fundamental Limits on Synchronizing Clocks over Networks," *IEEE Transactions on Automatic Control*, vol. 56, pp. 1352–1364, June 2011. 50, 99

[44] M. Lévesque and D. Tipper, "A Survey of Clock Synchronization Over Packet-Switched Networks," *IEEE Communications Surveys Tutorials*, vol. 18, pp. 2926–2947, Fourthquarter 2016. 50

[45] R. Zarick, M. Hagen, and R. Bartoš, "The Impact of Network Latency on the Synchronization of Real-World IEEE 1588-2008 Devices," in *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pp. 135–140, Sep. 2010. 50

[46] S. Lee, "An Enhanced IEEE 1588 Time Synchronization Algorithm for Asymmetric Communication Link using Block Burst Transmission," *IEEE Communications Letters*, vol. 12, pp. 687–689, Sep. 2008. 50

[47] S. Schriegel, H. Trsek, and J. Jasperneite, "Enhancement for a Clock Synchronization Protocol in Heterogeneous Networks," in *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pp. 1–5, Oct 2009. 50

[48] T. Murakami and Y. Horiuchi, "Improvement of Synchronization Accuracy in IEEE 1588 Using a Queuing Estimation Method," in *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pp. 1–5, Oct 2009. 50

[49] R. Exel, "Mitigation of Asymmetric Link Delays in IEEE 1588 Clock Synchronization Systems," *IEEE Communications Letters*, vol. 18, pp. 507–510, March 2014. 51

[50] M. Lévesque and D. Tipper, "Improving the PTP Synchronization Accuracy under Asymmetric Delay Conditions," in *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pp. 88–93, Oct 2015. 51

[51] M. J. Hajikhani, T. Kunz, and H. Schwartz, "A Recursive Method for Clock Synchronization in Asymmetric Packet-Based Networks," *IEEE/ACM Transactions on Networking*, vol. 24, pp. 2332–2342, Aug 2016. 51

[52] G. Daniluk, "White Rabbit Calibration Procedure, Version 1.1." CERN BE-CO-HT, November 9, 2015. 51

[53] A. J.Hodges, C. Jackson, "HTTP Strict Transport Security (HSTS)," RFC 6797, November 2012. 54

[54] J. Selvi, "Bypassing HTTP Strict Transport Security," in *Black Hat Europe,*, October 2014. 54

[55] A. Malhotra, IE. Cohen, E. Brakke and S. Goldberg , "Attacking the Network Time Protocol," in *ISOC Network and Distributed System Security Symposium NDSS 2016*, 2016. 54, 69

[56] D. Mills, "Security Requirements of Time Protocols in Packet Switched Networks," RFC 7384, 2014. 55, 56, 61

[57] A. Malhotra, M. Van Gundy, M. Varia, H. Kennedy, J. Gardner, and S. Goldberg, "The Security of NTP's Datagram Protocol," in *Financial Cryptography and Data Security* (A. Kiayias, ed.), (Cham), pp. 405–423, Springer International Publishing, 2017. 57

[58] T. U. S. N. O. (USNO), "Authenticated NTP." [Online]. Available from: http://www.usno.navy.mil/USNO/time/ntp/dod-customers, 2010. 57

[59] B. Haberman and D. Mills, "Network Time Protocol Version 4: Autokey Specification," tech. rep., 2010. RFC 5906. 58

[60] S. Röttger, "Analysis of the NTP Autokey Procedures," Master's thesis, Technische Universität Braunschweig, February 2012. 58

[61] B. Dowling, D. Stebila, and G. Zaverucha, "Authenticated Network Time Synchronization," in *25th USENIX Security Symposium*, (Austin, TX), pp. 823–840, 2016. 58, 61

[62] G. Zaverucha, B. Dowling, and D. Stebila, "ANTP: Authenticated NTP Implementation Specification," tech. rep., February 2015. Microsoft Research. 58, 61

[63] B. Dowling, *Provable Security of Internet Protocols*. PhD thesis, Queensland University of Technology, 2017. 58, 61

[64] D. Franke, D. Sibold, and K. Teichel, "Network Time Security for the Network Time Protocol," tech. rep., 2017. IETF draft-ietf-ntp-using-nts-for-ntp-10. 59

[65] E. Rescorla, "Keying Material Exporters for Transport Layer Security (TLS)," RFC 5705, 2010. 59

[66] A. Aldini and R. Gorrieri, "A Study about Trade-Off between Performance and Security in an Internet Audio Mechanism," pp. 203–228, 02 2003. 60

[67] H. Ding, Q. Zhao, and R. Wu, "The Trade-off between Security and Performance of Encrypted Networked Control Systems," in *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, pp. 5616–5619, Oct 2017. 60

[68] W. Zeng and M. Chow, "Optimal Tradeoff Between Performance and Security in Networked Control Systems Based on Coevolutionary Algorithms," *IEEE Transactions on Industrial Electronics*, vol. 59, pp. 3016–3025, July 2012. 60

[69] M. Haleem, C. Mathur, R. Chandramouli, and K. Subbalakshmi, "Opportunistic Encryption: A Trade-Off between Security and Throughput in Wireless Networks," *Dependable and Secure Computing, IEEE Transactions on*, vol. 4, pp. 313–324, 11 2007. 60

[70] S. S. Yau, Y. Yin, and H. G. An, "An Adaptive Tradeoff Model for Service Performance and Security in Service-Based Systems," in *2009 IEEE International Conference on Web Services*, pp. 287–294, July 2009. 60

[71] M. T. Mizrahi, "RFC 7384: Security Requirements of Time Protocols in Packet Switched Networks," October 2014. 62

[72] A. Malhotra, A. Langley, and W. Ladd, "Roughtime," Internet-Draft draft-ietf-ntp-roughtime-00, Internet Engineering Task Force, Jan. 2020. Work in Progress. 62

[73] L. T. Renzo Efrain Navas, "LATe: A Lightweight Authenticated Time Synchronization Protocol for IoT," in *Global Internet of Things Summit (GIoTS)*, June 2018. 63

[74] Abadi, Martín, "Security Protocols: Principles and Calculi," in *Foundations of Security Analysis and Design IV* (Aldini, Alessandro and Gorrieri, Roberto, ed.), pp. 1–23, Springer Berlin Heidelberg, 2007. 73

[75] 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. 73, 74

[76] M. Abadi and C. Fournet, "Mobile Values, New Names, and Secure Communication," in *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '01, (New York, NY, USA), pp. 104–115, ACM, 2001. 74

[77] M. Abadi, B. Blanchet, and C. Fournet, "The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication," *J. ACM*, vol. 65, pp. 1:1–1:41, Oct. 2017. 74

[78] N. Saeedloei and G. Gupta, "Timed pi-Calculus," in *TGC*, 2013. 74

[79] V. C. Bruno Blanchet, Ben Smyth and M. Sylvestre, "Automatic Verification of Security Protocols in the Symbolic Model: The Verifier ProVerif." [Online]. Available from: https://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf. 74

[80] 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. 75

[81] 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*, pp. 14–15, 2002. 75

[82] P. Hopcroft and G. Lowe, "Analysing a Stream Authentication Protocol Using Model Checking," *International Journal of Information Security*, vol. 3, pp. 2–13, Oct 2004. 75

[83] "OpenSSL." [Online]. Available from: https://www.openssl.org. 87

[84] 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. 88

[85] T. Hansen and D. Eastlake, "US Secure Hash Algorithms (SHA and HMAC-SHA)," tech. rep., 2006. RFC 4634. 88

[86] J. Song, R. Poovendran, J. Lee, and T. Iwata, "The AES-CMAC Algorithm," tech. rep., 2006. RFC 4493. 88

[87] D. J. Bernstein *et al.*, "High-Speed High-Security Signatures," *Journal of Cryptographic Engineering*, pp. 77–89, 2012. 88

[88] D. J. Bernstein, "Ed25519 Implementation." [Online]. Available from: https://github.com/floodyberry/supercop/tree/master/crypto_sign/ed25519. 88

[89] D. Gligoroski *et al.*, "MQQ-SIG," *Trusted Systems*, pp. 184–203, 2011. 88

[90] R. E. Jensen and D. Gligoroski, "NTP : Security Vulnerabilities." [Online]. Available from: https://www.cvedetails.com/vulnerability-list/vendor_id-2153/NTP.html. 88

[91] A. Malhotra and S. Goldberg, "Message Authentication Codes for the Network Time Protocol," tech. rep., 2006. IETF draft-aanchal4-ntp-mac-02. 88

[92] TechDesign&Internet, "The Blockchain as Verified Public Timestamps." [Online]. Available from: https://www.nickgrossman.is/2015/the-blockchain-as-time/. 93

[93] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System." [Online]. Available from: https://bitcoin.org/bitcoin.pdf. 93, 94

[94] "Opentimestamp Project." [Online]. Available from: https://opentimestamps.org/. 94

[95] R. Zuccherato, P. Cain, D. C. Adams, and D. Pinkas, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)," Tech. Rep. 3161, aug 2001. 94

[96] Y. Gao and H. Nobuhara, "A Decentralized Trusted Timestamping Based on Blockchains," *IEEJ Journal of Industry Applications*, vol. 6, no. 4, pp. 252–257, 2017. 94

[97] V. Lemieux, "Trusting Records: Is Blockchain Technology the Answer?," 11 2015. 94

[98] Z. Li, "Will Blockchain Change the Audit?," *China-USA Business Review*, vol. 16, 06 2017. 94

[99] C. Jämthagen and M. Hell, "Blockchain-Based Publishing Layer for the Keyless Signing Infrastructure," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pp. 374–381, July 2016. 94

[100] J. Clark and A. Essex, "CommitCoin: Carbon Dating Commitments with Bitcoin," in *Financial Cryptography and Data Security* (A. D. Keromytis, ed.), (Berlin, Heidelberg), pp. 390–398, Springer Berlin Heidelberg, 2012. 94

[101] S. Kalidindi, M. J. Zekauskas, and D. G. T. Almes, “A Round-trip Delay Metric for IPPM.” RFC 2681, Sept. 1999. 99

[102] A. Pathak, H. Pucha, Y. Zhang, Y. C. Hu, and Z. M. Mao, “A Measurement Study of Internet Delay Asymmetry,” in *ACM PAM*, pp. 182–191, 2008. 100

[103] G. Almes, S. Kalidindi, M. J. Zekauskas, and A. Morton, “A One-Way Delay Metric for IP Performance Metrics (IPPM).” RFC 7679, Jan. 2016. 100