Preventing (Network) Time Travel with Chronos

Omer Deutsch, Neta Rozen Schiff, Danny Dolev, Michael Schapira
School of Computer Science and Engineering, The Hebrew University of Jerusalem
omermaya@gmail.com, neta.rozenschiff@mail.huji.ac.il,danny.dolev@mail.huji.ac.il, schapiram@huji.ac.il

***Abstract*—The Network Time Protocol (NTP) synchronizes time across computer systems over the Internet. Unfortunately, NTP is highly vulnerable to "time shifting attacks", in which the attacker's goal is to shift forward/backward the local time at an NTP client. NTP's security vulnerabilities have severe implications for time-sensitive applications and for security mechanisms, including TLS certificates, DNS and DNSSEC, RPKI, Kerberos, BitCoin, and beyond. While technically NTP supports cryptographic authentication, it is very rarely used in practice and, worse yet, *timeshifting attacks on NTP are possible even if all NTP communications are encrypted and authenticated*.**

We present *Chronos*, a new NTP client that achieves good synchronization even in the presence of powerful attackers who are in direct control of a large number of NTP servers. Importantly, Chronos is *backwards compatible* with legacy NTP and involves no changes whatsoever to NTP servers. Chronos leverages ideas from distributed computing literature on clock synchronization in the presence of adversarial (Byzantine) behavior. A Chronos client iteratively "crowdsources" time queries across multiple NTP servers and applies a provably secure algorithm for eliminating "suspicious" responses and averaging over the remaining responses. Chronos is carefully engineered to minimize communication overhead so as to avoid overloading NTP servers.

We evaluate Chronos' security and network efficiency guarantees via a combination of theoretical analyses and experiments with a prototype implementation. Our results indicate that to succeed in shifting time at a Chronos client by over 100ms from the UTC, even a powerful man-in-the-middle attacker requires over 20 years of effort in expectation.

I. Introduction

A. NTP is Insecure

The Network Time Protocol (NTP) is the default protocol for synchronizing computer systems across the Internet and is ubiquitously deployed. Many applications, including security protocols and mechanisms such as TLS certificates, DNS (and DNSSEC), BGP security mechanisms (namely, RPKI), Kerberos, HTTP Strict Transport Security (HSTS), financial applications, and more, crucially rely on NTP for correctness and safety [9], [17], [18], [23].

However, similarly to other core components of the Internet's fragile infrastructure (e.g., TCP/IP, BGP, DNS), NTP was designed many decades ago and without security in mind. NTP's design thus reflects the need to achieve correctness in the presence of inaccurate clocks ("falsetickers" [33]), assumed to be fairly rare, as opposed to designated attacks by powerful adversaries. Consequently, NTP is alarmingly vulnerable to attacks, ranging from time shifting attacks that stealthily shift clocks on victim clients to denial-of-service attacks [24].

In a nutshell, NTP is based on a client-server architecture: an NTP-client periodically selects servers to sync to from a pool of servers. Selecting servers from the pool involves these servers passing a sequence of "tests" intended to establish their reliability and accuracy. The NTP client syncs its internal clock to the clock reading from these servers via an algorithm that mitigates the effects of variability in network latency.

Unfortunatey, man-in-the-middle attackers capable of intercepting traffic between a client and server (for instance, through BGP hijacking [13], [39], DNS hijacking [10], [16], [19]) can wreak havoc on time synchronization [3], [26]. Worse yet, even an off-path attacker incapable of observing NTP traffic can launch devastating attacks on the protocol by exploiting weaknesses in NTP's implementation [26].

Recently introduced patches to NTP's implementation eliminate/mitigate some off-path attacks and implementation flaws, yet attackers capable of manipulating client-server communications are deemed simply "*too strong for NTP, because a man-in-the-middle attacker can always bias time synchronization by dropping or delaying packets.*" [27].

Importantly, while the cure to *some* of NTP's ailments may lie in encrypting NTP traffic between clients and servers, even ubiquitous encryption and authentication is insufficient for protecting NTP time synchronization from an attacker capable merely of delaying and replaying packets. (In addition, this mode of operation is very rare in practice [26] and faces significant challenges to global adoption.)

B. Introducing Chronos

We present Chronos, an NTP client engineered to protect from timeshifting attacks. Chronos is engineered to achieve three desiderata:

- **Provable security** guarantees even against very powerful man-in-the-middle attackers. Specifically, Chronos is designed to protect even against attackers capable of compromising a large number of (even authenticated!) NTP servers.
- **Backwards-compatibility** with today's NTP servers. Chronos is designed to be readily deployable and, in particular, involves software changes to the NTP

Network and Distributed Systems Security (NDSS) Symposium 2018
18-21 February 2018, San Diego, CA, USA
ISBN 1-891562-49-5
http://dx.doi.org/10.14722/ndss.2018.23231
www.ndss-symposium.org

client side only and no changes whatsoever to NTP's message format or to NTP servers.

- **Low computational and communication overhead**. Today's NTP involves fairly little client-server communication. Overloading NTP servers can result in slower response times and, as a result, degraded synchronization. Chronos is thus engineered to avoid excessive overhead on both the clients and the servers.

Chronos' design leverages ideas from the rich body of literature in distributed computing theory in the presence of *Byzantine* attackers. Synchronizing clocks in distributed systems in the presence of faulty and malicious parties has been central to distributed computing research from its very early days [6], [15], [20], [44]. Chronos adapts ideas from this body of research, which typically assumes a relatively small group of computational nodes, to the context of large-scale, NTP-compatible time-synchronization across the global Internet.

A Chronos-client periodically queries small *subsets* of *large* pool of NTP-servers to solicit timing information and then applies a theory-informed algorithm for removing outliers and averaging over the remaining responses. We prove that this fairly low-overhead crowdsourcing scheme guarantees that the internal clock of each Chronos client remain close (time-wise) to the universal time (UTC), and that the clocks of any two Chronos-clients remain close to each other, even if the attacker controls a fairly large fraction of the NTP servers. Thus, Chronos provides meaningful security guarantees for adopters even under very partial deployment.

We evaluate Chronos' correctness, security, and network efficiency via a combination of theoretical and empirical analyses with a prototype implementation.

C. Our Contributions

We make the following main contributions:

- **Identifying key elements in NTP's architecture that render it vulnerable to man-in-the-middle attacks.** We highlight two crucial aspects of today's NTP architecture (and today's implementations) that make NTP clients particularly subsceptible to man-in-the-middle attackers: (1) the (typical) reliance on small sets of servers, and (2) utilizing (a variant of) Marzullos' algorithm [28]–[30] for selecting the servers to sync to.

- **Designing Chronos.** We present Chronos, a new NTP-compatible client carefully engineered to strike a good balance between security and deployability. Chronos' design revisits and replaces elements in the NTP client architecture that give rise to NTP's vulnerabilities to man-in-the-middle attacks (as outlined above). We grapple with algorithmic and operational issues involved in accomplishing this objective.

- **Evaluating Chronos from both a theoretical perspective and an empirical perspective.** We study Chronos' security guarantees and overhead through both extensive theoretical analysis and empirical analyses of a prototype implementation of Chronos on Amazon AWS (EC2) machines in 6 different regions in the USA and Europe. Our results indicate that to succeed in shifting time at a Chronos client by even a small amount (e.g., 100ms), even a powerful man-in-the-middle attacker requires many years of effort (e.g., over 20 years in expectation).

- **Outlining directions for future research.** We view Chronos as a first step towards securing NTP from man-in-the-middle attacks. We leave the reader with interesting research directions regarding how Chronos can be extended and regarding applications of our ideas to other time-synchronization contexts.

D. Organization

We provide a high-level overview of NTP and discuss some of NTP's security vulnerabilities in Section II-B. We present Chronos' design in Section III-B and analyze its security guarantees in Section IV. We report on empirical analyses using a prototype implementation of Chronos, and on how these can be used to guide parameter value assignments in Chronos in Section V. Section VI presents related work. We conclude and discuss interesting directions for future research in Section VII.

II. NTP: Overview and Vulnerabilities

We present below a high-level overview of NTP's architecture. To simplify exposition, many technical details are omitted (e.g., the notion of NTP strata, delay computation, and more). The reader is referred to [1], [31], [32] for a thorough exposition. After presenting NTP's architecture, we discuss NTP's notorious security vulnerabilities.

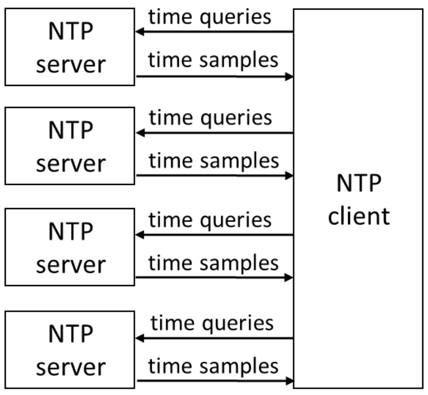
A. NTP Overview

NTP's basic client-server architecture consists of two main consecutive steps: (1) the *poll process*, in which the NTP client exchanges messages with NTP servers so as to gather time samples (and other parameters such as per server delay), and (2) selecting the "best" time samples, and updating the local clock accordingly. Next a high-level overview of these steps, as captured by today's implementation of NTP [31], is presented. See [1], [32] for more details.

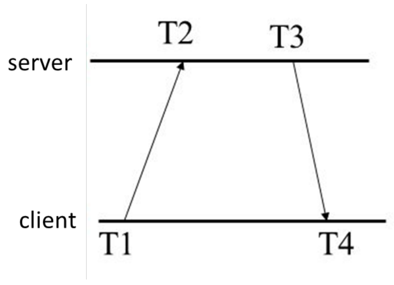
Poll process. Under NTP, the client periodically exchanges messages with a set of NTP servers Q so as to attain clock readings from each server in Q. See Figure 1(a) for an illustration. The set of servers Q, termed here the client-associated *server pool*, is typically determined via a DNS query to pool.ntp.org and consists of NTP servers that are "close" to the client. The client samples the time at each server in Q by sending an NTP time query to the server and receiving an NTP response. Through this interaction, the client obtains 4 distinct timestamps per query:

- T_1: the local time at the client upon sending the query.

- T_2: the local time at the server upon receiving the query.



(a) Client-server message exchange



(b) Measuring Offset in NTP

Figure 1: NTP's poll process

- T_3: the local time at the server upon sending the response.
- T_4: the local time at the client upon receiving the response.

These timestamps are then used to compute the *offset* $\theta = \frac{1}{2}((T_2 - T_1) + (T_3 - T_4))$ [24], [38], which is intended to capture the difference between the local time at the client and the local time at the server. The client gathers several time samples from each server in the server pool and computes, for each sample, the associated offsets. See Figure 1(b) for an illustration.

Selecting the "best" time samples. After gathering several time values (and measuring offsets) from each server on the list Q, the client applies a 5-step algorithm to compute a new time to update its local clock to (see [1], [31]). This process is described in Figure 2.

First, the time sample with the lowest offset collected from each server is identified and all other time samples are discarded. Then, various "sanity checks" are performed on the surviving samples so as to eliminate time samples from servers that seem unhealthy or unsuitable for synchronization [1], [31]. Then, Marzullo's algorithm [28]–[30] is applied to find, within the set of remaining time samples, a "majority clique of truechimers" [1], [31], i.e., a large cluster of servers with (fairly) accurate clocks. This set of time samples can be further pruned, with the aim of improving accuracy, by removing all but some predetermined number of time samples that are within the smallest distance of each other. The surviving time samples are then combined into a single average time value. In the event that the computed time value is "far" from the current local time (i.e., the current local time is "stale"), the local time is updated to reflect the new time value.

B. NTP's Security Vulnerabilities

NTP, similarly to other Internet protocols from the same era (e.g., BGP, TCP, DNS), was designed at a time when the Internet was comprised of only trusted parties and security was not a concern. NTP's design thus reflects the need to achieve correctness in the presence of faulty (slow/fast) clocks, not of designated attacks [21], [24].

Indeed, as asserted by Mills [33], a key design assumption was that truechimers, i.e., servers that maintain time accuracy, are numerous, whereas "falsetickers" are rare and, moreover, widely distributed across the measurement space. Consequently, NTP remains extremely vulnerable to different types of attacks.

Attackers in the NTP context can be categorized into two main classes: (1) off-path attackers, who cannot observe (let alone tamper with) the traffic between the NTP client and NTP servers, and (2) man-in-the-middle (MitM) attackers, who can eavesdrop on and manipulate traffic from client to server and vice versa. MitM attackers might be positioned on the path from NTP client to server (for instance, through BGP hijacking [13], [39] or DNS hijacking [10], [16], [19]), or even be in direct control of NTP servers.

Off-path attackers. Recent studies [24], [25] demonstrate the ability of off-path attackers to launch denial-of-service (DoS) attacks and also shift the local time at the client by exploiting weaknesses in NTP's implementation (e.g., via spoofed Kiss-o'-Death packets). Recently introduced patches to NTP's implementation eliminate/mitigate some of these vulnerabilities.

MitM attackers. What about MitM attackers? While some forms of MitM attacks on NTP can be thwarted [24], NTP is essentially defenseless against MitM attackers capable of dropping, delaying, and tampering with live NTP traffic [35], [37], [41]. Indeed, as articulated in [27]: *"... the MiTM model is too strong for NTP, because a MiTM can always bias time synchronization by dropping or delaying packets.*

While NTP supports cryptographic authentication [9], [43], even *perfect*, globally deployed authentication will not prevent a MitM attacker from shifting time through packet delays, and also leave NTP exposed to attackers who manage to gain control of (authenticated) NTP servers. (We note that in practice NTP traffic is very rarely authenticated anyway, as a result of a cumbersome key-distribution mechanism, weaknesses in the Autokey protocol for public-key authentication, and more [14], [37], [40].)

C. Why is NTP So Vulnerable to MitM Attacks?

To enhance NTP's security against MitM attackers we must first identify the elements in today's NTP architecture (and standard implementations) that underly its vulnerability to such attackers. We point out two such factors: (1) (typical)

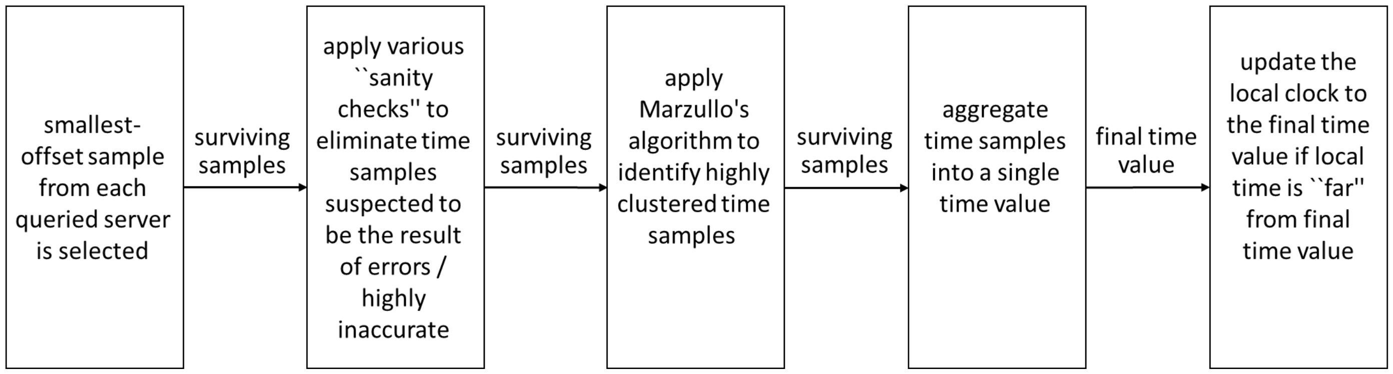

smallest-offset sample from each queried server is selected
surviving samples
apply various ``sanity checks'' to eliminate time samples suspected to be the result of errors / highly inaccurate
surviving samples
apply Marzullo's algorithm to identify highly clustered time samples
surviving samples
aggregate time samples into a single time value
final time value
update the local clock to the final time value if local time is ``far'' from final time value

Figure 2: NTP's selection process

reliance on a small server pool, and (2) NTP's algorithm for determining the set of servers to sync to (and, in particular, utilizing a variant of Marzullo's algorithm [28]). We next elaborate on each of these two elements.

NTP clients typically rely on small server pools. Today's NTP clients typically generate a server pool by querying DNS for the URL pool.ntp.org. To load balance NTP servers, instead of responding with a comprehensive list of the IP addresses of "close" NTP servers, the response contains few (typically at most 4) IPs, which are replaced once an hour.

Consequently, a MitM attacker capable of intercepting traffic between the NTP client and fairly few servers can succeed in shifting time at the client.

DNS responses are often cached, and so subsequent queries to pool.ntp.org will often result in the exact same list of server IP addresses. Hence, an attacker with MitM capabilities with respect to these servers is likely to be able to preserve these capabilities for extended periods of time.

We point out that reliance on small server pools is often not merely "forced" on NTP clients because of insufficient options, as described above, but is also an explicit guideline. Indeed, as prescribed in [12]: "For NTP Pool Project members to work properly, the NTP daemon needs at least three servers configured. The project recommends a minimum of four, and no more than seven sources."

NTP's algorithm for selecting the "best" time samples is not sufficiently resilient to MitM attacks. NTP clients employ a variant of Marzullo's algorithm to find a "majority clique of truechimers" within the set of queries servers, [1], [31]. This translates to seeking a set of at least half the queried servers whose time samples are within a small distance from each other.

The NTP specification in [1], [31] explains that this algorithm "uses Byzantine fault detection principles to discard the presumably incorrect candidates, called falsetickers, from the incident population, leaving only good candidates, called truechimers". Importantly, however, Marzullo's algorithm was not designed to withstand malicious (Byzantine) attacks, but to "withstand errors such as communication failures and inaccurate clocks" [28] (in contrast to, e.g., [7], [8], [44]). Indeed, as pointed out in [1], [31], "While not necessarily designed to defeat determined intruders, these algorithms and accompanying sanity checks have functioned well over the years to deflect improperly operating but presumably friendly scenarios."

Marzullo's algorithm, as adapted and employed in [1], [31], is intended to find a small time interval such that the time samples of a majority of the queried servers are within the interval. Hence, when the attacker has MitM capabilities with respect to over half of the queried servers, the client's is at the attacker's mercy. In fact, a sophisticated MitM attacker might be able to successfully launch timeshifting attacks even when in control of a *lower* fraction of the queried servers, e.g., (1) if the attacker is able to provide time samples that survive the NTP client's "sanity checks" while honest servers with (fairly) accurate clocks are removed from consideration, or (2) by taking advantage of situations in which the time samples of honest servers with (fairly) accurate clocks (while not far from each other) are not very closely clustered.

We point out that requiring that seeking a *majority* (i.e., at least half) of queried servers whose time samples are close to each other (as opposed to, e.g., a third, or two thirds) is somewhat arbitrary. This choice of threshold (50%) is intended to strike a balance between the desire to find a time value that a large fraction of the servers "agree on", and the risk that, if the threshold is set to be too high, such a time value might not even exist .

Another vulnerability of NTP's time-update algorithm is derived from the fact that the local clock is updated to the computed time value even if the new time value is "far" from the current local time (see Figure 2). This implies that an attacker who succeeds in contaminating the time samples can successfully shift time at the client by many seconds, even minutes (until the next time update occurs). See thorough discussion in [24].

Chronos to the rescue. Chronos, our NTP client, addresses the above two deficiencies of NTP's architecture by (1) generating large server pools (yet querying servers in these pools in a communication-efficient manner), and (2) replacing NTP's time-sample-selection algorithm with a scheme that is provably more resilient to attacks.

III. Chronos

The Chronos NTP client is carefully designed to achieve the following desiderata:

- **Provable security** guarantees even against very powerful man-in-the-middle attackers. Chronos is designed to protect even against attackers capable of compromising *authenticated* NTP servers.

- **Backwards-compatibility** with today's NTP servers. Chronos only involves software changes to the *NTP client* and is thus easy to deploy in practice.

- **Low computational and communication overhead**. Today's NTP involves fairly little client-server communication. Overloading NTP servers can result in slower response times and, as a result, degraded synchronization. Chronos is thus engineered to avoid excessive overhead on both the NTP servers and the Chronos clients.

To accomplish the above, Chronos' design relies on translating ideas from the rich literature on time synchronization in distributed computing into an operational reality.

We next provide a high-level overview of Chronos' architecture. We then elaborate on the main technical aspects involved in Chronos' design.

A. Overview

Our aim is to protect NTP from even the most powerful form of MitM attacks, i.e., attackers capable of compromising fairly many (possibly authenticated!) NTP servers. Our approach relies on a combination of several ingredients:

1) **relying on many servers** (in contrast to today's NTP clients, which *effectively* often rely on small server pools),
2) **querying few servers** so as to avoid excessive communication overhead in a manner that limits a MitM attacker's ability to attain a large presence within the set of queried servers, and
3) **crowdsourcing time queries across servers** by leveraging ideas from distributed computing literature [7], [8], [44]. We next discuss the algorithmic and operational aspects involved in Chronos' design, and on how Chronos contends with these challenges.

Generating a large pool of NTP servers to sync to. As discussed in Section II, an NTP client gathers time samples from a pool of servers, which are then used to generate a new local time. Today's NTP clients typically rely on a small pool of servers (e.g., $4-7$ [12]), rendering them vulnerable to MitM attacks. Chronos, in contrast, generates a large server pool (in the order of hundreds) per client so as to set a very high threshold for a MitM attacker, effectively forcing the attacker to gain control of the traffic between the clients and a large fraction of the servers so as to be successful.

While a large server pool can be generated in multiple ways (e.g., fed into Chronos by a trusted party), to avoid the need for coordination with others, Chronos clients can also locally utilize a simple approach leveraging DNS queries to create server pools that consist of hundreds of servers. We detail the specifics of this approach below and empirically evaluate its success in Section V-A .

Querying a small subset of the server-pool. While Chronos' security guarantees rely on generating a large server pool per client, as explained above, querying each and every server in the server pool is not advisable, as this will result in excessive overhead. Thus, Chronos is designed to strike a delicate balance between communication and server load and the attained security level. We prove that randomly querying a small fraction of the servers in the server pool, as few as 15, suffices for attaining very good security guarantees.

Sifting through the collected time samples. After gathering time samples from a subset of the server pool, Chronos discards "outliers" in the following manner: Suppose that m per server samples are gathered. Chronos discards the d lowest-value time samples and the d highest-value time samples for some predetermined constant d. Chronos next examines the remaining time samples and ensures that the following two conditions are satisfied:

1) every two surviving time samples are "close" to each other
2) the average time value of the surviving samples is "close" to the local time at the client.

In the event that the above two conditions are satisfied, Chronos updates the local time to be the average time value across all $m-2d$ time samples. Otherwise, the time samples are discarded and the server pool is re-sampled in the exact same manner. In the extreme scenario that the number of times the pool is re-sampled exceeds a certain "Panic Trigger", Chronos enters a "Panic Mode", which involves sampling *all* servers in the pool.

Intuitively, this simple scheme, which resonates classical ideas in distributed computing theory [7], [8], [44], poses a challenge for the attacker, as explained next. Suppose that the attacker manages to manipulate the time samples of a fairly large fraction, yet not all, of the queried servers in the pool. Reporting time values that are "too high" or "too low" will result in these time samples being discarded by Chronos and thus limit the attacker's ability to influence Chronos' time computation. However, reporting time values that are not outliers and that pass Chronos' checks implies that the reported values are close to those reported by NTP servers that the attacker cannot manipulate (and are hence close to the UTC). We formalize this intuition in Section III-B.

B. Chronos Design

We next present a more detailed exposition of Chronos' design. We first explain how Chronos generates a large server pool. We then explain how the server pool is sampled and how the new local time is computed.

The pseudocode for Chronos' sampling scheme and time computation, and the relevant terminology, are presented in the specification of Algorithm 1 and in Table I, respectively.

Choosing values for the parameters described in the below exposition of Chronos is delicate and relies on both our formal security analysis (Section IV) and empirical analyses (Section V). We discuss this issue in detail in the subsequent sections.

n	total number of servers in the server pool
m	number of servers chosen at random from the server pool
d	number of outliers removed from each end of the ordered m samples
T	the set of $m - 2d$ samples remaining after the removal of the $2d$ outliers
avg(T)	the average value of the time samples in T
ω	an upper bound on the distance from the UTC of the local time at any NTP server with an accurate clock ("truechimer")
Θ	an upper bound on the drift of the client's local clock [ms/sec]
Δ_t	the client's estimate for the time that passed since its last synchronization to the server pool [sec]
ERR	$\frac{\Theta \cdot \Delta_t}{1000}$
t_C	the current time, as indicated by the client's local clock [sec]
K	panic trigger

Table I: Relevant Notation

```
counter := 0;
while $counter < K$ do
    S := sample(m) // gather time samples from $m$ randomly chosen servers
    T := bi-sided-trim(S,d) // trim $d$ lowest and highest values;
    if $(max(T) - min(T) <= 2\omega)$ and $(|avg(T) - t_C| < ERR + 2\omega$ then
        return avg(T).
    end
    counter++;
end
// panic mode;
S := sample(n);
T := bi-sided-trim(S, $\frac{n}{3}$) // trim bottom and top thirds;
return avg(T).
```

Algorithm 1: Pseudocode of Chronos' Time Sampling Scheme

Generating the server pool. To generate a large server pool, our current backwards-compatible realization of Chronos executes the following procedure. The NTP client queries pool.ntp.org on an hourly basis for 24 consecutive hours and generates the union of all received IP addresses. Importantly, this is executed in the background once in a long time (e.g., every few weeks/months). Our empirical results in Section V-A demonstrate that server pools consisting of hundreds of servers can be generated in such a manner.

Sampling the server pool and removing outliers. Chronos periodically gathers time samples from the server pool as follows. Chronos first selects, uniformly at random, a subset of size m of the servers in the server pool. Out of the collected m samples, the d lowest-value samples and d highest value samples are discarded. Our security analysis in Section IV-B establishes that setting m to be in the range $15 - 50$ and d to be $\frac{m}{3}$ yields good security guarantees.

Checking samples for consistency. Let Θ, measured in msec/sec, be an upper bound on the *drift* of the local clock at the client, i.e., the worst rate at which the correctly functioning local clock desynchronizes from the universal time (UTC). The current implementation of the NTP client relies on a local estimation of Θ. Specifically, today's NTP clients keep track of the local clock drift (at /var/lib/ntp/ntp.drift, see, e.g., [12]) and can set Θ to be an upper bound on the locally-perceived drift. Identifying better ways of assessing the local drift is an interesting direction for future research. Let Δ_t denote the client's estimation of the time, measured in seconds, that passed since its last synchronization to the server pool. We define the *error margin* ERR to be $\frac{\Theta \cdot \Delta_t}{1000}$.

We refer to an NTP server whose local clock is at most at distance ω from the UTC, for some predetermined value ω, as a *truechimer*. Observe that the local times at any two truechimers can be at most 2ω away from each other. We discuss in Section V-B how the value of ω should be set in Chronos so as to strike a good balance between communication overhead and security guarantees. Our empirical observations (presented in Section V-B) suggest that setting ω to be in the range $25 - 30$ms provides both high time accuracy and good security.

Chronos checks whether the $m-2d$ time samples surviving the elimination of the $2d$ outliers satisfy two conditions:

- The maximal distance between every two time samples does not exceed 2ω.
- The average value of the $m - 2d$ time samples is at distance at most $ERR+2\omega$ from the time t_C indicated by the client's local clock.

In the event that both of these conditions are satisfied, the local clock is updated to be the average time value. Otherwise, m servers in the server pool are sampled again in the exact same manner. This re-sampling process continues until the two conditions are finally satisfied or the number of times the servers are re-sampled exceeds a "Panic Trigger" K, in which case, Chronos enters a "Panic Mode".

Panic mode. Upon entering the Panic Mode, a Chronos client queries *all* n servers in the server pool, orders the collected time samples from lowest to highest and eliminates the bottom third and the top third of the samples. The client then averages over the remaining samples and updates the local clock to be the computed value.

Remark I: Initializing Chronos. When a Chronos client synchronizes to NTP servers for the first time, this involves querying many servers, as in the above described Panic Mode.

Remark II: Accuracy. Chronos is designed to select time samples in a manner that yields provable security guarantees (see Section IV). Importantly, however, Chronos can easily be augmented with today's NTP clients "sanity checks", aimed to improve accuracy, minimize jitter, etc., without compromising its security guarantees.

In fact, the *only* component of the NTP client's multi-step synchronization algorithm that is modified in Chronos is the Clock Select Algorithm [1], [31]. In particular, Chronos' sample-selection scheme, as described above, can be applied after gathering *multiple* time-samples from each queried servers and removing all but the lowest-offset sample per server as in NTP's Clock Filter Algorithm [1], [31]). In addition, the time-samples that survive Chronos' removal of top and bottom samples can be further pruned as in NTP's Cluster Algorithm [1], [31].

IV. Security Guarantees

We discuss below our threat model and then present Chronos' security guarantees.

A. Threat Model

We consider a fairly powerful (in the NTP context) form of man-in-the-middle (MitM) Byzantine [4] attacker. MitM attackers in our model are capable of determining precisely the values of the time samples gathered by the Chronos client from a subset of the NTP servers in its server pool. Our threat model thus encompasses a broad spectrum of MitM attackers ranging from fairly weak (yet dangerous) MitM attackers only capable of delaying and dropping packets to extremely powerful MitM attackers who control *authenticated* NTP servers and can perfectly time the arrival times at the client of NTP responses from these servers.

MitM attackers captured by our framework might be, e.g., (1) in direct control of a fraction of the NTP servers (e.g., by exploiting a software vulnerability in a specific NTP implementation, or through physical presence), (2) an ISP (or other Autonomous-System-level attacker) on the default BGP paths from the NTP client to a fraction of the available servers, (3) a nation state with authority over the owners of NTP servers in its jurisdiction, or (4) an attacker capable of hijacking (e.g., through DNS cache poisoning or BGP prefix hijacking) traffic to some of the available NTP servers. Importantly, we abstract away the details of the specific attack scenario by reasoning about MitM attackers in terms of the fraction of servers with respect to whom the attacker has MitM capabilities.

Specifically, we quantify the attacker's MitM capabilities in terms of a value $p \in [0, 1]$ p indicates the probability, for each and every server in the client's server pool, that the attacker controls that NTP server. $p = 0.1$, for instance, means that the attacker controls each server with probability 0.1 and thus controls 10% of the server pool in expectation.

We refer to servers not controlled by the attacker as *truechimers* and assume that the local clock of each truechimer is at most at distance ω from the UTC, for some predetermined value ω (see discussion in Section V-B about good choices of ω).

Remark I: Servers with inaccurate clocks are modeled as controlled by the attacker. Observe that our model does not explicitly reason about "honest" servers with inaccurate clocks ("falsetickers" in [33]). To provide strong security guarantees, our security analysis relies on the assumption that *all* non-truechimers are controlled by an attacker who can adversarially and arbitrarily manipulate time samples.

Remark II: Some MitM attackers are indeed too powerful for NTP. A MitM attacker in control of a majority of the NTP servers in the server pool can manipulate time as it pleases under *any* NTP security scheme. To see this, consider the scenario that all attacker-controlled NTP servers report the same (false) time when queried and the timesamples of all truechimers are precisely the UTC. Observe that this scenario is indistinguishable, from the NTP client's perspective, from the scenario that the attacker-controlled servers are actually the truechimers and vice versa.

Hence, if the attacker is, e.g., the client's sole network gateway (say, a compromised home router), then no level of security is achievable. Thus, the security of any NTP client is subject to the assumption that the attacker does not have MitM capabilities with respect to "too many of the servers". Our results below show that Chronos is capable of providing meaningful security guarantees even with respect to MitM attackers in control of a large fraction (e.g., one third) of the servers.

B. Security Analysis

We present below the security analysis of Chronos' time computation scheme. We first present an illustration of the established security guarantees for reasonable choices of parameters (see discussion of parameter values in the remainder of this section and in Section IV-E): $n = 500$, $m = 15$, $d = 5$, $\omega = 25ms$, $K = 4$, and $p = \frac{1}{7}$.

Theorem 4.1: When $n = 500$, $m = 15$, $d = 5$, $\omega = 25ms$, $K = 4$, $p = \frac{1}{7}$, Err = 50ms, and the Chronos client synchronizes once an hour:

1) To succeed in shifting time at a Chronos client by at least 100ms from the UTC, the attacker requires at least 22 years in expectation.
2) To succeed in creating a time difference of at least 100ms between two Chronos clients, the attacker requires at least 11 years in expectation.

The proof of Theorem 4.1 follows from plugging in the above parameters into the security analysis presented below.

Since local clocks tend to drift, NTP clients must synchronize with NTP servers periodically so that the time distance from the UTC does not exceed a certain bound. The length of the time interval between synchronization periods is set to be such that, despite the drift, the local clock is still within some desired range from the UTC. See [6] for a thorough analysis of how to set this value. To simplify exposition, our analysis below assumes that synchronization occurs frequently (say, on an hourly basis, as with today's NTP clients), and so ignores drift-related factors.

To gain intuition into Chronos' security guarantees, consider first a single application of Chronos' time sampling scheme (we later discuss the effect of multiple re-samplings and of the Panic Mode). As explained in Section III-B, the Chronos client queries m servers, chosen uniformly at random from a server-pool of size n. Suppose that an attacker controls some of the servers in the server-pool. We next consider

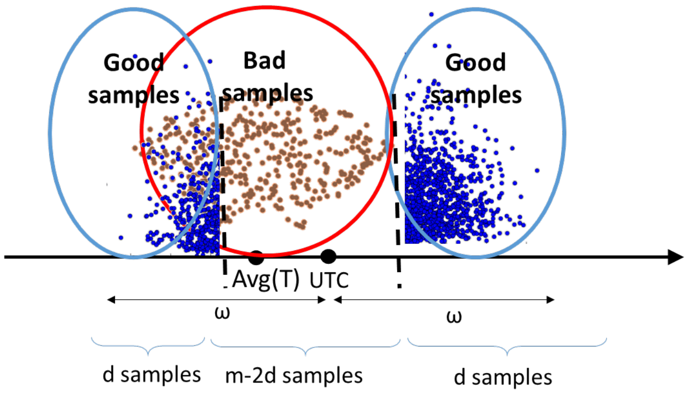


Figure 3: Case I, scenario I: all truechimers' time samples are discarded

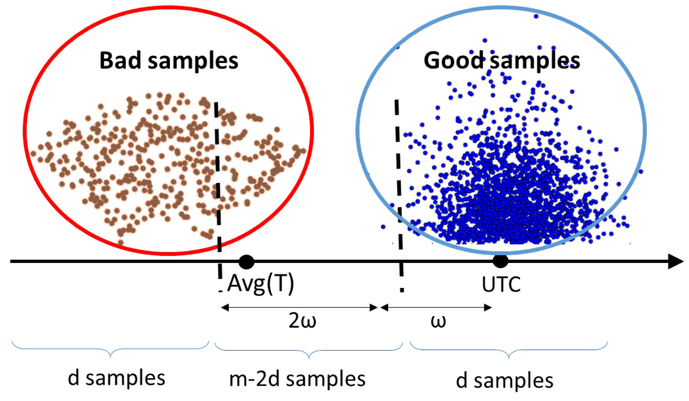


Figure 4: Case I, scenario II: at least one of the $m-2d$ surviving time samples is of a "truechimer".

the spectrum of feasible attack scenarios so to evaluate the effectiveness of Chronos in thwarting timeshifting attacks.

- **Case I: Less than $m-d$ of the queried servers are under the attacker's control.** Observe that, in this scenario, since at least $d+1$ of the time samples are reported by truechimers, the attacker is faced with a choice between two strategies: (1) reporting time values so that all truechimers' time samples are discarded by Chronos (this is only feasible if the attacker controls sufficiently many servers), and (2) reporting time values such that at least a single truechimer's sample survives Chronos' sample elimination.
  Now, consider the first of these two scenarios. As the attacker controls less than $m-d$ of the m sampled servers, at least $d+1$ of the servers are truechimers. Since Chronos discards the d bottom samples and the d top samples, the elimination of all $d+1$ truechimers samples implies that at least a one truechimer's sample is at the bottom d values and at least a one honest sample is at the top d values. Figure 3 illustrates this scenario. Consequently, all surviving $m-2d$ samples must be between two truechimers samples. However, this implies that all these samples are at most ω-away from UTC and consequently, the average value is also within ω distance from the UTC. Hence, this attack strategy is ineffective.
  Consider next the scenario that not all truechimers' samples are discarded, i.e., at least a single truechimer's sample survives the elimination. Since Chronos checks that every two surviving samples are at distance no more than 2ω from each other, for the attacker to pass this test *all* bogus samples must be within this distance from this truechimer's sample and, consequently, so is the average value. Hence, again, this attack strategy is ineffective. Figure 4 illustrates this scenario.

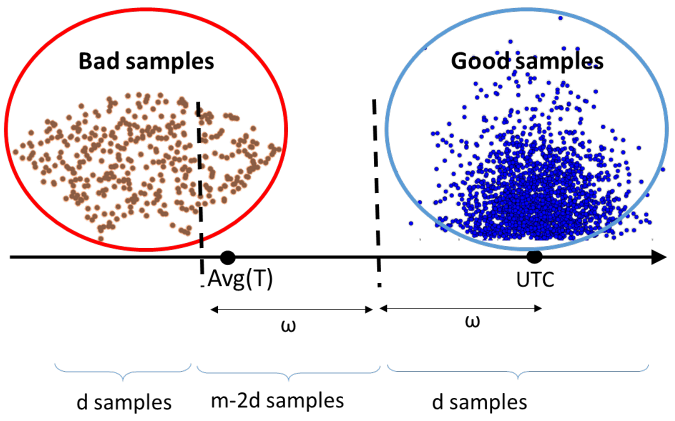


Figure 5: Case II: all surviving $m-2d$ samples are reported by the attacker

- **Case II: At least $m-d$ of the queried servers are under the attacker's control.** Observe that, in this scenario, the attacker can dictate the computed average time value simply by ensuring that all (at most d) truechimers' samples are within the d lowest time samples or that all truechimers' samples are within the d highest samples. Figure 5 illustrates this scenario. We show below, however, that this happens with tiny probability. We stress, moreover, that even if the attacker succeeds in controlling at least $m-d$ of the queried servers, Chronos' validation that the average time value is not "too far" from the local time implies that the attacker cannot shift the local time by "too much", namely, more than $Err+2\omega$. This implies that the attacker must succeed in attaining such large presence in the sampled set multiple times to cause significant time shifts. We discuss this further below.

Let Y be the random variable that captures the number of servers controlled by the attacker in the set of m randomly chosen servers queried by the Chronos client. The probability that the attacker controls r servers in the m samples is:

$$P_{Y=r} = P_r = p^r \cdot (1-p)^{m-r} \cdot \binom{m}{r} \tag{1}$$

Hence, the probability that the attacker controls more than

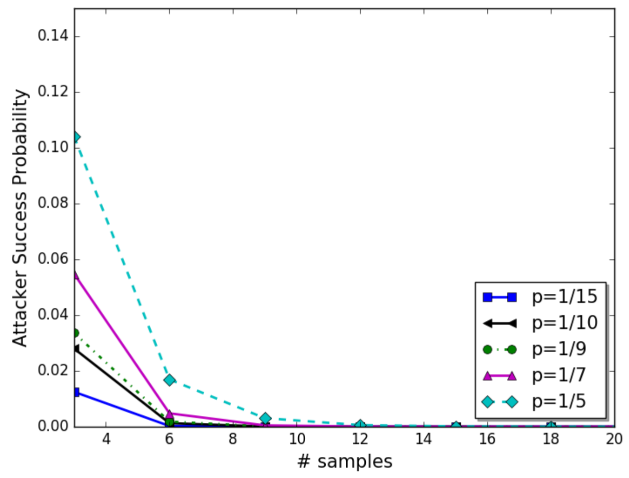


Figure 6: Probability that the attacker controls at least $m - d$ samples in one sampling iteration

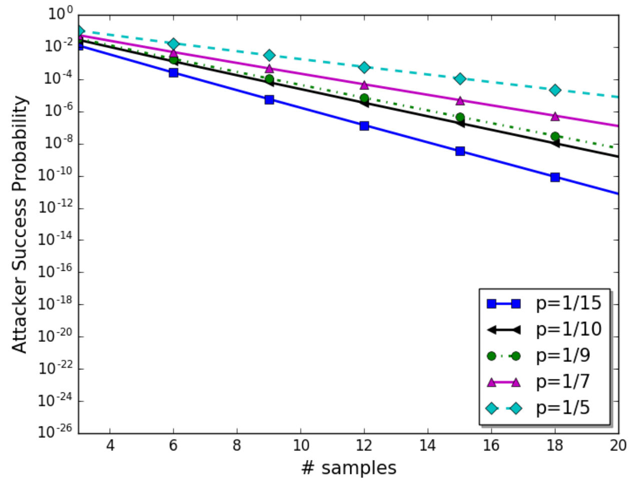


Figure 7: Probability that the attacker controls at least $m - d$ samples in one sampling iteration (log scale)

k_1 servers and at most k_2 servers is:

$$P(k_1 < Y \le k_2) = P_{(k_1,k_2]} = \sum_{k=k_1+1}^{k_2} p^k \cdot (1-p)^{m-k} \cdot \binom{m}{k} \tag{2}$$

The probability that the number of attacker-controlled servers is at least $m - d$ (Case II above) is thus:

$$P_{(m-d,m]} = \sum_{k=m-d+1}^{m} p^k \cdot (1-p)^{m-k} \cdot \binom{m}{k} \tag{3}$$

Figure 6 plots the numerical results of Equation 3 when $d = \frac{m}{3}$ for different values of p (Figure 7 shows the values in a log-scale). As is clear from the figure, even for fairly low

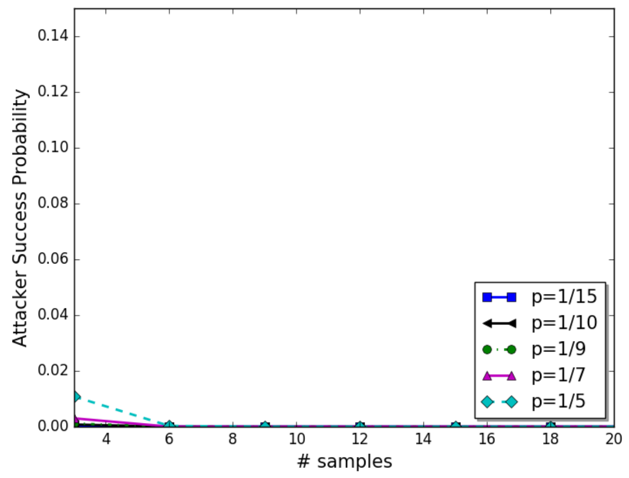


Figure 8: Probability that the attacker controls at least $m - d$ samples in two sampling iterations

values of m the probability is tiny. We set $d = \frac{m}{3}$ since this value provides the highest level of security in this context [6], [15], [44].

Recall, however, that (when not in Panic Mode) Chronos only updates the local clock at the client if the newly computed time value is “close” to the local time, i.e., at distance at most $Err + 2\omega$. Hence, even if the attacker-controlled servers in the sample constitute at least two thirds, the attacker cannot change the local time by “too much” (otherwise the server pool is re-sampled). Hence, to succeed in significantly shifting the time at the client, the attacker must succeed in controlling at least two thirds of the sampled servers in several server-polls. The probability of succeeding in doing so two times is upper bounded by $P^2_{(m-d,m]}$, which, as illustrated in Figure 8, is negligible.

Upon reaching Panic Mode, Chronos queries all servers in the server pool. The analysis of this scenario is equivalent to substituting m with n in Equation 3 (as now the sampled set consists of all servers in the server pool). Thus, so long as the attacker's MitM capabilities are below $p = \frac{1}{3}$, for large values n, e.g., in the order of hundreds, the attacker's ability desynchronize the client is practically nonexistent.

C. Selecting d

Intuitively, the choice of value for d, i.e., of the number of top samples and bottom samples to discard, involves a tradeoff: the lower the value of d, the easier it is for the attacker to “contaminate” at least one of the time samples that survive Chronos' elimination of the d highest and d lowest time-values (by controlling at least $d+1$ of the m sampled servers); the higher the value of d, the easier it is for the attacker to contaminate *all* these samples (by controlling at least $m - d$ of the sampled servers). What should the value of d be then?

To formalize the above, suppose that the attacker controls αn of the servers in the server pool. Then, asymptotically, so long as $d > \alpha m$, the attacker's probability of being in control

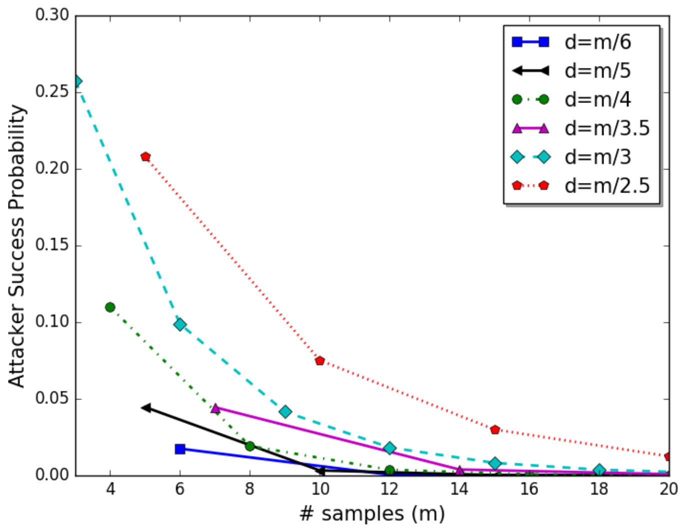


Figure 9: The attacker's probability, for different values of d, of injecting $m-d$ bad samples into the sample set when $\frac{n}{3}$ of the servers in the server pool are under his control.

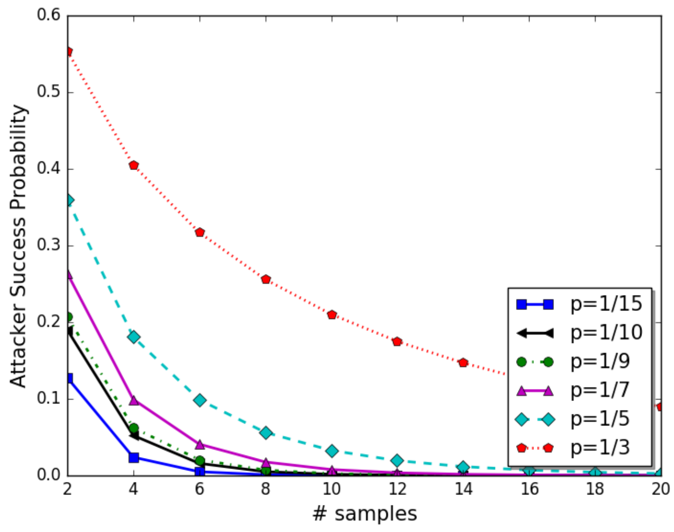


Figure 10: Probability that the attacker controls at least half of the queried servers (and can thus successfully attack today's NTP clients) in one sampling iteration

of at least $d+1$ of the sampled servers goes to 0 as m is increased; the higher the value of d the lower this probability is. In addition, so long as $d < m(1-\alpha)$, the attacker's probability of being in control of at least $m-d$ of the sampled servers goes to 0 as m is increased; the lower the value of d the lower this probability is. To illustrate the latter point, consider Figure 9, which plots the attacker's probability of injecting $m-d$ bad samples when in control of $\frac{n}{3}$ of the servers in the server pool. Observe that, as expected, the lower the value of d the lower the attacker's success probability is.

We find that choosing $d = \frac{m}{3}$ is a reasonable choice (both theoretically and empirically), as whenever the attacker is in control of strictly less than $\frac{n}{3}$ of the hundreds of servers in the server pool (a fairly conservative assumption), its probability of contaminating the samples is low.

D. Chronos Client vs. Today's NTP Clients

As discussed in Section II-C, today's NTP clients are vulnerable to MitM attacks for two main reasons: (1) reliance on a small server pool and (2) utilizing an algorithm for selecting time sample that is not sufficiently resilient to Byzantine MitM attackers. Chronos addresses the first issue by generating a large server pool (yet carefully querying fairly small sets of servers so as to be efficient). To address the second issue, Chronos replaces NTP's algorithm for time-sample selection, which extends Marzullo's algorithm, with an algorithm that leverages ideas from the literature in distributed computing on time synchronization in the presence of Byzantine attackers (as described above).

We now focus on analyzing in isolation the security benefits of Chronos' sample-selection algorithm by comparing Chronos to standard NTP clients when the size of the queried server set is identical for both (i.e., the standard NTP client to generate a large server set, as in Chronos, and its parameters are tuned so as to).

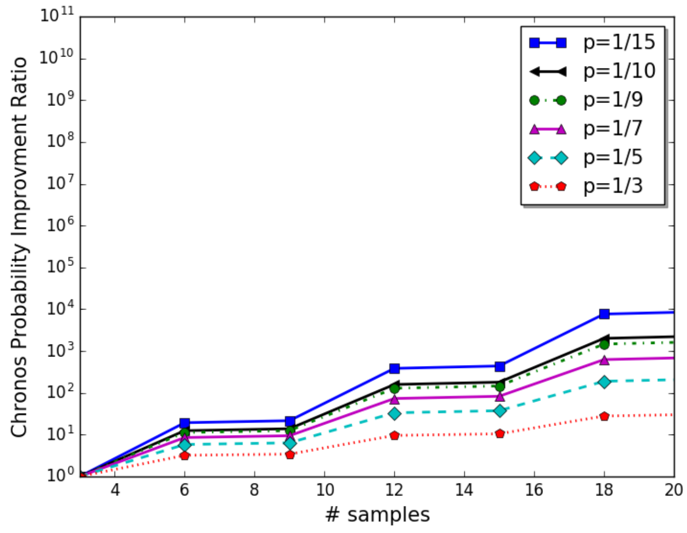


Figure 11: The ratio (in log scale) between the probability that the attacker controls at least half of the queried servers (and can thus successfully attack today's NTP clients) and the probability that the attacker controls two thirds of the queried servers (and can thus successfully attack Chronos for $d = \frac{m}{3}$) in one sampling iteration

As explained in Section IV-A, an attacker with MitM capabilities with respect to a very large fraction (e.g., over half) of the NTP servers can successfully shift time under any NTP security scheme. Our focus is thus on attack scenarios in which the fraction of the server pool controlled by the attacker is smaller. We show below that Chronos significantly outperforms today's NTP clients even when the attacker is in control of a fairly large (e.g., one third) of the NTP servers.

Recall (from Section II-C) that an attacker in control of at least a half of the servers queried by today's NTP clients can ensure that its time samples are used to synchronize the local clock at the client. In fact, as explained in Section II-C, in practice a sophisticated attacker can exploit NTP's "sanity checks" for servers and the fact that not all truechimers are perfectly aligned, to successfully launch attacks even with fewer queried servers under his control. In addition, recall (see discussion in Section II-C) that, unlike Chronos, today's NTP clients allow the attacker to report time values that are far from the local clock without triggering alarms (whereas in Chronos this results in re-sampling the server pool). In our comparison of Chronos to today's NTP client, discussed below, we make the assumption that the attacker can launch a successful attack on today's NTP clients *only if* 50% or more of the queried servers are his. We also ignore the *extent* to which the local clock value is shifted if the attacker is successful. Thus, our results for today's NTP clients greatly *overestimate* the ability of these clients to thwart MitM attacks. Still, as our analysis below indicates, Chronos is significantly more secure than today's approach.

Suppose that the attacker controls a third of the servers in the server pool, i.e., $\frac{n}{3}$ and 8 servers are queried out of a pool of 500 servers. The probability that at least half of the queried servers are under the attacker's control is fairly high (0.25), as can be seen in Figure 10. Importantly, even fairly low probabilities of success in attacking NTP translate to increasingly higher success probabilities as the number of sampling iterations grows, enabling a patient attacker to succeed eventually. (This is addressed in Chronos through the introduction of the Panic Mode, in which the attacker's success probability is effectively 0).

As discussed in Section IV-B, successful attacks on Chronos involve the attacker gaining control of $d-m$ of the queried servers. Figure 11 presents the log ratio between the probability that the attacker controls at least half of the servers (and can so successfully attack today's NTP client) and the probability that the attackers controls $d-m$ samples (and can so successfully attack Chronos), for $d=\frac{m}{3}$).

Let us revisit the above example of querying 8 servers out of a pool of 500 when the attacker controls $\frac{n}{3}$ of the servers in the pool. The probability of successfully attacking Chronos is significantly lower ($13\times$) than the probability of successfully attacking today's NTP clients. Observe that Chronos' security gains in comparison to today's NTP only increase as the number of queried servers is increased. Again, the results in Figure 11 should be regarded as a *strict lower bounds* on the improvement of Chronos, in terms of security, over today's NTP clients, as in practice a MitM attacker need not necessarily control such a large fraction of the queried servers to succeed in shifting time at the client.

The reader might wonder about whether the following alternative approach to "fixing" NTP's insecurity is useful: NTP's time-sample-selection algorithm can be altered so that, instead of seeking a time value that is (approximately) agreed upon by the *majority* of queried servers, a larger fraction of servers (say, two thirds) be required, thus setting a higher bar for the attacker. This is, however, not a feasible strategy for enhancing NTP's security. Indeed, recall (from Section II-C) that increasing the threshold of queried servers that must be in agreement might result in the NTP client not finding *any* agreed upon value, even when not attacked, as such point might not even exist. This will, in turn, result in not updating the local time at the client at all. Chronos's scheme for pruning time values, in contrast, sets a high bar for the attacker without running this risk.

E. Attacking Chronos by Creating Overhead

Our above analysis focused on the attacker's ability to shift time at the client. We now investigate a different attack strategy: generating communication overhead. Now, instead of shifting time, the attacker's desire is to overload NTP servers. The motivation for launching such attacks might be using Chronos clients to amplify denial-of-service attacks while preventing the attacked servers from tracing the attack back to the nefarious origin. We show below, however, that utilizing Chronos clients in this manner is largely ineffective.

As explained above, the considered attack is intended to create overhead and so the attacker's goal is to lead the Chronos client to re-sample the server pool multiple times and, eventually, enter Panic Mode and query all servers in the pool. The maximum number of re-samplings allowed before entering Panic Mode is specified by the Panic Trigger K.

We point out that so long as Panic Mode is not reached, Chronos' overhead is comparable to that of today's NTP clients. By default, NTP client-to-server synchronization takes place each hour. Today's NTP clients sync to few servers (e.g., $4-7$ [12]) and collect multiple (e.g., 8 [24]) samples from each server. Chronos, in contrast, queries more servers but does not need to query each server multiple times. Hence, so long as Panic Mode is not reached, for reasonable values of m and K, e.g., $m=15$ and $K=4$, the total number of queries cannot exceed $m(K-1)=45$.

So, to succeed in generating considerable overhead, the attacker need to be able to force the client into Panic Mode. Observe, however, that this can only occur in the event that at least $d+1$ attacker-controlled NTP servers are chosen in each and every one of the $K-1$ sampling iterations that occur prior to the panic mode. This only occurs with probability $(P_{(d+1,m]})^{K-1}=(1-P_{[0,d]})^{K-1}$. When $m=15$ and $K=4$, this probability equals $2\cdot10^{-6}$. Thus, if synchronization takes place once an hour, $5\cdot10^{5}$ hours are required, in expectation, for the attacker to succeed in reaching Panic Mode.

V. Empirical Analyses

Chronos' security guarantees rely on generating a large server pool per client. We explained in section III-B how such a server pool can be generated in practice in a backwards compatible manner and without coordination with others. We investigate the application of this approach in practice below.

Recall also that the security of Chronos is quantified in terms of ω, an upper bound on the distance between any truechimer and the UTC. Choosing a value for ω is challenging; setting the value to be "too low" might result in Chronos treating many honest NTP servers (whose local clocks are more than ω away from the UTC) as attackers; setting the value to be "too high", in contrast, yields poor security guarantees. Ideally, ω should be a low value such that the vast majority

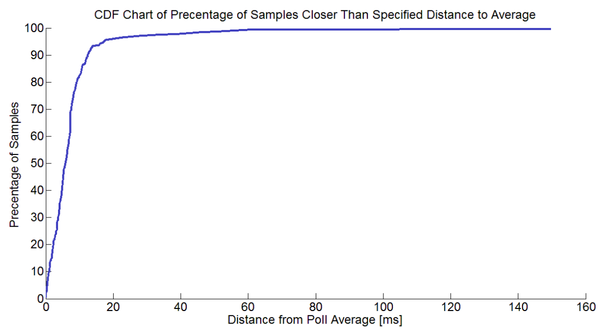


Figure 12: CDF chart presenting percentage of samples within specified distance of the poll's average.

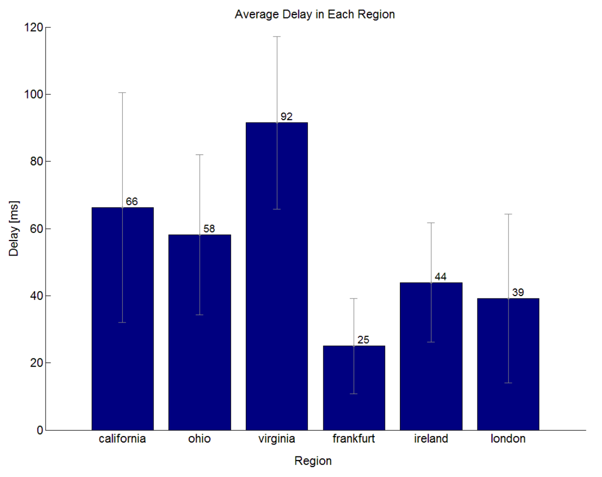


Figure 13: Average delay of NTP servers, as measured in each region

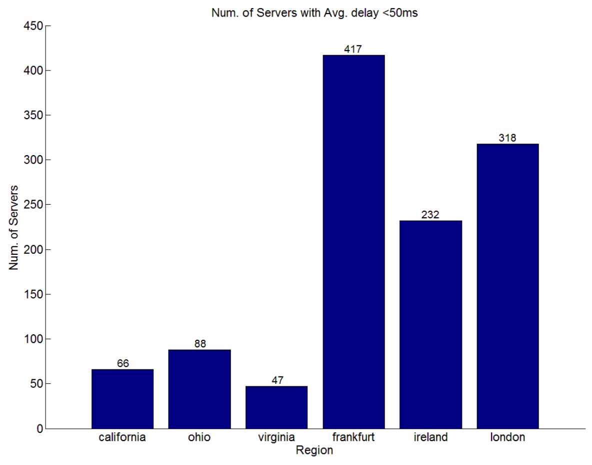


Figure 14: Number of servers with average delay less than 50ms in each region

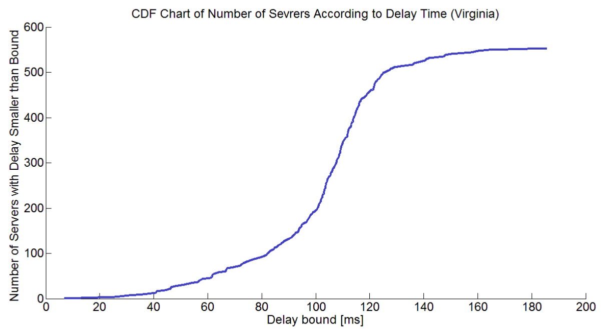


Figure 15: CDF chart presenting number of servers within specified delay bound in the Virginia region.

of honest NTP servers are within ω distance of the UTC. We embark on an empirical investigation of how the value of ω should be chosen.

To address the above challenges, we implemented a prototype version of Chronos in Python using the SNTP library (a "Simple NTP" client capable of querying a specific NTP server by IP and extract the relevant data from its reply). We ran our Chronos implementation on Amazon AWS (EC2) machines in 6 different regions: 3 in the USA (California, Ohio and Virginia) and 3 in Europe (Frankfurt, London and Ireland). We report on our findings below.

A. *Generating the Server Pool*

Generating large server pools. We ran the 24-hour server-pool generation process detailed in Section III-B from multiple EC2 machines in the US (West Coast) and Europe. Our experiments resulted in $300 - 600$ NTP server IPs per region, thus demonstrating that a large server pool can indeed be generated in such a manner. (See details about the NTP servers in each region at pool.ntp.org).

Precision vs. delay. Some NTP protocol versions eliminate servers from consideration if the delay (RTT) values for these servers exceed a certain threshold [1]. The guiding logic is that the values measured for closer (delay-wise) servers are more accurate, resulting in more accurate time updates. We show below that large server pools can be generated even when discarding servers with high delays.

We first point out, however, that for Chronos clients, in general, delay thresholds are much less meaningful than for standard NTP clients. The reason is that Chronos crowdsources time queries across many NTP servers, which gives rise to accurate time measurements, as explained next. Figure 12 plots a CDF of the distance of a time sample from the average of all time samples collected in Frankfurt (results for other regions exhibit similar trends). As can be seen in the figure, 96% of values are within only 20ms difference of the average value. Hence, when sampling a large enough fraction of the servers (as in Chronos), the average value is expected to be very close (with very high probability) to the global average.

Figure 13 plots the average delay and standard deviation for queried servers in different regions across multiple (2000) queries. As can be seen in the figure, the delay in Europe is, in general, much lower than in the US. Indeed, even after discarding all servers in the server pool with delay $>$ 50ms, the

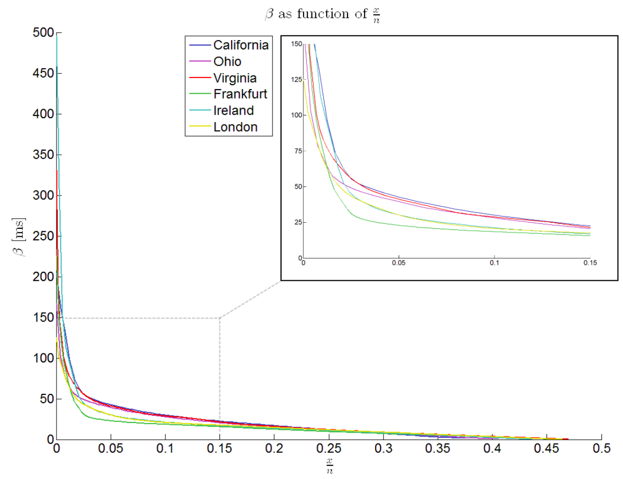


Figure 16: β as a function of $\frac{x}{n}$

server pool for the Chronos clients in Europe still consists of hundreds of servers, as described in Figure 14. Also, examining the delay values and distribution shown in Figure 13 reveals that setting a delay threshold of, say, $80ms$, is sufficient to obtain large server pools in California and Ohio, containing the majority of available NTP servers in these regions. We turn our attention to Virginia, which exhibits the highest average delay. Figure 15 depicts a CDF. As shown in the figure, setting the delay threshold to be $80ms$ yields a server pool of only 90 servers, whereas a threshold of $107ms$ yields a pool of 300 servers. We conclude that the desired delay threshold varies across geographical regions and that, in general, a delay threshold lower than $100ms$ enables the generation of very large server pools.

B. *Choosing ω*

Bounding inter-sample distances. Recall that Chronos, after sampling the server pool, discards the highest and lowest samples and then checks that the distance between the highest and lowest surviving samples does not exceed a certain threshold (specifically, 2ω). Our aim is to gain empirical insights into what would be a reasonable threshold to set.

We examine each of the 2000 all-server-sample sets and, for each such collection of samples, remove the x lowest-offset-value and x highest-offset-value samples for varying values of x. We refer to the distance between the lowest and highest surviving samples as β.

Figure 16 plots the average value of β, across all experiments, as a function of $\frac{x}{n}$, i.e., of the fraction of samples removed (from each end) from the total number of queried servers. Observe that when $\frac{x}{n} = 0.33$, i.e., two thirds of the samples are removed, $\beta <$ 4ms. Observe also that when $\frac{x}{n} = 0.033$, i.e. after removing 6.6% of the samples, β is always below 50ms (with a maximum difference of 25ms between regions). As these results indicate, even setting x to be very low with respect to the size of the sample n, i.e., when only the very lowest and very highest values are discarded, time values from servers are expected to be fairly close to each other.

Selecting ω. The above has direct implications for setting the value of ω. Suppose that the goal is to protect from an attacker capable of gaining control of, e.g., 10% of the hundreds of servers in the server pool (and moreover, controlling the servers with the most accurate clocks). Setting ω to be, e.g., $25ms$, implies that, over 83% of the servers (on average) are honest servers whose clocks are all within 2ω = 50ms from each other. Hence, setting ω to be in the order of 10s of milliseconds yields both high accuracy and strong security guarantees.

VI. Related Work

NTP's insecurity. The network time protocol (NTP) [14], [32], designed by Mills [34], is one of the Internet's oldest protocols. Already in 1985, in the context of the development of the Kerberos security model, NTP's inadequacy for achieving secure time synchronization was pointed out [34].

Recent years have witnessed renewed interest in NTP security. See taxonomy of attacks against NTP in [37]. Recent studies [24], [25] demonstrated the ability of even off-path attackers to launch denial-of-service (DoS) attacks and timeshifting attacks against NTP by exploiting weaknesses in NTP's implementation (e.g., via spoofed Kiss-o'-Death packets). Recently introduced patches to NTP's implementation eliminate/mitigate some of these vulnerabilities [1].

Today's NTP is effectively defenseless against Man-in-the-Middle (MitM) attackers capable of dropping, delaying, and tampering with client-server traffic (see section II).

Approaches to securing NTP. Although NTP supports cryptographic authentication [9], [43], in practice NTP traffic is very rarely authenticated [24], as a result of a cumbersome key-distribution mechanism, weaknesses in the Autokey protocol for public-key authentication, and beyond [14], [37], [40]. "Autokey (RFC 5906) public-key encryption scheme vulnerability issues too severe to be patched" motivated the development of NTPsec [2], implemented in 2015 [22].

Still, today's usage of NTPsec is very low and, more importantly, encryption and authentication do not prevent MitM attacks, as such attacks can rely simple on delaying and dropping (encrypted) traffic and, moreover, even perfect, globally deployed authentication will leave NTP exposed to attackers that gain control of (authenticated) NTP servers.

Another approach to NTP security is utilizing path redundancy on the Internet to avoid MitM attackers [36], [42]. Under this approach, multiple network paths are used to connect client and server. Chronos, in contrast, generates server redundancy through the creation of large server pools and carefully samples servers in these pools. Chronos is also designed to contend with a stronger notion of MitM attacker than that considered in [36], [42]; the threat model for Chronos considers an attacker in control of NTP servers, as opposed to a MitM attacker on a specific single path from the client to a server. Lastly, generating multiple disjoint Internet paths for client-server communication, as proposed in [36], [42], is a nontrivial task in practice.

Clock synchronization in distributed computing theory. Synchronizing clocks in a distributed system has been central to distributed computing research from its early days [20]. The early work of Cristian [5] influenced the design of the clock synchronization protocols that are still in use. The specific clock synchronization protocol used in NTP was suggested by Marzullo [28]–[30].

The topic of fault tolerant clock synchronization has also been the subject of much attention [6], [15], [44]. The idea of using approximate agreement [7], [8] to synchronize clocks was introduced in [44]. We point out, however, that the above mentioned fault tolerant protocols assumed a relatively small group of processors and were not designed to operate at Internet scale. Chronos relies on the adaptation ideas from this rich body of literature to the context of NTP time synchronization. This involves contending with challenges such as maintaining backwards compatibility, avoiding excessive overhead, and beyond.

VII. Conclusion and Future Research

We presented Chronos, an NTP client designed to enable secure time synchronization in a backwards compatible, easily deployable manner, and without involving excessive communication overhead. We view Chronos as a promising approach for securing NTP in a way that sidesteps the many obstacles facing prior approaches.

We leave the reader with interesting directions for future research. We list a few of these directions below.

- **Chronos' time synchronization guarantees.** Our primary motivation in designing Chronos was preventing timeshifting attacks. We conjecture, however, that as Chronos relies on more time samples than today's NTP clients and also crowdsources queries across multiple servers, Chronos might improve upon today's NTP clients also in terms of time accuracy. We leave this question for future research. We do point out, however, that our results in section V-A (see Figure 12) suggest that the majority of servers in Chronos' server pools exhibit local times that are very close to each other. This suggests that averaging over the clock values of many servers is a reasonable approach for attaining time accuracy.

- **Better security guarantees.** Our analysis of Chronos' security guarantees was aimed merely to set an upper bound on the attacker's probability of success, showing that it is negligible. We believe that deeper investigation of the Chronos approach (the implications of parameter setting, more nuanced probabilistic analyses, and more) could lead to better formal security guarantees.

- **Upper bounds on the security guarantees of NTP security schemes.** Our results establish that Chronos significantly outperforms today's NTP clients in terms of resilience to MitM attacks. How close, however, is Chronos to the *optimal* level of security achievable via such security schemes? Answering this question requires setting upper bounds on the level of security attainable by *any* backwards-compatible NTP client design. Consider a model in which an NTP client can, as in our framework, repeatedly query a server pool (of legacy NTP servers) of size n and is limited to some (expected) number of queries x per time-update round. What are the optimal security guarantees of deterministic/randomized time-sampling schemes as a function of n, x, and the fraction of the server pool controlled by the attacker?

- **Weighing servers according to reputation.** Chronos' time sampling scheme treats all servers in the server pool as equal both when sampling and when averaging over samples. Another approach might be to weigh servers differently according to their reliability (e.g., assign higher weights to servers with GPS clocks), trustworthiness (e.g., servers controlled by trusted parties), and past behavior (e.g., servers with good "reputation" from past sampling iterations).

- **Benefits from changes to the server-side.** A key principle guiding Chronos' design is not relying on changes to NTP servers (only to the client side). This is needed, in our view, to render deployment easier and more realistic. Understanding, however, the implications for security of being able to modify (e.g., a small number of NTP servers) is of value and might prove beneficial.

- **Extending our approach to other time-synchronization protocols.** To improve upon NTP's time-accuracy guarantees, other approaches, such as PTP [11], [45], have been designed. While our focus is on NTP, application of our high-level approach to other time-synchronization protocols might aid in securing these alternate schemes.

We advocate the further exploration of the merits and limitations of Chronos, and the further investigation the feasibility of this new path to NTP security.

Acknowledgements

We thank the Cyber Security Research Center at Hebrew University, in conjunction with the Israeli National Cyber Bureau (INCB) in the Prime Minister's Office, for financial support. We also thank the INCB for helpful discussions. We thank the anonymous reviewers and Vyas Sekar for valuable comments. Michael Schapira and Neta Rozen Schiff are supported by an ERC Starting Grant.

References

[1] Ntp version 4.2.8p9 code, November 2016.

[2] The secure network time protocol (ntpsec) distribution, April 2017.

[3] Andreeva, O., Gordeychik, S., Gritsai, G., Kochetova, O., Potseluevskaya, E., Sidorov, S. I., and Timorin, A. A. Industrial control systems vulnerabilities statistics. Tech. rep., Kaspersky lab, 2016.

[4] Awerbuch, B., Curtmola, R., Holmer, D., Nita-rotaru, C., and Rubens, H. Mitigating byzantine attacks in ad hoc wireless networks. Tech. rep., Department of Computer Science, Johns Hopkins University, Tech, 2004.

[5] Cristian, F. Probabilistic clock synchronization. *Distributed Computing 3*, 3 (1989), 146–158.

[6] Dolev, D., Halpern, J. Y., Simons, B., and Strong, R. Dynamic fault-tolerant clock synchronization. *J. ACM 42*, 1 (Jan. 1995), 143–185.

[7] Dolev, D., Lynch, N. A., Pinter, S. S., Stark, E. W., and Weihl, W. E. Reaching approximate agreement in the presence of faults. In *Third Symposium on Reliability in Distributed Software and Database Systems, SRDS 1983, Clearwater Beach, FL, USA, October 17-19, 1983, Proceedings* (1983), IEEE Computer Society, pp. 145–154.

[8] Dolev, D., Lynch, N. A., Pinter, S. S., Stark, E. W., and Weihl, W. E. Reaching approximate agreement in the presence of faults. *J. ACM 33*, 3 (1986), 499–516.

[9] Dowling, B., Stebila, D., and Zaverucha, G. Authenticated network time synchronization. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, 2016), USENIX Association, pp. 823–840.

[10] Duan, H., Weaver, N., Zhao, Z., Hu, M., Liang, J., Jiang, J., Li, K., and Paxson, V. Hold-On: Protecting against on-path DNS poisoning. In *Securing and Trusting Internet Names* (2012), National Physical Laboratory.

[11] Estrela, P. V., and Bonebakker, L. Challenges deploying ptpv2 in a global financial company. In *2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings* (September 2012), pp. 1–6.

[12] Gerstung, H. How to configure ntp for use in the ntp pool project on ubuntu 16.04, May 2017.

[13] Goldberg, S. Why is it taking so long to secure internet routing? *Queue 12*, 8 (Aug. 2014), 20:20–20:33.

[14] Haberman, B., and Mills, D. Rfc 5906: Network time protocol version 4: Autokey specification. internet engineering task force (ietf), 2010.

[15] Halpern, J. Y., Simons, B., Strong, R., and Dolev, D. Fault-tolerant clock synchronization. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 1984), PODC '84, ACM, pp. 89–102.

[16] Herzberg, A., and Shulman, H. Fragmentation Considered Poisonous: or one-domain-to-rule-them-all.org. In *IEEE CNS 2013. The Conference on Communications and Network Security.* (2013).

[17] Hoch, D. Integrating sun kerberos and microsoft active directory kerberos, 2005.

[18] Hodges, J., and Jackson, C. Http strict transport security (hsts), November 2012.

[19] Kaminsky, D. It's the end of the cache as we know it. In *Black ops 2008*, Black Hat USA.

[20] Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21*, 7 (July 1978), 558–565.

[21] Lee, K. S., Wang, H., Shrivastav, V., and Weatherspoon, H. Globally synchronized time via datacenter networks. In *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference* (New York, NY, USA, 2016), SIGCOMM '16, ACM, pp. 454–467.

[22] Liska, A. *NTP Security: A Quick-Start Guide*. Apress, 2016.

[23] Ltd, N. M. Time traceability for the finance sector. Tech. rep., NPL Management Ltd, United Kingdom, March 2016.

[24] Malhotra, A., Cohen, I. E., Brakke, E., and Goldberg, S. Attacking the network time protocol. *IACR Cryptology ePrint Archive 2015* (2015), 1020.

[25] Malhotra, A., and Goldberg, S. Attacking ntp's authenticated broadcast mode. *SIGCOMM Comput. Commun. Rev. 46*, 2 (May 2016), 12–17.

[26] Malhotra, A., Gundy, M. V., Varia, M., Kennedy, H., Gardner, J., and Goldberg, S. The security of ntp's datagram protocol. *IACR Cryptology ePrint Archive 2016* (2016), 1006.

[27] Malhotra, A., Gundy, M. V., Varia, M., Kennedy, H., Gardner, J., and Goldberg, S. The security of ntp's datagram protocol. Tech. rep., 2016.

[28] Marzullo, K., and Owicki, S. Maintaining the time in a distributed system. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 1983), PODC '83, ACM, pp. 295–305.

[29] Marzullo, K. A. Maintaining the time in a distributed system. Tech. rep., Xerox, 1984.

[30] Marzullo, K. A. *Maintaining the Time in a Distributed System: An Example of a Loosely-coupled Distributed Service (Synchronization, Fault-tolerance, Debugging).* PhD thesis, Stanford, CA, USA, 1984. AAI8506272.

[31] Mills, D. How ntp works, 2014.

[32] Mills, D., Martin, J., Burbank, J., and Kasch, W. Rfc 5905: Network time protocol version 4: Protocol and algorithms specification. internet engineering task force (ietf), 2010.

[33] Mills, D. L. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications 39* (1991), 1482–1493.

[34] Mills, D. L., Mamakos, L., and Petry, M. Network Time Protocol (NTP). RFC 958, sep 1985.

[35] Mizrahi, T. A game theoretic analysis of delay attacks against time synchronization protocols. In *2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings* (Sept 2012), pp. 1–6.

[36] Mizrahi, T. Slave diversity: Using multiple paths to improve the accuracy of clock synchronization protocols. In *2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings* (Sept 2012), pp. 1–6.

[37] Mizrahi, T. Rfc 7384 (informational):security requirements of time protocols in packet switched networks, October 2014.

[38] Novick, A. N., and Lombardi, M. A. Practical limitations of ntp time transfer. In *2015 Joint Conference of the IEEE International Frequency Control Symposium the European Frequency and Time Forum* (April 2015), pp. 570–574.

[39] Peterson, A. Researchers say u.s. internet traffic was re-routed through belarus. that's a problem., November 2013.

[40] Rottger, S. Analysis of the ntp autokey procedures. master's thesis, technische universitt braunschweig, 2012.

[41] Selvi, J. Bypassing http strict transport security. In *Black Hat Europe* (2014).

[42] Shpiner, A., Revah, Y., and Mizrahi, T. Multi-path time protocols. In *2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings* (Sept 2013), pp. 1–6.

[43] Time, N., and Division, F. The nist authenticated ntp service, 2010.

[44] Welch, J. L., and Lynch, N. A. A new fault-tolerance algorithm for clock synchronization. *Inf. Comput. 77*, 1 (1988), 1–36.

[45] Wolfe, M. Improving network timing in financial institutions: Regulatory imperative or opportunity to achieve operational excellence? In *The ATIS Workshop on Time Sync in Financial Markets* (November 2016).