An Open-source Implementation and Security Analysis of Triad's TEE Trusted Time Protocol

Matthieu Bettinger*[§], Sonia Ben Mokhtar*, Anthony Simonet-Boulogne[†]
*INSA Lyon, CNRS, Universite Claude Bernard Lyon 1, LIRIS, UMR5205, 69621 Villeurbanne, France
{given-name}.{surname}@liris.cnrs.fr [§]Corresponding author
[†]iExec Blockchain Tech, 69008 Lyon, France {given-name}.{surname}@iex.ec

Published in the *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*
https://doi.org/10.1109/DSN-S65789.2025.00053

Abstract—**The logic of many protocols relies on time measurements. However, in Trusted Execution Environments (TEEs) like Intel SGX, the time source is outside the Trusted Computing Base: a malicious system hosting the TEE can manipulate that TEE's notion of time, e.g., jumping in time or affecting the perceived time speed. Previous work like Triad propose protocols for TEEs to maintain a trustworthy time source. However, in this paper, based on a public implementation of Triad that we contribute, we empirically showcase vulnerabilities to this protocol. For example, an attacker controlling the operating system, and consequently the scheduling algorithm, may arbitrarily manipulate their local TEE's clock speed. What is worse, in case of faster malicious clock speeds, an attacker on a single compromised machine may propagate the attack to honest machines participating in Triad's Trusted Time protocol, causing them to skip to timestamps arbitrarily far in the future. Then, infected honest machines propagate time-skips themselves to other honest machines interacting with them. We discuss protocol changes to Triad for higher resilience against such attacks.**

Index Terms—**resilience, delay attack, trusted execution environment (TEE), trusted time**

I. Introduction

Provisioning trustworthy timestamps is critical for many applications, both for traditional and confidential computing (i.e., with integrity and confidentiality requirements fulfilled using Trusted Execution Environments, TEEs, like Intel SGX [1]). Use-cases and impacts are far-reaching, ranging from TimeStamping Authorities [2] and Proof-of-Elapsed-Time [3]; credential expiration and revocation [4], [5]; latency-sensitive systems (e.g., trading [6], real-time systems [7], consistent and available databases [8]); time-constrained resource allocation (e.g., resource leasing [9]); resilience to timeout manipulation (e.g., BFT leader changes, procrastinating BFT leaders [10]); latency Quality-of-Service and resilience to malicious message delaying [11]; to decorrelation between data and timestamps in time-series [12].

In the context of confidential computing, iExec [13] proposes a decentralized computing marketplace, where anyone can contribute datasets, applications, and TEE-enabled hardware. These computing assets can then be matched by anyone to execute tasks, i.e., a given application's logic processing datasets while running on a server. TEEs on those servers are in place to guarantee execution integrity and confidentiality. However, applications are user-defined and therefore arbitrary: they can rely on timestamps. If those timestamps can be manipulated by a server's malicious owner, e.g., because they control the OS or hardware, then task results themselves can be manipulated, e.g., as part of use-cases presented above.

Indeed, with CPU-level TEEs, also called enclaves, like Intel SGX, the time source is outside the TCB: since 2020, the `sgx_get_trusted_time` primitive [14] is deprecated. Some previous trusted time mechanisms relied on this (e.g., TimeSeal [15]) or other deprecated primitives (e.g., on Intel TSX [9], [16]). In recent years, new protocols have been proposed to address trusted time in Intel SGX, like T3E [17] and Triad [18]. While T3E is open-source [19], the original Triad protocol is closed-source (due to its implementation on top of the proprietary containerization solution Scone [20]): we have been able to reach Triad's authors, but not to obtain implementation artifacts or supplementary specifications. Therefore, in this paper, based on the specifications in the original Triad paper [18], we contribute a public implementation [21] of Triad and reproduce results. However, we also experimentally highlight attacks on the Triad protocol (the original paper does not include empirical evaluation of attack scenarios). While Triad makes a cluster of TEEs collaborate to keep a shared notion of trusted time, assuming that all underlying OSs or hypervisors may be compromised, we show that even a single compromised node can manipulate its own notion of time and cause honest nodes to skip to timestamps arbitrarily far in the future. This effect then cascades to honest nodes contacting infected honest nodes, infecting them in turn. Consequently, we also discuss protocol improvements towards a TEE trusted time protocol with higher resilience.

This paper is structured as follows: first, Section II presents related work on TEE trusted time mechanisms; then, Section III describes the Triad protocol and attacks upon it; based on the contributed implementation of Triad, Section IV reproduces results in the original Triad paper and additionally

This work was supported by a French government grant managed by the Agence Nationale de la Recherche under the France 2030 program, reference "ANR-22-PEFT-0002" as well as the ANR Labcom program, reference "ANR-21-LCV1-0012". We thank the Complex Systems research group at the IIUN (University of Neuchâtel, Switzerland) for granting us access to their experimental platform.

illustrates the feasibility and impact of attacks on the Triad protocol; finally, Section V discusses protocol improvements to increase resilience to these attacks and Section VI summarizes this work.

II. Related work

Time sources, like the TimeStamp Counter (TSC) on a CPU or a remote Time Authority (e.g., NTP time servers [22], [23]), are outside the TCB of TEEs. Therefore, security mechanisms must be put in place to prevent timestamp tampering by an attacker, positioned at the hypervisor, OS, or on the network. Hereafter, we describe recent solutions and their features.

A. CPU-level TEE trusted time

With Standard Intel SGX (SGX1), reads of the TSC with `rdtsc` are mediated by the OS, which can manipulate the value. Meanwhile, with Scalable Intel SGX (SGX2), `rdtsc` instructions can be executed in-enclave, bypassing the OS (but remaining vulnerable to a malicious hypervisor). Furthermore, a malicious OS controls the scheduling of enclaves, so enclaves may be interrupted arbitrarily. Additions to Intel SGX like AEX-Notify [24] enable the TEE to react to interruptions and handle them with arbitrary developer-defined code after the enclave resumes. Triad [18] makes a cluster of TEEs cooperate to keep a common and continuous notion of time. Each TEE monitors its TSC and relies on AEX-Notify to detect when its notion of time continuity is severed: the TEE then either obtains a timestamp from a peer enclave in the cluster or, failing that, from a Time Authority (e.g., NTP time servers). However, in this paper, we show vulnerabilities in Triad's calibration protocol and in communications between TEEs in the cluster, allowing time manipulations at single nodes and their propagation to others. T3E [17] uses a Trusted Platform Module [25] as a time source, colocated with the TEE. T3E hinders delaying messages coming from the TPM by limiting how many times the same timestamp can be used by the TEE and by stalling TEE execution if uses are depleted. In turn, the underlying application will drop in throughput, which may be detected by that application's user. However, quantifying appropriate numbers of uses, to neither block execution when there are no attacks nor give too much room for delay attacks, is complex, because effective ideal usage rates are code-, workload-, and hardware-dependent. Further, if the application is non-interactive or, on the contrary, if there are many users consuming a remote TEE service, with some users who may be malicious, trustworthy monitoring of demanded and effective throughput is again difficult. Additionally, the TPM can be configured by an attacker owning it (leading to up to a $\pm 32.5\%$ drift-rate compared to real time [25]) and may more generally be a target for attacks as a root of trust [26].

B. VM-level TEE trusted time

VM-level TEEs like Intel TDX have started becoming available at Cloud Service Providers (e.g., Microsoft Azure [27], Google Cloud [28], IBM Cloud [29]). With the TCB now comprising the operating system, attackers must devise new strategies to harm the system. Notably, with respect to trusted time, TEEs like Intel TDX and AMD SEV-SNP have their time sources protected even against a malicious hypervisor, respectively with their virtualized TSC [30] and SecureTSC [31] features. With Intel TDX, writing in the TimeStamp Counter's registers is forbidden from inside the Trust Domain (TD), i.e., the guest VM. A hypervisor offsetting the TSC during a VM exit is similarly detected and results in an error upon VM entry [30]. Meanwhile, AMD's SecureTSC lets the hypervisor and VM guests modify the TSC without affecting other guests, whose TSC remains linearly increasing [31]. However, VM-level TEEs' attack surface is still undergoing research [31]–[35] and a large TCB is more demanding to properly audit. Our objective in this paper is therefore to get closer to the guarantees provided by VM-level trusted time mechanisms, but using CPU-level TEEs with a smaller TCB.

III. Triad specifications & attacks

Now, we describe Triad [18], the attack vectors it originally aimed to mitigate, as well as our new attacks against Triad.

A. Attacker model

The attacker in Triad is assumed to control the operating system or hypervisor. Notably, it can delay or drop any message between the TEE and other devices. By controlling the OS, it may also arbitrarily cause interruptions. Interestingly, while the original paper considers *adding* interruptions, removing interruptions, e.g., by further isolating cores running Triad, is not mentioned. We show in experiments how low interruption rates help strengthen some attacks. Finally, regarding Triad's use of the TSC as a local time source, a hypervisor virtualizing the TSC may change its value's offset and scaling factor for the guest VM running a Triad node.

B. Triad protocol and building blocks

To protect against such an attacker, Triad uses the following building blocks. First, to prevent arbitrary manipulations of the TSC, a thread in the TEE is dedicated to monitoring the TSC increment rate. With SGX2, reading the TSC does not require exiting the enclave: as long execution remains in the enclave, the OS or hypervisor cannot manipulate the TSC read by the enclave. Triad calibrates the monitoring thread by measuring increases in the TSC during uninterrupted executions of that thread. AEX-Notify [24] enables the TEE to be aware of when such interruptions occur, called Asynchronous Enclave Exits (AEXs). More precisely, arbitrary user logic can be triggered when a TEE thread resumes execution.

Once an AEX occurs, however, the timestamp is considered "tainted": an arbitrarily long time may pass before TEE execution resumes and the attacker may offset the TSC to make that duration seem shorter or even longer. As a consequence, the TEE communicates with remote entities to refresh, to "untaint" its timestamp. The root of trust is a remote Time Authority (TA), e.g., an NTP server, which serves as the time reference. Remote communications introduce network delays and the TEE is unavailable to client applications while its timestamp is tainted. For shorter roundtrip delays and fewer requests to the TA, Triad nodes are organized in clusters of multiple TEEs. After resuming from an interruption, a TEE first asks its peers in the cluster for a timestamp and only asks the TA upon failure to receive any responses from peers.

We now focus on two key protocol steps in Triad: TSC monitoring calibration and untainting using peer timestamps.

C. Attacking Triad's calibration protocol

A critical aspect of Triad's calibration is to estimate the relationship between the passage of time with respect to the TA's reference clock and increments in the TSC. To do so, Triad relies on roundtrip communications with the TA, bounded by the monitoring thread's continuous execution, i.e., between two AEXs. For example, the TEE may ask the TA to wait 1s before sending back the response. Meanwhile, the monitoring thread checks its uninterrupted execution and reports the TSC increment once the TA's response arrives.

However, the TEE is not aware *a priori* of how much reference time can pass between two AEXs and, as a consequence, cannot reliably bound the requested TA waittime by inter-AEX delays. Even given such an estimate, e.g., using the TSC frequency measurement by the OS at boot-time, the effective network delay is also unknown by the TEE, giving an attacker a margin of freedom to delay messages. The original paper's specifications do not fully define how calibration should be performed, besides repeated and independent short interactions with the TA which waits a requested amount of time s before sending a response. To account for the offset error introduced by network delays, we consider a linear regression over requested waittimes and measured TSC increments. The slope is the TSC's increment rate with respect to the TA's reference time. Without regression over multiple measurements, e.g., with only the mean obtained with long waittimes, the offset error would always overestimate the TSC's increment rate, i.e., slow the TEE's perceived clock speed. In the original paper's experiments, some nodes do have positive drifts from the reference, i.e., their calibration does compensate the offset created by network delays. Note that Triad's measurements over short intervals can lead to poor precision in estimating the clock speed, even without attacks. As a comparison, instead of measurements of around 1s, NTP uses long drift measurement timeframes, between 16s and 36h [22].

Based on the above considerations, we design the following "F+" and "F–" attacks that respectively increase and decrease a node i's perceived TSC increment rate F_i^{calib} compared to its real rate F_i^{TSC}, i.e., slow down or quicken the TEE's perceived passage of time. A TEE, as part of its TSC speed calibration protocol, sends messages to the TA, which waits a requested duration s included in the message. Communications are authenticated and encrypted, so the attacker does not have access to s. However, the attacker is able to measure network delays between its machine and the TA, as well as roundtrip times part of Triad's calibration protocol, so the attacker can estimate s. To slow down the TEE's clock, the attacker causes a steeper regression, i.e., $F_i^{\text{calib}} > F_i^{\text{TSC}}$, by adding delays to messages with high s, which we call an F+ attack. Conversely, in an F– attack, i.e., with $F_i^{\text{calib}} < F_i^{\text{TSC}}$ and leading to a faster TEE clock, the attacker adds delays to messages with low s.

D. Propagating the attack to TEE peers

When the TEE is not calibrating, it asks its peers for timestamps upon resuming after an AEX. If any peers are not also "tainted", they send their current timestamp. In the original Triad protocol [18], the policy to handle peer timestamps is as follows: if a received timestamp is higher than the local one (the last one before the interrupt), then the incoming timestamp becomes the new reference; otherwise, the local timestamp is increased by the smallest possible increment to ensure monotonicity when serving client applications. Such a policy ensures that TEEs cannot go back in time. However, it also means that all TEEs will follow the fastest clock in the cluster. Such drift errors can persist, because the TA is only contacted if all TEEs are "tainted" at the same time.

IV. Results

Based on these specifications, we implement the protocol in C++ for Intel SGX TEEs, as well as F+ and F– attacks. The source code is available [21]. All protocol communications use UDP and are encrypted using AES-256-GCM [36]. In the provided implementation, TSC rate estimation is performed through regression over roundtrips of messages with 0s-sleep (immediate responses) and 1s-sleep at the TA.

Research questions (RQ) in this paper are along two axes:

A. Can the proposed public implementation reproduce results in Triad's original paper?

B. Given this implementation based on the original specifications, how resilient is Triad to presented attacks?

More precisely, reproducibility is explored according to the following questions:

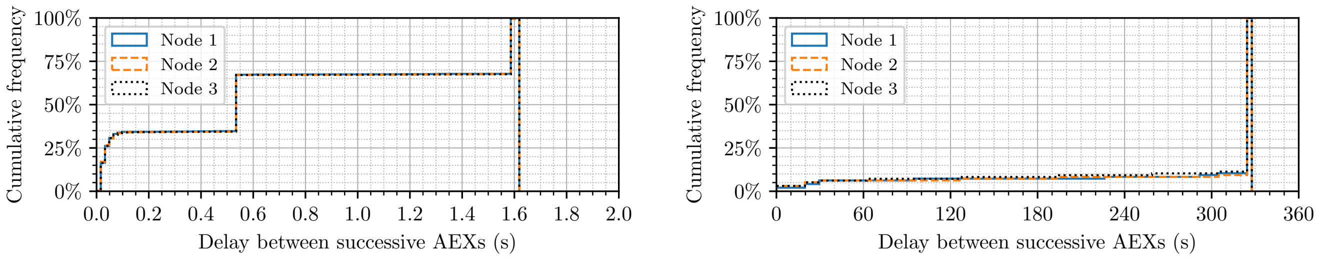
A.1 How accurate is TSC-monitoring using a counter in an enclave thread?

A.2 Without attacks, how available are Triad nodes and how much do they drift compared to a reference time source?

Note that we can only compare results without attacks, as the original paper does not provide evaluation under attacks. Next, we attack the Triad protocol ourselves and we evaluate the drift of nodes participating in the protocol, by launching F+ and F– attacks from a single compromised Triad node.

For the following experiments, we run three Triad nodes and the TA on an Intel SGX2 (Scalable SGX) machine with 32 cores. The TSC monitoring thread for each node is pinned to a core isolated from most OS interruptions. As a result, those monitoring threads experience delays between AEXs as illustrated in Figure 1b: most AEXs occur every 5.4 minutes.

To properly reproduce Triad's results, we simulate their distribution of inter-AEX delays (10ms, 532ms, and 1.59s, each with probability $1/3$), which we called "Triad-like". The resulting distribution on our machine is shown in Figure 1a, approximating Triad's original environment. We do not have information on correlations that existed in their setup's successive delays between AEXs: we assume in this work that their



(a) Triad-like [18] simulated interruption distribution.

(b) TSC monitoring core isolated from most OS interruptions.

Fig. 1: Cumulative distribution of delays between successive Asynchronous Enclave Exits (AEXs) on the TimeStamp Counter (TSC) monitoring enclave thread. Figure 1a is simulated on top of Figure 1b's system environment, by triggering AEXs at the TSC monitoring thread's core, using `rdmsr` instructions on that core's TSC MSR (Model Specific Register, `0x10` for TSC).

successive delays were independent, i.e., $P(D_{i+1} = d) = P(D_{i+1} = d|D_i)$, $\forall D_i, d$, with D_i the duration between the i^{th} and $(i+1)^{\text{th}}$ AEXs, and $d \in \{10, 532, 1590\}$ms.

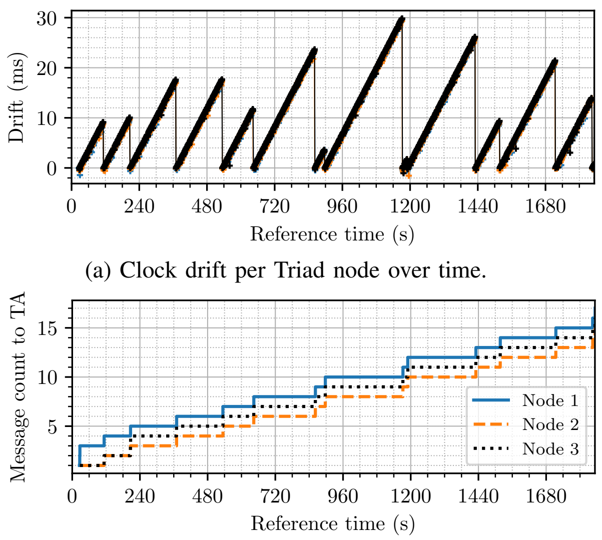
Finally, to save space and avoid legends hiding data points, we do not repeat legends for every figure. However, legends are consistent across figures, i.e., Nodes 1, 2, and 3 are always represented respectively in blue, orange, and black. Note also that Nodes 1 and 2 are both honest in all experiments: Node 1's data points may overlap and hide Node 2's data points.

A. *Reproducing Triad [18] results without attacks*

We now show our reproducibility results based on our public implementation [21] of the Triad protocol.

1) TSC monitoring with TEE enclave `INC`-counters: With a fixed core-frequency, we run 10k measurements counting `INC` instructions until the TSC has incremented by 15E6, representing around 5ms in realtime (the TSC increments at $F^{\text{TSC}} = 2899.999$MHz, as measured by the OS at boot-time). Additionally, we have set the monitoring thread's core with the "performance" frequency-scaling governor (i.e., it runs at maximum frequency: 3500Mhz), we obtain 632181`INC` as mean with 109.5`INC` of standard deviation. Removing two outliers (the experiment's first run with 621448`INC` and another with 630012`INC`), we obtain 632182`INC` as mean and go down to 2.9`INC` of standard deviation. The range between measurement values without the two outliers is of 10`INC`: a monitoring thread running at a fixed frequency can therefore reliably detect TSC discrepancies, both in speed or time jumps (forward and back in time). To answer *RQA.1*, given that Intel CPUs allow only discrete pre-determined frequency settings [37], coupling this accurate but frequency-dependent monitoring with a less accurate but frequency-independent monitoring (e.g., memory- [18] or randomness- [9] accesses) may lock an attacker from manipulating the TSC rate and offset. However, we will show in later experiments that this mechanism is not sufficient to protect against an attacker manipulating the TEE's time perception: the attacker can still impact what duration of real elapsed time is equated to a number of TSC increments.

2) Triad nodes availability and drift rates: Here, we assess Triad nodes' availability as well as drift rates they experience



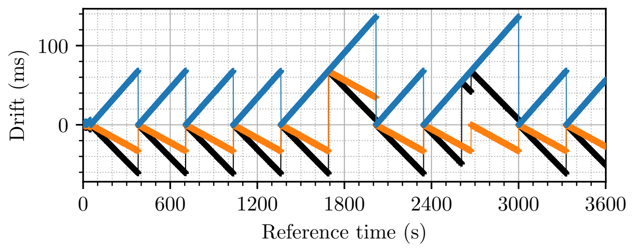
(a) Clock drift per Triad node over time.

(b) Number of received time references from the Time Authority.

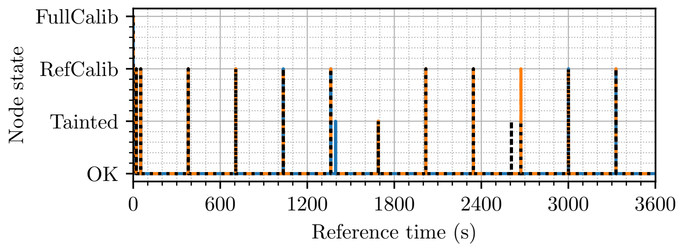
Fig. 2: Long-term fault-free behavior of Triad nodes under Figure 1a's AEX delay distribution ($F_1^{\text{calib}} = 2900.089$MHz; $F_2^{\text{calib}} = 2900.113$MHz; $F_3^{\text{calib}} = 2899.653$MHz).

compared to the TA's reference time. In all experiments, all nodes only required to perform full calibration, i.e., both clock time reference and speed, once with the TA. Figure 3b illustrates this, showing the first hour of an experiment lasting 8h and a single stay in the "FullCalib" state at the start of the experiment. The TSC was not manipulated during these experiments and monitoring cores ran at maximum frequency: no discrepancies in TSC update rates were detected by the TSC monitoring enclave thread's `INC`-instruction-counting. Otherwise, additional full calibrations would have been triggered.

Regarding drift, NTP's standard allowed clock drift-rate is 15ppm (parts-per-million, i.e., 15µs/s or 1.3s/day) [22]. In our scenarios without attacks, e.g., in Figure 2a, all nodes drift at around 110ppm (0.11ms/s), while Node 1 drifts at 210ppm in Figure 3a. These *effective* drift-rates are an order of magnitude higher than the standard *upper bound* drift-rates. This can be attributed to Triad's calibration protocol based on



(a) Clock drift per Triad node over time.



(b) Timing diagram of Triad node states.

Fig. 3: Long-term fault-free behavior of Triad nodes under Figure 1b's AEX delay distribution ($F_1^{\text{calib}} = 2899.363$MHz; $F_2^{\text{calib}} = 2900.260$MHz; $F_3^{\text{calib}} = 2900.510$MHz).

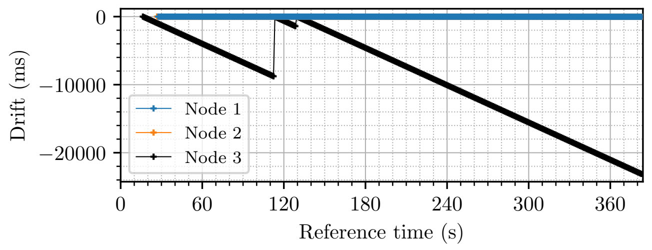


Fig. 4: Clock drift of Triad nodes under an F+ attack on Node 3 ($F_3^{\text{calib}} = 3191.224$MHz; $F_1^{\text{calib}} = 2900.223$MHz; $F_2^{\text{calib}} = 2900.595$MHz), which is in Figure 1b's low AEX environment, while Nodes 1 and 2 experience Figure 1a's Triad-like AEXs.

measurements over short durations, in the order of seconds or less, while standard clock synchronization like NTP monitor drift over long timeframes [22]: 2^τs, with $\tau \in [\![4, 17]\!]$, i.e., 16s to 36h. Clock drifts reset to 0 in Figure 2a when a node's message count to the TA increments in Figure 2b, i.e., when the node calibrates its time reference with the TA in absence of peer responses. Figures in the original paper do not allow estimating their drift rates.

Nodes are unavailable to serve timestamps if they are tainted or calibrating: for Figure 2's 30min experiment without attacks, each node's availability to serve timestamps exceeds 98% including initial calibration. Figure 3's experiment over 8 hours shows that availability can rise to 99.9%.

In a low-AEX setting, monitoring cores experience only a few AEXs with minutes of delay between them, as previously shown in Figure 1b. However, in our OS setup, these specific remaining interruptions by the OS do not target select individual cores but rather all cores. This means that because all three nodes run on the same machine's cores, even under simulated AEXs, their TSC monitoring threads sometimes experience an AEX simultaneously (with higher probability than the original Triad experiment setup). As a result, their timestamps will become tainted at the same time and they will not be able to fetch a fresh one from their peers: they must contact the TA, resetting their drifts. This behavior also explains the sawtooth pattern of each node's drift time-series in Figure 2a.

Compared to Figure 2, without those correlated simultaneous AEXs, we can expect the node i which underestimates the TSC frequency F_i^{calib} the most to lead all other nodes to drift positively, i.e., to perceive a faster passage of time, even without attacks. A low-AEX environment helps showcase this behavior, i.e., in Figure 3 at reference times $t = 1705$s for Node 2 and $t \in \{1705, 2623, 2688\}$s for Node 3. See how in Figure 3b, those nodes do not perform a time reference calibration with the TA ("RefCalib"), but were able to switch from a tainted to an OK state using peer untainting. This results in time jumps for Nodes 2 and 3 to Node 1's increased clock time, i.e., in Figure 3a, by 50–70ms, followed by the nodes resuming timestamp updates at their own clock speeds.

To summarize and answer *RQA.2*, Triad nodes exhibit high availability to serve timestamps. However, even without attacks, a node underestimating F^{TSC} is able to lead all other nodes to follow its drift, e.g., in Figure 2a, Node 3's drift (we have $F_3^{\text{calib}} < F^{\text{TSC}} < F_1^{\text{calib}} < F_2^{\text{calib}}$). Furthermore, without simultaneous AEXs, this can happen arbitrarily long.

B. *Triad [18] under attacks*

Hereafter, we launch F+ and F– attacks on a single Triad node among the three and observe the system's behavior.

1) Node 3 launching an F+ attack: To start, we slow down the perceived time speed at Node 3 using an F+ attack, adding a 100ms delay to the TA's 1s-sleep messages. In Figure 4, Node 3 is additionally set in a low-AEX environment. Besides two calibrations with the TA (due to correlated simultaneous AEXs on all cores), Node 3 drifts at -91ms/s (-9.1E4ppm).

In Figure 5, Node 3 experiences Triad-like AEXs. The calibrated frequency is nearly the same as in the previous case (with a 4E-6 relative difference). Node 3's drift now oscillates between two bounds: Nodes 1 and 2's drifts when it obtains peer timestamps after an AEX; and -150ms when it updates timestamps based on its own slow clock in-between AEXs.

2) Node 3 launching an F– attack: Now, we quicken perceived time speed at Node 3 using an F– attack, adding a 100ms delay to TA's immediate (0s-sleep) messages. Node 3 drifts positively at 113ms/s. To better highlight that attack's impact, Nodes 1 and 2 start the experiment with rare AEXs, then, after 104s (dashed red line in Figure 6), they experience Triad-like AEXs. Figure 6b shows the total number of AEXs at each node over time: while Node 3's AEXs linearly increase from the start, Nodes 1 and 2's AEXs stay around 0 for $t < 104$s then also linearly increase. Figure 6a illustrates the nodes' drift before and after the change in AEX rates. Without AEXs, both Nodes 1 and 2 experience relatively low drift-rates, similar to the case without attacks. However, with higher

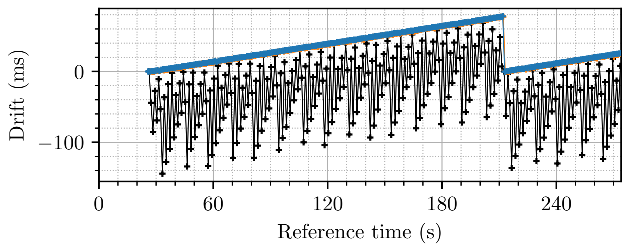
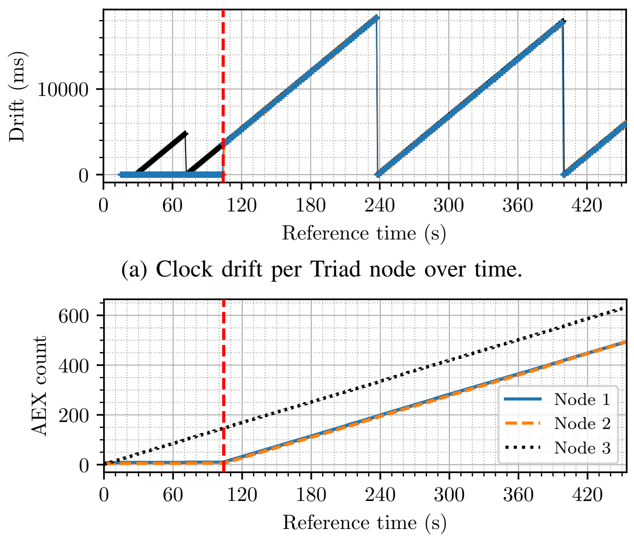


Fig. 5: Clock drift of Triad nodes under an F+ attack on Node 3 ($F_3^{\text{calib}} = 3191.210$MHz; $F_1^{\text{calib}} = 2898.751$MHz; $F_2^{\text{calib}} = 2900.836$MHz), with all nodes experiencing Figure 1a's Triad-like AEXs.



(a) Clock drift per Triad node over time.

(b) Number of AEX over time.

Fig. 6: Triad nodes behavior under an F– attack on Node 3 ($F_3^{\text{calib}} = 2609.951$MHz; $F_1^{\text{calib}} = 2899.347$MHz; $F_2^{\text{calib}} = 2900.052$MHz), which experiences Figure 1a's Triad-like AEXs, while Nodes 1 and 2 experience Figure 1b's low AEXs for reference time $t < 104$s, then Triad-like ones for $t \geq 104$s, i.e., after the dashed red line.

AEX rates once $t > 104$s, both nodes now communicate with the compromised Node 3 and use its timestamps, because they are farther in time than theirs. As a result, both nodes jump forward in time, e.g., by around 35ms at $t = 104$s, then alternate between their own clock's timestamps between AEXs and forward time jumps after AEXs.

Finally, to answer *RQB*, we observe that Triad is particularly sensitive to F– attacks speeding up a single TEE's perceived passage of time, because that TEE propagates its positive drift to TEE peers. With frequent AEXs, slowing down time with an F+ attack is less impactful, causing only the compromised node to oscillate between its peers' timestamps and its own slow clock measurements. However, an attacker controlling the OS can prevent AEXs altogether, causing an arbitrary negative drift to that compromised node. Additionally, these attacks on a node's calibration do not negatively affect availability: AEXs dictate how often the node communicates with peers and the TA. In fact, as a consequence, a lower AEX rate, for example used above to strengthen an F+ attack, increases availability.

V. Discussion

In this section, we discuss our empirical results and propose protocol changes to address existing vulnerabilities.

First, in the original protocol, events to refresh a TEE's timestamp only come from outside the TCB, from attacker-controlled OS interruptions. A compromised node may use its miscalibrated clock speed arbitrarily long. To reduce the attack power, an in-TCB trigger can be added, e.g., using deadlines after given numbers of TSC increments. As before, when a deadline is reached, the enclave will try to check its timestamp's quality or obtain a better one.

Moreover, the base protocol's synchronization precision and accuracy for uncompromised nodes should be close to that of a system without attacks. With Triad [18], nodes with honest OSs instead use a calibration protocol designed to restrain an attacker's power, with a cost in synchronization quality. Triad's calibration phases with short-duration measurements of clock speed and offset can be replaced by more mature synchronization protocols like NTPsec [23]. If honest, uncompromised nodes exist, they will be able to calibrate high-quality clocks over time. Standard synchronization protocols use the notion of consistency in (sub-)sets of clocks in a system [38]. Given clocks with timestamps t_i and e_i an estimation on each clock's possible drift error, consistent subsets of clocks have all their intervals $t_i \pm e_i$ overlap with a non-empty intersection. These clocks are usually called true-chimers. Additionally, this method can be applied to clock time and speed.

With such a synchronization protocol, a node may now check if its clock is consistent with the TA. If the node is compromised, messages may be delayed by the attacker: it may be consistent with a time reference offset towards the past. However, honest nodes communicating with the compromised node will not consider it a true-chimer. Nodes may publish, e.g., on a blockchain, or simply to other nodes, their list of true-chimers. Nodes with the highest timestamp obtained from the TA have the most credibility to be honest. Furthermore, many secure distributed protocols already rely on the assumption of an honest (super-)majority. Under such an assumption, a majority clique of true-chimers may be used to maintain clock consistency and rely less often on the TA.

Ongoing work is dedicated to implementing and evaluating a protocol that builds upon these specifications.

VI. Conclusion

Many traditional and confidential computing protocols rely on trustworthy provisioning of timestamps. However, current TEEs like Intel SGX lack a built-in trusted time mechanism. We contribute a public implementation of the recent but closed-source Triad protocol, which aims to provide trusted time to a cluster of Intel SGX TEEs. We show vulnerabilities in the protocol, e.g., that enable a single compromised node to propagate faster passage of time to honest nodes. Finally, we discuss the found vulnerabilities' sources and propose changes for higher resilience against such attacks. Future work is dedicated to a protocol that implements those changes.

References

[1] V. Costan and S. Devadas, "Intel SGX explained," 2016. [Online]. Available: https://eprint.iacr.org/2016/086

[2] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato, "RFC3161: Internet x.509 public key infrastructure time-stamp protocol (TSP)," USA, 2001.

[3] M. Bowman, D. Das, A. Mandal, and H. Montgomery, "On elapsed time consensus protocols," in *Progress in Cryptology – INDOCRYPT 2021*, A. Adhikari, R. Küsters, and B. Preneel, Eds. Cham: Springer International Publishing, 2021, p. 559–583.

[4] F. Alder, G. Scopelliti, J. Van Bulck, and J. T. Mühlberg, "About time: On the challenges of temporal guarantees in untrusted environments," in *Proceedings of the 6th Workshop on System Software for Trusted Execution*, ser. SysTEX '23. New York, NY, USA: Association for Computing Machinery, Jun. 2023, p. 27–33. [Online]. Available: https://doi.org/10.1145/3578359.3593038

[5] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg, "Attacking the network time protocol," in *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.

[6] A. Addison, C. Andrews, N. Azad, D. Bardsley, J. Bauman, J. Diaz, T. Didik, K. Fazliddin, M. Gromoa, A. Krish, R. Prins, L. Ryan, and N. Villette, "Low-latency trading in the cloud environment," in *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, Aug. 2019, p. 272–282.

[7] J. Wang, A. Li, H. Li, C. Lu, and N. Zhang, "RT-TEE: Real-time system availability for Cyber-physical Systems using ARM TrustZone," in *2022 IEEE Symposium on Security and Privacy (SP)*, May 2022, p. 352–369.

[8] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's globally distributed database," *ACM Transactions on Computer Systems*, vol. 31, no. 3, p. 1–22, Aug. 2013.

[9] B. Trach, R. Faqeh, O. Oleksenko, W. Ozga, P. Bhatotia, and C. Fetzer, "T-Lease: a trusted lease primitive for distributed systems," in *SoCC '20: ACM Symposium on Cloud Computing, Virtual Event, USA, October 19-21, 2020*, R. Fonseca, C. Delimitrou, and B. C. Ooi, Eds. ACM, 2020, pp. 387–400. [Online]. Available: https://doi.org/10.1145/3419111.3421273

[10] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "RBFT: Redundant byzantine fault tolerance," in *2013 IEEE 33rd International Conference on Distributed Computing Systems*, Jul. 2013, p. 297–306. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6681599

[11] M. Bettinger, E. Rivière, S. Ben Mokhtar, and A. Simonet-Boulogne, "COoL-TEE: Client-TEE collaboration for collusion-resilient distributed search," in *25th IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, May 2025.

[12] A. Nasrullah and F. M. Anwar, "Trusted Timing Services with Timeguard," in *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2024, pp. 1–14. [Online]. Available: https://ieeexplore.ieee.org/document/10568071/

[13] G. Fedak, H. He, M. Morca, W. Bendella, and E. Alves, "iExec Blockchain-Based Decentralized Cloud Computing," iExec Blockchain Tech, White Paper, 2018. [Online]. Available: https://github.com/iExecBlockchainComputing/whitepaper/blob/master/V3/iExec-WPv3.0-English.pdf

[14] S. Cen and B. Zhang, "Trusted time and monotonic counters with intel® software guard extensions platform services," https://www.intel.com/content/www/us/en/content-details/671564/trusted-time-and-monotonic-counters-with-intel-software-guard-extensions-platform-services.html, 2017.

[15] F. M. Anwar, L. Garcia, X. Han, and M. Srivastava, "Securing time in untrusted operating systems with TimeSeal," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2019, p. 80–92. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9052115

[16] F. Alder, N. Asokan, A. Kurnikov, A. Paverd, and M. Steiner, "S-FaaS: Trustworthy and accountable function-as-a-service using Intel SGX," in *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, ser. CCSW'19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, p. 185–199. [Online]. Available: https://doi.org/10.1145/3338466.3358916

[17] G. M. Hamidy, P. Philippaerts, and W. Joosen, "T3E: A Practical Solution to Trusted Time in Secure Enclaves," in *Network and System Security*, S. Li, M. Manulis, and A. Miyaji, Eds. Springer Nature Switzerland, 2023, vol. 13983, pp. 305–326. [Online]. Available: https://link.springer.com/10.1007/978-3-031-39828-5_17

[18] G. Fernandez, A. Brito, and C. Fetzer, "Triad: Trusted Timestamps in Untrusted Environments," in *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2023, pp. 169–176. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10475818

[19] G. Hamidy and P. Philippaerts, "TPM-based Trusted Time Extensions (T3E)," 2023. [Online]. Available: https://github.com/DistriNet/T3E

[20] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure linux containers with intel SGX," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 689–703.

[21] M. Bettinger and J. Acker, "Triad tee trusted time public implementation," https://github.com/RedChainLab/Triad-TEE-Trusted-Time, 2025.

[22] D. L. Mills, *Network Time Protocol (NTP)*, Sep. 1985, no. RFC 958. [Online]. Available: https://datatracker.ietf.org/doc/rfc958

[23] E. S. Raymond, "NTPsec: a secure, hardened NTP implementation." [Online]. Available: https://dl.acm.org/doi/fullHtml/10.5555/3014186.3014187

[24] S. Constable, J. V. Bulck, X. Cheng, Y. Xiao, C. Xing, I. Alexandrovich, T. Kim, F. Piessens, M. Vij, and M. Silberstein, "AEX-Notify: Thwarting precise Single-Stepping attacks through interrupt awareness for intel SGX enclaves," 2023, p. 4051–4068. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/constable

[25] TPM 2.0 Library. Trusted Computing Group. [Online]. Available: https://trustedcomputinggroup.org/resource/tpm-library-specification/

[26] B. Parno, "Bootstrapping trust in a "trusted" platform," in *3rd USENIX Workshop on Hot Topics in Security, HotSec'08, San Jose, CA, USA, July 29, 2008, Proceedings*, N. Provos, Ed. USENIX Association, 2008.

[27] R. Echevarria. (2023, Nov.) https://community.intel.com/t5/Blogs/Tech-Innovation/Cloud/Microsoft-Azure-Adds-Confidential-VMs-to-Expand-Options-for/post/1543740.

[28] A. P. (2024, Oct.) https://community.intel.com/t5/Blogs/Products-and-Solutions/Security/Intel-and-Google-Cloud-Announce-Confidential-VMs-for-the-Masses/post/1634718.

[29] M. Veramonti. (2025, Jan.) https://community.ibm.com/community/user/cloud/blogs/meryl-veramonti/2025/01/16/Intel-TDX-Beta.

[30] "Intel® tdx module base architecture specification," https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html, Intel Corporation, Specification, 2024.

[31] S. R. Neela, "Like clockwork: A systematic analysis of AMD SEV-SNP's securetsc," Graz University of Technology, Master's thesis, 2024.

[32] S. Gast, H. Weissteiner, R. L. Schröder, and D. Gruss, "CounterSEVeillance: Performance-counter attacks on AMD SEV-SNP: Network and distributed system security symposium 2025," *Network and Distributed System Security (NDSS) Symposium 2025*, Feb. 2025.

[33] L. Wilke, J. Wichelmann, A. Rabich, and T. Eisenbarth, "SEV-Step a single-stepping framework for AMD-SEV," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 1, p. 180–206, Dec. 2023.

[34] U. Mandal, S. Shukla, N. Mishra, S. Bhattacharya, P. Saxena, and D. Mukhopadhyay, "Exploring side-channels in intel trust domain extensions," no. 2025/079, 2025, publication info: Preprint. [Online]. Available: https://eprint.iacr.org/2025/079

[35] L. Wilke, F. Sieck, and T. Eisenbarth, "TDXdown: Single-stepping and instruction counting attacks against Intel TDX," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, Dec. 2024, p. 79–93.

[36] M. M. Li, "Maxul/sgx-aes-256," 2020. [Online]. Available: https://github.com/Maxul/SGX-AES-256

[37] "Intel® 64 and ia-32 architectures software developer's manual combined volumes." [Online]. Available: https://www.intel.com/content/www/us/en/content-details/782158/intel-64-and-ia-32-architectures-software-developer-s-manual-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4.html

[38] K. Marzullo and S. Owicki, "Maintaining the time in a distributed system," 1983.