About Time: On the Challenges of Temporal Guarantees in Untrusted Environments

Fritz Alder
fritz.alder@acm.org
imec-DistriNet, KU Leuven, Belgium

Gianluca Scopelliti
gianluca.scopelliti@ericsson.com
Ericsson Security Research, Sweden
imec-DistriNet, KU Leuven, Belgium

Jo Van Bulck
jo.vanbulck@cs.kuleuven.be
imec-DistriNet, KU Leuven, Belgium

Jan Tobias Mühlberg
jan.tobias.muehlberg@ulb.be
Université Libre de Bruxelles, Belgium

Abstract

Measuring the passage of time and taking actions based on such measurements is a common security-critical operation that developers often take for granted. When working with confidential computing, however, temporal guarantees become more challenging due to trusted execution environments residing in effectively untrusted environments, which can oftentimes influence expectations on time and progress. In this work, we identify and categorize five different levels of tracking the passage of time that an enclave may be able to mesure or receive from its environment. Focusing first on the popular Intel SGX architecture, we analyze what level of time is possible and how this is utilized in both academia and industry projects. We then broaden the scope to other popular trusted computing solutions and list common applications for each level of time, concluding that not every use case requires an accurate access to real-world time.

ACM Reference Format:
Fritz Alder, Gianluca Scopelliti, Jo Van Bulck, and Jan Tobias Mühlberg. 2023. About Time: On the Challenges of Temporal Guarantees in Untrusted Environments. In *6th Workshop on System Software for Trusted Execution (SysTEX '23), May 8, 2023, Rome, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3578359.3593038

1 Introduction

Since the rise of atomic clocks, the precise progression of time is measured by examining the radioactive decay of atoms. In computing, this is often simulated by quartz clocks that trade a lower accuracy for affordability in everyday technology. To enable the use of time in human communication, standards like coordinated universal time (UTC) define the precise time at a given moment as points of reference. However, what is colloquially called *wall-clock time* is an inherently human-made concept that is agreed upon by social convention. Unix systems, for example, translate UTC to *system time* as the number of passed seconds since the 1st of January 1970. As such, wall clock time in computing is a challenging task that is nowadays dealt with by utilizing time servers and running time synchronization protocols.

In the area of secure systems, recent years have seen the rise of capable confidential computing solutions that can protect data in use, such as Intel SGX, AMD SEV, Intel TDX, and ARM CCA. The key idea behind these trusted execution environments (TEEs) is to protect critical code and data in so-called *enclaves* that are isolated from the rest of the system stack. Particularly, while the privileged operating system and hypervisor remain in charge of enclave resource management and availability, any direct access to enclave code or data is prevented by means of hardware-level access control logic. TEEs can, hence, effectively protect sensitive data while it resides in memory. However, enclaves may still request untrusted network communication from the potentially hostile operating system to perform computations.

The challenge of time-keeping in the context of confidential computing can be troubling in multiple ways. First, agreeing on time in a distributed context often relies on external trusted time servers, together with an approximation of the time delay between server and recipient. However, enclaves that need time information have to to communicate with remote servers through an untrusted, attacker controlled environment. This potentially allows the adversary to unpredictably delay network packets.Second, even if no trusted time server is utilized and time is only communicated between local peers, adversaries can often control the execution and interruption of enclaves, which may give the attacker some control to induce further delays.

In this paper, we investigate the challenge of reasoning about time and providing temporal guarantees in the context of confidential computing. Particularly, we identify five levels of increasingly more capable time tracking approaches and

SysTEX '23, May 8, 2023, Rome, Italy

ACM ISBN 979-8-4007-0087-3/23/05…$15.00
https://doi.org/10.1145/3578359.3593038

overview existing timing capabilities in popular TEEs, with an explicit focus on the widespread Intel SGX architecture and software ecosystem. Our specific contributions are:

- We identify five distinct levels of tracking the passage of time from within isolated enclaves.
- Focusing on Intel SGX, we analyze which time levels can be provided to enclaves and how this is used by the software ecosystem and in selected academic work.
- We classify other TEE architectures and analyze their limitations in regards to trusted time-keeping.
- Lastly, we overview use cases and applications, highlighting the overall difficulty of guaranteeing trusted time in untrusted environments.

2 System Model

We consider a general TEE architecture, where a security-critical application executes in a trusted enclave protection domain that is isolated from any other software on the system, including the privileged operating system and hypervisor. The enclave may further include a shielding system that offers trusted services to the application, e.g., in form of a library OS or software development kit (SDK).

This study considers applications that need some notion of "wall-clock time" from a trusted time source in order to realize their security objectives. The time source could be local, e.g., a secure timestamp counter in the CPU, or remote, e.g., a trusted server. We explicitly focus on reasoning about the wall-clock time that passes between two execution points, which does not necessarily correspond to the actual running time of the enclave. That is, while the latter may be easier to measure, e.g., by counting the number of instructions executed in the enclave, it is decidedly not a good proxy for wall-clock time in the presence of attacker-induced interrupts that may arbitrarily delay enclave execution.

Note that in case the application needs *absolute* wall-clock time, i.e., calendar time, an extra step may be needed such as an initial time synchronization with a trusted source.

3 Notions of Time

This section identifies five successive levels of time tracking approaches for confidential applications, summarized in Table 1. These levels are additive, so that each level represents an increasingly more capable and accurate time source. However, higher levels may also place more demands on the underlying TEE architecture, and, as discussed later, lower levels may be sufficient for certain applications.

Figure 1 depicts each time level, together with a sample time trace reported to and experienced by the enclave. The top row represents the ground truth, where no attacker is present and wall-clock time advances linearly and at a constant rate. The running time of the enclave application is visualized in the colored squares, whereas the clock symbols represent a query to the abstract time source. The example value returned by the time source is indicated above the clock symbol and will be used in the subsequent enclave computation, colored blue if the time is correct w.r.t. the wall-clock time, or orange if time is influenced by the attacker.

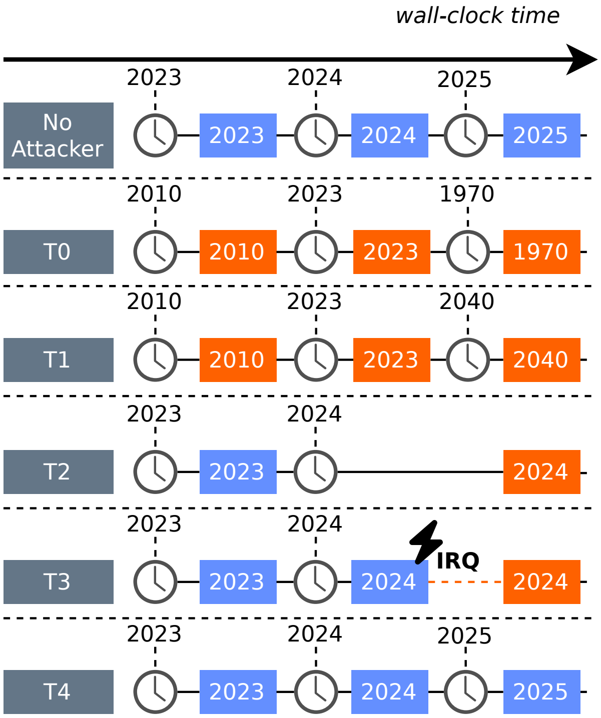

Figure 1. Visual representation of different time levels with attacker influences marked in orange. T_0 is controlled by the attacker, T_1 can only move forward but make jumps, T_2 can be arbitrarily delayed, and finally in T_3 the attacker can still interrupt the enclave before it utilizes the time. Instead, level T_4 gives the same guarantees as when no attacker is present.

***T_0*: No trusted time.** The default case is to have no trusted time, i.e., to fully rely on untrusted input without performing any checks on the incoming time data. As illustrated in Fig. 1, the observed time can be any point in the past or in the future and may vary arbitrarily. While this may suffice for testing scenarios, e.g., annotating untrusted timestamps in a debug log, it is clear that any application-specific, security-sensitive use of time cannot rely on a T_0 timer.

***T_1*: Monotonically increasing time.** If enclaves only have access to an untrusted time source, they can perform certain minimal checks on received timestamps before acceptance. The most important check to perform here is to ensure that received time data only ever increases, i.e., that time never winds backwards. Depending on the granularity of the untrusted time source and the sampling frequency, additional checks may be added, e.g., checking that two time queries should never return the same value or zero. While these checks are minimal and the underlying time source is still untrusted, the counter value can be stored inside the

Table 1. Levels of time and the guarantees given by them. Checkmarks identify that a level protects against this type of attack.

Type	Added guarantee	Rollback	Freq.	Delay	Interrupt	Example time source	Discussed use case
T_0	None					Untrusted OS	—
T_1	Time monotonically advances	✓				Untrusted OS + check	Time-based policies
T_2	Time moves at constant pace	✓	✓			ME, timer thread, remote server	Rate limiting
T_3	Time is read with known delay	✓	✓	✓		Secure TSC, MMIO timer	Resource counting
T_4	Use of time is atomic	✓	✓	✓	✓	Trusted scheduler	Credential expiration, DRM

enclave and T_1 can at least prevent attackers from rewinding the internal time of an enclave. Since T_1 timers cannot rely on external data, they have to store the time locally. This makes them prone to rollback attacks where time is reverted by restarting the enclave with an obsolete local timestamp.

Fig. 1 shows that attackers in T_1 can advance time at will and can trigger arbitrarily large time jumps for the enclave.

T_2: Trusted remote time. To improve time-keeping, the enclave can switch from an untrusted to a remote trusted time source, guaranteed to have a fixed, known frequency. As such, this time source can be relied upon for trusted time information, when standard cryptographic measures are in place to protect and validate the integrity and authenticity of the time packets over the untrusted network (e.g., this secure time channel can be configured as part of the attestation protocol). Note that the "remote" trusted time source can be truly remote, e.g., a server reached over the Internet, but could also be provided on the same system-on-chip, e.g., by a trusted co-processor. If the monotonicity of the external time source can be trusted, rollback attacks targeting the time value are also not possible anymore since on enclave restarts, the time can be re-queried. Obviously, rollback attacks are still a threat to enclaves with a T_2 timer and above, but they will not affect the time stored in the enclave.

As the communication channel between the time source and the enclave is still under the attacker's control in regards to availability, T_2 time requests and responses may be arbitrarily delayed. Hence, T_2 timestamps received by the enclave can only ever be trusted as a *lower bound* on the reference wall-clock time. Figure 1 depicts a potential situation where the attacker would allow the enclave to query the time source, but then delay the response.

T_3: Trusted local time. In order to prevent arbitrary delay attacks, enclaves must be able to access the trusted time source through a direct channel without adversary intervention, e.g., when the time source is an on-chip timestamp counter that can be queried via dedicated CPU instructions. If both time source and channel are trusted, adversaries cannot arbitrarily delay timestamps anymore before they reach the enclave. This, technically, grants enclaves access to accurate time information with both a lower and an *upper bound.*

We argue, however, that in order to utilize such accurate time information in practice, enclaves also need to be able to act *atomically*, i.e., without (undetected) attacker-induced interruptions. Figure 1 depicts a possible attack that is still possible on T_3 where the adversary precisely interrupts the enclave right after it reads from the trusted time source. The adversary can then keep the enclave interrupted until it suits them and only resume the enclave at a convenient time, effectively reducing the accurate temporal guarantees back to only the lower bound offered by a T_2 time source.

Consider an enclave that only grants access to a resource if a presented certificate is valid, and denies access otherwise. If the adversary begins the validation process at a time that the certificate is still valid, the enclave will read a permissible time from the time source. Right after the time value has been read into the enclave, the adversary could however interrupt the enclave. When the enclave is later resumed, the adversary could gain access to the resource even if the initially presented certificate is not valid anymore.

T_4: Trusted and atomic local time. The dangers of the above attack vector are nuanced. Many interruption attacks on T_3 timers only work if the adversary initially had access to a resource.Thus, arguably, maintaining this access until after the credentials expired can be considered as a parallel issue to denial of service attacks where output of the enclave is delayed by the adversary. We note, however, that for some applications and use cases this attack vector may not be acceptable and enclaves should only present an output at the correct time. We discuss two examples of this in Section 6.4.

4 Intel SGX

The Intel SGX architecture does, in itself, not provide any notion of trusted time. However, several proposals exist to provide trusted time services to SGX enclaves.

Platform service. Intel provides platform software (PSW) to set up and use trusted time and monotonic counters [6]. Particularly, the PSW offers a T_2 time source based on a coarse-grained clock in the Intel management engine (ME), a chip that physically resides on the same package but is entirely separate from the main CPU. This allows timer-based policy enforcements on offline platforms that are not necessarily connect to a network. All communication between enclaves and the ME passes via the untrusted operating system and is, hence, cryptographically protected, but can be arbitrarily delayed. Enclaves can use the `sgx_get_trusted_time`

API to track the amount of time (in seconds) passed since a previous read of the timer. ME services are excluded in the Intel SGX Linux PSW from version 2.8 from 2020 onwards.

Timestamp counter. Initial SGX versions did not allow to execute the x86 `rdtsc` instruction to receive the processor's internal timestamp counter (TSC) in enclaves. However, more recent SGX2 processors now support the `rdtsc` instruction in enclave mode, with an explicit warning that the instruction may be trapped and the timestamp returned may be influenced by the untrusted operating system.

We experimentally confirmed that privileged adversaries can arbitrarily control the values returned by `rdtsc` on SGX2 platforms by directly manipulating the underlying `IA32_TIME_STAMP_COUNTER` model-specific register (MSR). For this, we developed a sample enclave that executes two `rdtsc` instructions and returns the measured cycle difference. One would expect this difference to always be positive and reflect the elapsed time in between the two `rdtsc` instructions. However, in our proof-of-concept, we make the observed time in the enclave move "backwards" by interrupting the enclave and overwriting the TSC value using the privileged `wrmsr` instruction in between both `rdtsc` calls. Thus, we conclude that `rdtsc` on current SGX platforms is not any better than an adversary-controlled T_0 timer.

We note, however, that the TSC MSR is duplicated per logical processor and can, hence, only be changed by interrupting the enclave. That is, as long as the enclave is not interrupted, any successive `rdtsc` reads will behave as a trusted T_4 timer that monotonically increases at fixed frequency. This may be leveraged with upcoming ISA changes [8] that will make SGX enclaves interrupt-aware. At the same time, the restriction that an enclave is never interrupted may be too limiting for some real-world deployments.

Academic timers. TrustedClock [14] is a system that relies on an interrupt handler in system management mode (SMM) to implement functionality that is similar in its security guarantees to `sgx_get_trusted_time` by the Intel SGX PSW. The authors base their security argument on the handler being loaded at boot time with a secret key that is not accessible at runtime by the untrusted OS. Enclaves can then communicate with the handler by establishing a secure channel based on the pre-shared public key. Aside from the security implications of including the unattested SMM handler in the trusted computing base, TrustedClock is stated to not be completely immune to timer modifications by the untrusted OS. Instead, TrustedClock relies on combining several x86 hardware timers and asserting their monotonicity (similar to T_1), thus making it harder for a malicious OS to precisely tamper with all involved timers at the same time.

TimeSeal [3] utilizes a cohort of multiple threads that each periodically poll `sgx_get_trusted_time` to monotonically increase the time. The authors argue that by using multiple threads and nuanced synchronization policies between the threads, attacks can be limited in their scope. Nonetheless, attackers that delay or prevent scheduling of timer threads at convenient times, such as right before a thread returns a time response, will be able to skew the reported time of TimeSeal. Additionally, with the discontinuation of `sgx_get_trusted_time` on Linux systems, TimeSeal may not be implementable anymore as of today.

Table 2. Intel SGX software projects and their use of time. Stars indicate that we prototyped a proof of concept to roll back time for a project. Intel SDK is shown for Linux/Win.

SDK	OE	EDP	Gramine	LKL	Occlum	Mystikos	Ego	Enarx
$-/T_2$	T_0*	T_0	T_1	T_1	T_0*	T_0	T_1	T_0

Software ecosystem. Several shielding systems offer a "lift-and-shift" approach to port legacy applications to SGX. However, these applications were not designed for SGX's threat model and may rely on sane OS time services.

Table 2 summarizes how existing SGX shielding systems expose the untrusted OS time to applications. Interestingly, we found that most projects include a warning, but forward OS time without any restrictions (T_0), whereas the Gramine and LKL library operating systems at least ensure that time does not move backwards (T_1). We developed two minimal proof-of-concepts to demonstrate backwards time advances in OpenEnclave and Occlum. We, hence, recommend that all projects include minimal checks to adopt the T_1 time model.

5 Other Architectures

Intel TDX. The TDX specification [12] discusses TSC virtualization, which provides a consistent virtual TSC value to a trusted domain (TD) among all its vCPUs. The virtual TSC starts counting from zero when the TD is initialized and runs at a fixed frequency defined by a parameter stored in the TD configuration. Guest TDs can access the virtual TSC using the `rdtsc` instructions, and its value is calculated from the physical TSC adjusted to the TD's offset and frequency.

We analyzed the source code of the TDX module, which makes sure that the TSC is not modified before a TD enter by comparing the current `TSC_ADJUST` MSR value against a reference value stored on TD creation, raising an exception if a mismatch between the two values occurs. This allows TDs to obtain a stable and consistent TSC value across TD lifetime, guaranteeing a time level T_3.

AMD SEV. The AMD SEV-SNP [2] extensions introduced a so-called *Secure TSC* feature, which should provide enhanced protection against TSC manipulations. The Secure TSC feature can be enabled by a guest VM at boot and prevents the hypervisor from intercepting any `rdtsc` instructions called by the guest VM. Furthermore, the calculation of the TSC value is made using offset and frequency parameters stored in VM secure memory, thus not accessible by a malicious hypervisor. However, it is unclear from the specification whether the Secure TSC feature prevents the hypervisor from directly writing into the TSC via the `wrmsr` instruction, similar to our proof-of-concept on SGX2. In such case, the hypervisor could effectively tamper with the TSC while the guest VM is not running, preventing it from leveraging a stable and consistent representation of elapsed time. Without such guarantee, the TSC counter would only be a T_0 time level for SEV-SNP guest VMs.

ARM TrustZone. ARM TrustZone [15] separates progam execution into two protection domains, the secure world and the normal world. Both worlds are hardware-isolated, run their own OS, and normal-world software is prevented from directly accessing secure world resources. TrustZone also allows for system devices, such as a time source, to be restricted to one world. This, however, is implementation specific, with the TrustZone protection controller being an optional component. TrustZone further extends the processor's interrupt controller with prioritized secure and non-secure interrupt sources to prevent denial-of-service attacks from the normal world. Based on these features, ARM TrustZone supports up to T_4, and has been previously leveraged to implement secure real-time systems [18] or TPM functionality in software [16].

ARM CCA. ARM confidential compute architecture [4] adds additional execution worlds to the ARM architecture that allow to run multiple so-called Realms in parallel to the ARM TrustZone secure world. All Realms are all managed by a trusted realm management monitor (RMM) component [5]. While each Realm has direct access to architectural timers, the RMM does not make scheduling decisions or manage interrupts, both controlled by the untrusted hypervisor [4]. As such, ARM CCA has access to a T_3 time source.

TPM. A trusted platform module (TPM) is a hardware-based security component that provides a secure foundation for system integrity usually in form of a co-processor. TPMs comprise timing components and monotonic counters. According to the specification [17], a TPM includes two primary timing components referred to as *Time* and *Clock*. The *Time* component is separate hardware that provides the number of milliseconds since it has been powered on and cannot be controlled by software. The *Clock* component yields a value that software can advance but can never roll back.

Thus, by mapping this specification to our time levels, the *Clock* component only guarantees a T_1 time, since a privileged adversary can advance the *Clock* value arbitrarily, while *Time* may reach up to the T_4 level if an offset is securely provided to the TPM to align its value to the wall-clock time. However, according to [17], attackers may change the TPM clock frequency by at most ±32.5%, which should be taken into account when checking the time.

6 Use Cases

This section focuses on use cases that rely on some form of time. To facilitate this discussion, we adopt an incremental approach, beginning with time-based policies that only necessitates level T_1, and gradually progressing through each level up to credential expiration, which requires T_4.

6.1 Time-Based Policies

A simple time-based policy is that a user that once lost access to a resource will not regain access to it. By utilizing T_1, enclaves can enforce this behavior so that the attacker cannot revert time to regain access. Enclaves can be useful in systems where the OS is initially in a trusted state but may become compromised during runtime. In this case, the enclave would periodically be invoked with linearly increasing timestamps and then at some time be invoked with an old timestamp. If the attacker timestamp precedes the last timestamp sent by the benign operating system, the enclave would be able to detect this attack and prevent access.

It is important to note that, by using T_1, the enclave will not be able to revoke access if the OS does not wish this to happen, as progress of time is still attacker-controlled. The only guarantee is that, once revoked, access can not be reinstated. This is similar to *Clock* guarantees of TPMs [17]. Similarly, using T_1 timers assumes that rollback attacks on the enclave state can be detected by the enclave. Otherwise, the adversary could restart the enclave and provide a stale timestamp or stale sealed data to the restarted enclave to get the desired result. If rollback attacks on the time value can not properly be prevented, a T_2 timer has to be used.

6.2 Rate Limiting and TOTP

Rate Limiting. Time-based rate limiting is a mechanism often used to narrow the impact of brute force attacks. Consider a service running in an enclave that performs password checking [13]. If time is OS-controlled, time-based rate limiting would not be effective as attackers can artificially advance time. The service would easily be tricked into processing more requests than desired. Thus, T_1 is not a sufficient time level for rate limiting. Only when the frequency of the clock is independent from the attacker can the enclave rely that at least a minimum amount of time has passed.

This use case does not have higher requirements on the time source than a T_2 time source. It does not matter whether a minute or a year has passed between two requests as long as *at least* a minimum amount of time has passed. If the untrusted OS increases the time between requests, this does not go against the security guarantees offered by rate limiting and would only impact availability of the service.

TOTP. Time-based one-time passwords (TOTPs) [10] are commonly used in second-factor authentication systems. The algorithm takes as input a fixed parameter k (e.g., a key) and a variable parameter x, and gives as output the one-time password, which is simply computed as a HMAC truncated to the desired size. In TOTP, the variable parameter x depends on the current time t, an initial value t_0, and a fixed time window W according to the equation $x = \lfloor (t - t_0)/W \rfloor$. Typically, x_0 is equal to 0 and W is equal to 30 seconds. This means that, within the same 30-second window, the resulting OTP will always be the same, provided that the same k is given as input. As such, attackers who have control over the time source, as in levels T_0 and T_1, could potentially slow down or even freeze the time perceived by the TOTP server, preventing the OTP from changing. This could be exploited by attackers to circumvent the second factor authentication provided by TOTP, either through brute-force attacks or by reusing an old OTP previously used by the legitimate user. Thus, we conclude that TOTP requires a T_2 time source.

In the context of Intel SGX, a few projects have proposed the use of TOTP authentication. For instance, SGX-UAM [19] utilizes TOTP to authenticate a client to the identity provider, while SCONE [7] employs it as a second-factor authentication for their configuration and attestation server. While the former utilizes `sgx_get_trusted_time`, now discontinued on Linux, it is unclear which time source is used by the latter.

6.3 Resource Accounting

When utilizing confidential computing in cloud environments, both users and cloud providers may require a trustworthy and accountable measurement of spent resources during a computation. Interesting nuances with this use case occur when code running inside the enclave is not necessarily trusted by both parties that wish to rely on the measurement. Take for example the case of Function-as-a-Service in enclaves [1]. There, workload providers run a workload and only wish pay for the resources used. If possible, workload providers may want to pay less than what they owe. Cloud providers, on the other hand, may wish to overestimate the resources consumed and demand more compensation.

The cloud provider can always make accurate estimates on enclave time, since the workload runs on their infrastructure. However, the workload provider cannot trust any time levels below T_3: If the untrusted OS controls the time, such as in T_0 or T_1, time can be manipulated to overestimate resource consumption, leading to higher billing for the workload provider. Furthermore, even with a T_2 clock the cloud provider could arbitrarily delay the enclave communication with the clock and artificially increase the workload time. Only if the channel to the clock is independent from the cloud provider, and the clock is independent from influence, the time source can be accurately utilized for resource accounting. Note that, without a full T_4 utilization of the clock, the cloud provider could still cheat and interrupt the workload for periods of time. In this case, the system would need to be set up in a way that punishes such behavior and undercounts time spent potentially interrupted, e.g., through means of resetting the count on interrupt reentries.

6.4 Credential Expiration and DRM

A common use cases for using time is to check the validity of credentials, such as PKI certificates and JSON web tokens (JWTs). Typically, these credentials have a validity window defined by a *Not Before* and a *Not After* field. Thus, it is important that the time source provides a reliable time to correctly validate such credentials, e.g., during the TLS-handshake or the authorization step at application level.

If time is not reliable inside the enclave, it could lead to security breaches such as allowing access to sensitive data to unauthorized users. If an untrusted T_0 or T_1 clock is used, attackers may rewind the time perceived by the enclave to fall within the validit window of an expired certificate. If instead stronger T_2 or T_3 timers are used, the adversary may still induce network delays and/or interrupts to coerce the enclave into accepting credentials that were valid at the beginning of the communication, but have expired before their actual use. This problem is exacerbated when credentials are short-lived: typically, access tokens are refreshed frequently and thus have a rather short lifetime of minutes to hours [11]. Thus, the enclave should only trust a time level T_4 in order to make correct decisions.

The open-source `rust-mbedtls` library written by Fortanix [9] allows TLS-termination inside SGX enclaves. Time-validity of certificates is only checked if the `time` feature is enabled, which means that either it is not checked at all (if disabled), or it is checked against the insecure OS time (if enabled). This shows that the absence of a trusted time source in SGX can potentially compromise the security of applications, thus software engineers should take into account such limitations when developing their applications.

The use case for DRM can be reduced to an instance of credential expiration, where a user wants to access a resource that should already be locked to them. If the enclave is interrupted after the authorization check but before the use of the resource, the latter might take place when the user has no longer permission to access it.

7 Conclusions

Tracking the passage of time is nuanced when relying solely on measurements that are partially controlled by an adversary. To better reason about the usage of time inside TEEs, we identified five progressive levels of time-keeping, T_0 to T_4 and classified the popular Intel SGX as well as other commonly used TEEs. Considering relevant use cases such as rate limiting or resource counting, we show that not every application requires access to the highest level of time, for which existing solutions may already suffice. However, this classification also shows that other use cases, such as credential expiration, need stronger time guarantees that most architectures cannot yet provide. Thus, reliable time-keeping is highly application-specific and software engineers should take into account the limitations of current technologies.

Acknowledgments

This research is partially funded by the Research Fund KU Leuven, by the Flemish Research Programme Cybersecurity, and by the CyberExcellence programme of the Walloon Region, Belgium. This research has received funding under EU H2020 MSCA-ITN action 5GhOSTS, grant agreement no. 814035. Fritz Alder and Jo Van Bulck are supported by a grant of the Research Foundation – Flanders (FWO).

References

[1] Fritz Alder, N Asokan, Arseny Kurnikov, Andrew Paverd, and Michael Steiner. 2019. S-faas: Trustworthy and accountable function-as-a-service using intel SGX. In *Cloud Computing Security Workshop*.

[2] AMD64 Technology. 2023. *AMD64 Architecture Programmer's Manual Volume 2: System Programming*. Specification.

[3] Fatima M Anwar, Luis Garcia, Xi Han, and Mani Srivastava. 2019. Securing time in untrusted operating systems with timeseal. In *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 80–92.

[4] ARM. 2021. *ARM CCA Security Model 1.0*. Security Model (SM).

[5] ARM. 2023. *ARM Realm Management Monitor specification*. Specification.

[6] Shanwei Cen and Bo Zhang. 2017. Trusted time and monotonic counters with intel software guard extensions platform services. (2017).

[7] SCONE Confidential Computing. 2023. 2FA with Time-based One-time Passwords. https://sconedocs.github.io/CAS_session_lang_0_3. (2023).

[8] Intel Corporation. 2022. *Asynchronous Enclave Exit Notify and the EDECCSSA User Leaf Function*. White Paper.

[9] Fortanix. 2023. Idiomatic Rust wrapper for MbedTLS. https://github.com/fortanix/rust-mbedtls. (2023).

[10] Internet Engineering Task Force (IETF). 2011. *TOTP: Time-Based One-Time Password Algorithm*. RFC- Proposed Standard 6238.

[11] Internet Engineering Task Force (IETF). 2013. *OAuth 2.0 Threat Model and Security Considerations*. RFC- Informational 6819.

[12] Intel. 2022. *Architecture Specification: Intel® Trust Domain Extensions (Intel® TDX) Module*. Specification.

[13] Klaudia Krawiecka, Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, and N Asokan. 2018. Safekeeper: Protecting web passwords using trusted execution environments. In *WWW Conference*.

[14] Hongliang Liang and Mingyu Li. 2018. Bring the Missing Jigsaw Back: TrustedClock for SGX Enclaves *(EuroSec'18)*.

[15] Sandro Pinto and Nuno Santos. 2019. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Computing Surveys (CSUR)* 51 (2019).

[16] Himanshu Raj, Stefan Saroiu, Alec Wolman, Ronald Aigner, Jeremiah Cox, Paul England, Chris Fenner, Kinshuman Kinshumann, Jork Löser, Dennis Mattoon, Magnus Nyström, David Robinson, Rob Spiger, Stefan Thom, and David Wooten. 2016. fTPM: A Software-Only Implementation of a TPM Chip. In *USENIX Security Symposium*.

[17] Trusted Computing Group. 2019. *Trusted Platform Module Library, Part 1: Architecture*. Specification TPM 2.0 Library.

[18] Jinwen Wang, Ao Li, Haoran Li, Chenyang Lu, and Ning Zhang. 2022. RT-TEE: Real-time System Availability for Cyber-physical Systems using ARM TrustZone. In *2022 IEEE Symposium on Security and Privacy*.

[19] Liangshun Wu, HJ Cai, and Han Li. 2021. SGX-UAM: A secure unified access management scheme with one time passwords via intel sgx. *IEEE Access* 9 (2021), 38029–38042.