ProximiTEE: Hardened SGX Attestation by Proximity Verification

Aritra Dhar
ETH Zurich

Ivan Puddu
ETH Zurich

Kari Kostiainen
ETH Zurich

Srdjan Čapkun
ETH Zurich

ABSTRACT

Intel SGX enables protected enclaves on untrusted computing platforms. An important part of SGX is its remote attestation mechanism that allows a remote verifier to check that the expected enclave was correctly initialized before provisioning secrets to it. However, SGX attestation is vulnerable to relay attacks where the attacker, using malicious software on the target platform, redirects the attestation and therefore the provisioning of confidential data to a platform that he physically controls. Although relay attacks have been known for a long time, their consequences have not been carefully examined. In this paper, we analyze relay attacks and show that redirection increases the adversary's abilities to compromise the enclave in several ways, enabling for instance physical and digital side-channel attacks that would not be otherwise possible.

We propose ProximiTEE, a novel solution to prevent relay attacks. Our solution is based on a trusted embedded device that is attached to the target platform. Our device verifies the proximity of the attested enclave, thus allowing attestation to the intended enclave regardless of malicious software, such as a compromised OS, on the target platform. The device also performs periodic proximity verification which enables secure enclave revocation by detaching the device. Although proximity verification has been proposed as a defense against relay attacks before, this paper is the first to experimentally demonstrate that it can be secure and reliable for TEEs like SGX. Additionally, we consider a stronger adversary that has obtained leaked SGX attestation keys and emulates an enclave on the target platform. To address such emulation attacks, we propose a second solution where the target platform is securely initialized by booting it from the attached embedded device.

1 INTRODUCTION

Trusted execution environments (TEEs) like Intel's SGX enable to securely execute applications on untrusted computing platforms. Remote attestation is a key feature of SGX, and other similar TEE architectures, as it allows a remote verifier to check that the attested enclave was correctly constructed before provisioning secrets to it.

Relay attacks. While SGX's remote attestation guarantees that the attested enclave runs the expected code, it *does not*, however, guarantee that the enclave runs on the expected computing platform. An adversary that controls the OS (or other software) on the target platform can relay incoming attestation requests to another platform. Such relay attacks are a long-standing open problem in trusted computing, as already a decade ago Parno identified such attacks in the context of TPM attestation [25].

Upon a first look, it might seem that relay attacks do not pose a problem for TEEs. If the attacker relays the attestation to another machine, the same security guarantees should hold since the data will only be available within the remote TEE and the enclave code that can access the provisioned secrets is verified. However, such simple reasoning is incorrect.

In this paper, we provide the first careful analysis of the *implications* of relay attacks on SGX and show that by relaying, the adversary increases his capabilities to attack the attested enclave significantly. One example of increased adversarial capabilities is physical side-channel attacks. If the adversary redirects the attestation to a platform that he physically controls, he can mount various physical side-channel attacks, like [12, 13, 31, 33], that would not have been possible without the relay. Another example are enhanced side-channel attacks. While controlling the OS is in the SGX attacker model, it is not unrealistic that an adversary might be in the situation of controlling only user-privileged code on the target platforms. This degree of control however allows him to redirect attestation to another platform where he controls the OS, which allows him to launch software-based side-channel attacks, such as [7, 14, 24], that leverage system privileges to attack enclaves. In Section 3, we explain further examples of attacks that are enabled by attestation redirection.

A typical "solution" to relay attacks is to assume *trust on first use* (TOFU). However, in many application scenarios, TOFU is neither secure nor practical. For example, solutions, where attestation is performed immediately after a fresh OS installation, cannot be applied to settings where OS reinstallation is simply not possible. Besides, all TOFU variants assume that the target platform OS is trusted, even if momentarily, which violates the SGX's trust model.

The SGX attestation protocol is designed to be anonymous. The protocol is based on EPID group signatures [15] and thus the remote verifier cannot distinguish whether the correct enclave on the target platform was attested or if the attestation was redirected to another platform. Upon first inspection, it may seem like relay attacks are only possible because of such anonymity features and that relaying could be easily prevented if attestation protocols were designed to be non-anonymous. However, such simple reasoning is incorrect as well. We show that all SGX attestation variants, including the "linkable" attestation mode and the recently introduced Data Center Attestation Primitives (DCAP) [28] are vulnerable to relay attacks. We also explain why relay attacks would remain possible, even if all anonymity features would be removed from the attestation.

Our solution. We propose a new solution, called ProximiTEE, that prevents relay attacks by leveraging a simple embedded device that is attached to the attested target platform. Our solution is best suited to scenarios where i) the deployment cost of such an embedded device is minor compared to the benefit of more secure attestation, and ii) TOFU solutions are not acceptable. Attestation of servers at cloud computing platforms and setup of SGX-based permissioned blockchains are two such examples.

In ProximiTEE, the remote verifier establishes a secure connection to the embedded device whose public key it knows through

standard device certification. The device performs normal SGX attestation and additionally *verifies the proximity* of the attested enclave using a simple distance-bounding protocol [6]. After the initial attestation, the device performs *periodic* distance-bounding measurements and the communication channel created during the attestation stays active only as long as the device is connected to the same platform. Thus, the physical act of attaching the device to an SGX platform enables secure attestation (enrollment) while detaching the device will prevent further communication with the attested enclave (revocation). Neither enrollment nor revocation requires interaction with a trusted authority. This property is useful in applications like permissioned blockchains where validator nodes are separate organizations assigned by a trusted authority. The authority can issue one device per organization, and each organization is free to manage their computing resources (e.g., detach the device from one platform and attach it to another) without interaction with the authority.

Main results. Parno [25] identified distance bounding as a candidate solution to TPM relay attacks already ten years ago, but concluded that it could not be realized securely as the slow TPM identification operations (signatures) make a local and relayed attestation indistinguishable. Our evaluation shows that proximity verification *is* possible for SGX assuming very fast adversaries. The main reason why distance bounding protocols work for SGX, but not with TPMs, is that SGX is a programmable TEE where it is possible to use pre-established security associations and efficient challenge-response protocols based on simple operations such as XOR.

To evaluate ProximiTEE, we implemented it using a USB 3.0 prototyping board. The main purpose of our evaluation is to demonstrate that the adversary cannot redirect the attestation *over the Internet* to an adversary-controlled platform without being detected. We focus on such re-direction, as it offers the most increased capabilities to the adversary (e.g., physical attacks). The secondary purpose of our evaluation is to determine whether proximity verification can prevent redirection to a co-located platform, like another server on the same server rack. Such relays are typically less harmful, but ideally, they should be prevented as well.

In our evaluation, we *simulate* a strong adversary that i) is only a single network hop away from the target, ii) performs the required protocol computations instantaneously, iii) has infinitely fast hardware interface, and iv) has enabled software-based packet forwarding optimizations on the target platform. We measure the legitimate challenge-response latency on our prototype to be $185\mu s$ on average. In the case of the simulated relay attack, the average latency is about $264\mu s$. These two latency distributions are distinguishable and allow us to set our proximity verification protocol parameters such that the adversary's probability of performing a successful relay attack is negligible (3.55×10^{-34}), while legitimate verification succeeds with a very high probability (0.999999977). Importantly, the adversary cannot increase his success probability with repeated attempts, as attestation is triggered by the trusted remote verifier. Our experiments also show that enclave revocation using periodic proximity verification is both secure and practical.

The performance overhead of proximity verification is small: the initial proximity verification adds only a small delay to the attestation protocol, and the periodic proximity verification consumes only a very minor fraction of the available USB 3.0 channel capacity. Our implementation shows that the complexity of such a device can be small: the software TCB of our prototype is 3.8 KLoC.

Emulation attacks. Additionally, we consider a stronger adversary that has obtained leaked, but not yet revoked, attestation keys and can *emulate* an SGX-enabled processor. Proximity verification alone cannot prevent emulation attacks, as a perfectly emulated enclave would pass any proximity test. Therefore, we propose a second attestation mechanism based on *boot-time initialization*.

In this solution, the target platform loads a small, single-purpose kernel from the attached device and launches an enclave that seals a secret key known by the device. Subsequently, when attestation is needed, the enclave can verify the proximity of other enclaves on the same platform using SGX's local attestation. This enables secure attestation regardless of potentially leaked attestation keys. Our second solution can be seen as a novel variant of the well-known TOFU principle. The main benefits over previous variants are easier adoption (e.g., no OS re-installation) and increased security (e.g., OS not trusted even temporarily). Due to space constraints, we provide the full details of this second solution in Appendix A.

Contributions and outline. This paper contributions are organized as follows:

1. *Analysis of relay attacks.* While relay attacks have been known for more than a decade, their implications have not been fully analyzed. In Section 3, we provide the first such analysis and show how relaying amplifies the adversary's capabilities for attacking SGX enclaves.
2. *ProximiTEE: Addressing relay attacks.* In Section 4, we propose a hardened SGX attestation mechanism based on an embedded device and proximity verification to prevent relay attacks. ProximiTEE does not rely on the common TOFU assumption, and hence, our solution improves the security of previous attestation approaches.
3. *Experimental evaluation.* We implement a complete prototype of ProximiTEE and evaluate it against a very strong and fast adversary. Our evaluation in Section 5 is the first to show that proximity verification can be both secure and reliable for TEEs like SGX.
4. *Addressing emulation attacks.* We also propose another attestation mechanism based on boot-time initialization to prevent emulation attacks. This mechanism, described in Appendix A, is a novel variant of TOFU with deployment, security and revocation benefits.

2 SGX BACKGROUND

Intel SGX is a TEE architecture that isolates application enclaves from all other software running on the system, including the privileged OS [9]. Enclave's data is encrypted and integrity protected whenever it is moved outside the CPU chip. The untrusted OS is responsible for the enclave creation and its initialization actions are recorded securely inside the CPU, creating a *measurement* that captures the enclave's code. Enclaves can perform local attestation, which allows one enclave to ask the CPU to generate a signed report that includes its measurement. Another enclave on the same platform can verify the validity of the report without interacting with any other external services. Enclaves can *seal* data to disk, which allows them to securely store confidential data such that only the same enclave running in the same CPU will be able to retrieve it later.

2.1 Remote Attestation

Remote attestation enables an external verifier to check whether a specific enclave has been correctly instantiated in a SGX protected environment. In the following, we describe the two main classes of remote attestation supported by Intel: i) "enhanced privacy ID" (EPID) attestation [15], and ii) the recently introduced "data center attestation primitives" (DCAP) [28].

EPID attestation. The EPID remote attestation is an interactive protocol between three parties: the remote verifier; the attested SGX platform; and the Intel Attestation Service (IAS), an online service operated by Intel. Each SGX platform includes a system service called *Quoting Enclave* (QE) that has exclusive access to an attestation key. The remote verifier sends a random challenge to the attested platform, which replies with a QUOTE structure, capturing the enclave's measurement from its creation, signed with the attestation key. The verifier can then send the QUOTE to the IAS that verifies its signature and correctness, checks that the attestation key has not been revoked, and in case of successful attestation signs the QUOTE.

The attestation key used by the QE is part of a group signature scheme called EPID that supports two signature modes: random base mode and name base mode, also called "linkable" mode. Both signature modes do not uniquely identify the processor to the IAS; but only a group, like a particular processor manufacturing batch. The difference between them is that the linkable signature mode allows to check whether two attestation requests came from the same CPU.

DCAP attestation. Whereas the EPID attestation variant requires connectivity to an Intel-operated attestation service, and is limited to pre-defined signature algorithms, the main goal of the DCAP attestation variant is to enable corporations to run their own local attestation services with freely chosen signature types. To achieve this, each SGX platforms is, at the time of manufacturing, equipped with a unique *Platform Provisioning ID* (PPID) and *Provisioning Certification Key* (PCK). Intel also provides a trusted *Provisioning Certification Enclave* (PCE) that acts as a local CA and certifies custom Quoting Enclaves that can use freely-chosen attestation services and signatures.

DCAP attestation requires a trusted enrollment phase, where the enrolled SGX platform sends its PPID (in encrypted format) to a local corporate key management system that obtains a PCK certificate for the enrolled platform from an Intel-operated DCAP service. After that, the custom Quoting Enclave can create a new attestation key that is certified by the PCE enclave on the same platform. The certified attestation key can then be delivered to the corporate key management system that verifies it using the previously obtained PCK certificate. Once such enrollment phase is complete, the custom QE can sign attestation statements that can be verified by a local corporate attestation service without contacting Intel.

2.2 Side-Channel Leakage

Recent research has demonstrated that the SGX architecture is susceptible to side-channel leakage. Secret-dependent data and code access patterns can be observed by monitoring shared physical resources such as CPU caches [7, 14, 24] or the branch prediction unit [17]. The OS can also infer enclave's execution control flow or data accesses by monitoring page fault events [35]. Many such attacks can be addressed by hardening the enclave's code, e.g., using cryptographic implementations where the data or code access patterns are independent of the key.

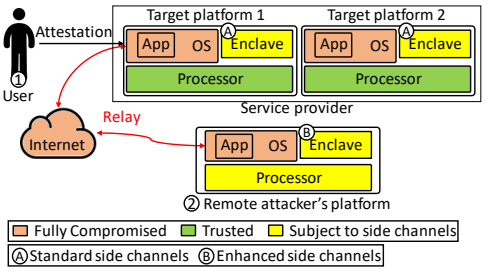

Figure 1: Relay attack. The adversary redirects attestation to his own platform which gives him increased (side-channel and kernel-level) abilities to attack the attested enclave.

The recently discovered system vulnerabilities Spectre [16] and Meltdown [19] allow application-level code to read memory content of privileged processes across separation boundaries by exploiting subtle side-effects of transient execution. The Foreshadow attack [8] demonstrates how to extract SGX attestation keys from processors by leveraging the Meltdown vulnerability.

Microcode updates. During manufacturing, each SGX processor is equipped with hardware keys. When SGX software is installed on the CPU for the first time, the platform runs a provisioning protocol with Intel. In this protocol, the platform uses one of the hardware keys to demonstrates that it is a genuine Intel CPU running a specific microcode version and it then then joins a matching EPID group and obtains an attestation key [15] (or a signing key for the PCE enclave).

Microcode patches issued by Intel can be installed to processors that are affected by known vulnerabilities such as the above mentioned Foreshadow attack. When a new microcode version is installed, the processor repeats the provisioning procedure and joins a new group that corresponds to the updated microcode version and obtains a new attestation key which allows IAS to distinguish attestation signatures that originate from patched processors from attestation signatures made by unpatched processors [15].

3 RELAY ATTACK ANALYSIS

In this section, we provide an analysis of relay attacks on SGX.

3.1 Relay Attacks

We consider a system model shown in Figure 1 that consists of three parties: the target platform, the remote verifier, and the attacker's platform. The remote verifier is a trusted party that wishes to connect and attest to a specific SGX platform. The target platform is the SGX platform to which the remote verifier intends to connect. Finally, the attacker's platform is a platform owned by the attacker that is connected to the target platform through the Internet.

Adversary model. We consider the following adversary model that we call the *relay attacker*. The relay attacker controls the OS and all other privileged software on the *target* platform at least *temporarily*, in particular at the time of the remote attestation. The OS compromise on the target platform may be later detected and disinfected. We consider the case in which the target platform resides in a data

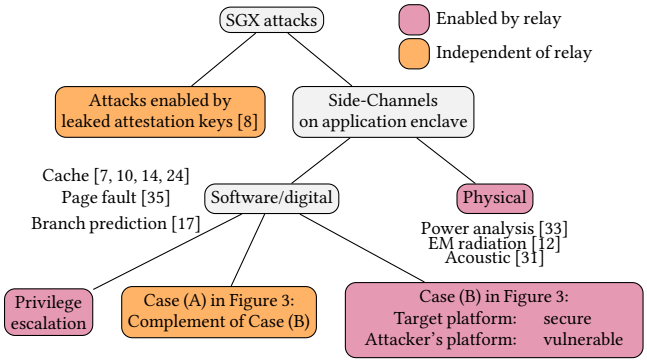

Figure 2: Relay attack implications. The tree shows the types of attacks that are enabled by redirection and ones that are independent of relay.

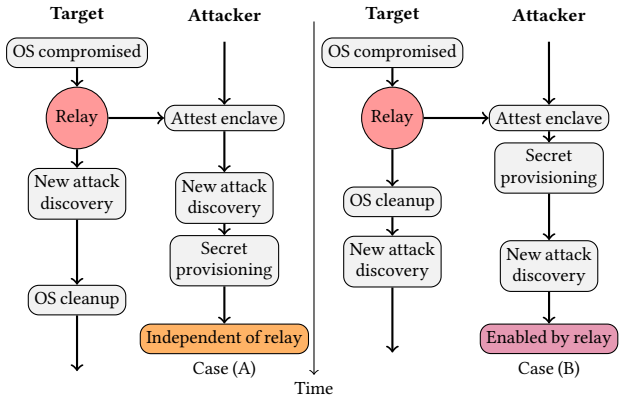

Figure 3: Example sequences of events. In Case A the attack success is independent of relay. In Case B attestation redirection enables the attack.

center or otherwise in a facility with restricted physical access. The attacker hence *does not* have physical access to the target platform (or any other co-located platform in the same facility).

The relay attacker controls the OS and all other privileged software on the attacker's platform *permanently* and has physical access to that platform. The attacker also controls the network between the target platform and his platform. At the time of the attestation, the adversary has not been able to extract attestation or sealing keys from his platform or any other SGX processor.

The relay attack. The relay attacker can redirect the attestation requests intended for the target platform to his platform, as shown in Figure 1. This is a realistic attack for two reasons. First, in the SGX attacker's model the adversary is allowed to control the OS and can hence easily redirect any network request the target platform receives. Second, even if the attacker cannot compromise the OS in the target platform, it might be able to exploit some vulnerability of the untrusted application managing the enclave. The exploit might allow the attacker to manipulate the application's control flow to redirect attestation request to any platform he desires.

3.2 Relay Attack Implications

Although relay attacks have been known for a long time [25], their implications to modern TEEs like SGX have not been carefully analyzed. Next, we perform the first such analysis.

The main consequence of attestation redirection is that it *increases the adversary's ability to attack the attested enclave* through side-channels which are a well-known limitation of SGX (see Section 2.2). In Figure 2 we highlight two major classes of attacks: those that are only possible by first performing a relay attack, which we denote as "enabled by relay", and those that can be done whether or not the attacker also does a relay attack, which we call "independent of relay."

Attacks using leaked attestation keys. Our first observation is that attacks based on leaked attestation keys (e.g., ones obtained through the Foreshadow attack [8]) are independent of relaying. If the adversary has obtained a valid and non-revoked attestation key, he can emulate an SGX processor on the target platform and obtain any secrets provisioned to it. We revisit such emulation attacks and propose a solution for addressing them in Appendix A.

Physical side channels. One major benefit of the relay, from the adversary's point of view, is that it enables *physical* side-channel attacks against application enclaves. Once a secret has been provisioned to the attacker's platform, she has as much time as she likes to perform the attack. Some examples of physical side-channel attacks are acoustic, electric and electromagnetic monitoring, which have been shown to be both effective and inexpensive means to extract secrets from modern PC platforms (see [13] for a summary of known attacks). Since the adversary does not have physical access to the target platform, such attacks are clearly not possible without relay. Hardening programs like enclaves against physical side channels is difficult and currently an open problem [13]. Therefore, developers cannot easily defend their enclaves against physical side channels that are enabled by attestation redirection.

Privilege escalation for digital side channels. Another possible benefit of relay attacks is that it may enable *privilege escalation*. In cases where the adversary has only compromised the user-space application that manages the enclave, and not the OS, the application can redirect the attestation to the attacker's remote platform where he controls the OS as well. In such cases, the relay enables *digital* side-channel attacks that require system privileges. Several such attacks have been recently demonstrated against SGX [7, 14, 24].

Attacks that depend on timing of events. The third, and perhaps the most subtle, implication of relay is that it can also enable software-based side-channel attacks that would not be possible to launch on the target platform due to *timing of certain events*. These events include, but are not restricted to the provisioning of secrets to the enclave, the possible disinfection of the target platform from malicious software, and the discovery of a new side-channel attack.

We group the relative ordering of these events into two cases: A and B. Case A covers event sequences that only lead to attacks which are independent of relay and Case B covers event sequences in which redirection gives extra capabilities to the adversary. Below, and in Figure 3, we provide examples of sequences belonging to these two cases:

Case A: independent of relay. A digital side-channel is independent of relay if the adversary could perform it on the target platform as well. An example of such case is shown th timeline depicted in

Figure 3, where a new attack is discovered after secret provisioning but before the target platform OS is disinfected.

Case B: attack enabled by relay. Case B is reached whenever it occurs that by using a side channel the enclave is exploitable on the attacker's platform, but not on the target platform. A timeline of such case is shown in Figure 3, where at the time of attestation and secret provisioning, the enclave is hardened against all known digital side-channel attacks (using tools like Raccoon [26], ZeroTrace [27] or Obfscuro [4]). After secret provisioning, the OS compromise is detected and cleaned. Later, a new side-channel attack vector (that is not prevented by the used tools) is discovered. If the adversary performed redirection and the secret was provisioned to the attacker's machine, the new side channel is exploitable. Without the relay, the attack is not possible.

3.3 Limitations of Known Solutions

Next, we review commonly suggested solutions and their limitations.

Trust on first use. A common "solution" in the research literature is to rely on *trust on first use* (TOFU) [34]. Simple TOFU solutions assume that the OS is clean at the time of attestation or perform attestation only immediately after fresh OS installation. Both of these approaches have obvious security and deployment problems. OS re-installation is not always possible and trusting the OS, even if momentarily, is undesirable (and violates the SGX's trust model).

SGX attestation variants. As we explain in Section 2.1, SGX supports different variants of remote attestation. Unfortunately, none of these schemes prevents relay attacks without some form of TOFU assumption.

1. *The EPID attestation scheme* is based on group signatures, and thus the remote verifier cannot distinguish between attestation responses that are received from the expected target platform or the adversary's platform. To accept a successful attestation, the remote verifier must rely on trust on first use.

2. *The linkable EPID attestation mode* allows the remote verifier to check if he has attested the same platform before, but the first attestation protocol run is vulnerable to relay attacks, and therefore also in this case the remote verifier must assume TOFU.

3. *The DCAP scheme* allows corporations to operate their own local attestation services after an enrollment phase. However, if the adversary controls the target platform during the enrollment, he can replace the enrolled platform identifier PPID with the identifier of his own platform PPID' and enroll the adversary's platform instead. Thus, also the DCAP variant scheme requires trust on first use. In addition, the entire corporate key management system must be trusted at the time of the enrollment (and after it).

Non-anonymous attestation. Because SGX's attestation protocol support anonymity features, like the EPID signature scheme, one may think that relay attacks are caused by such privacy protection mechanism. However, such reasoning is incorrect. Even if all anonymity features would be removed from attestation, the problem of relay attacks would still persist. The root cause of relay attacks is that certified keys can be securely installed to processors at the time of manufacturing, but the processor ownership by private individuals or companies is established much later. Therefore, common PKI mechanism do not eliminate relay attacks — unless the processor manufacturing and distribution model is completely changed such that factories start to manufacture and certify customers-specific processors batches on demand (which would be very expensive).

Other TOFU variants. Recent research papers use slightly different TOFU variants. For example, the Rote system [21] assumes fresh OS installation at system initialization time and for each used platform it requires a local administrator to input a credential to the enclaves. As another example, in the VC3 system [29] enclaves generate a public/private key pair at the time of trusted initialization, output the public key and seal the private key. The public key can be sent to a trusted authority for certification, which then enables clients to securely connect to enclaves. Both of these solutions essentially avoid insecure attestation by pre-authorizing known enclaves during a setup phase that is assumed trusted.

In general, TOFU solutions suffer from the following limitations:

1. *OS re-installation:* Forcing users or administrators to re-install the OS is not always possible.

2. *Manual configuration:* Manual interaction tasks, such as an administrator that needs to enter credentials to enclaves during initialization, complicates platform enrollment, especially in scenarios like data centers with many enrolled platforms.

3. *Pre-defined enclaves:* Solutions that only work with enclaves that are known at the time of initialization are not applicable to scenarios like cloud computing platforms where users need to install new enclaves after platform installation.

4. *Large temporary TCB:* Modern operating systems have a large TCB and trusting the OS even temporarily is unideal.

5. *Online authorities:* Solutions where a trusted authority needs to either certify or revoke new enclaves typically require that the authorities are online, which increases their attack surface.

4 PROXIMITEE

Our goal is to design a solution that addresses the above limitations of previous solutions. In short, our solution should be *secure* (no TOFU assumption, small TCB, no online authorities) and *easy to deploy* (no OS re-installation, manual configuration or pre-defined enclaves). In this section, we provide an overview of our approach, outline possible use cases, describe our solution in detail and analyze its security.

4.1 Approach Overview

We propose a hardened SGX attestation scheme, called ProximiTEE, based on a simple embedded device that we call ProximiKey. The embedded device is attached to the target platform over a local communication interface such as USB.

Our main idea is to use the combination of such trusted device and *proximity verification* to prevent relay attacks. In our solution, the ProximiKey device verifies the proximity of the attested enclave and after successful proximity verification it facilitates the creation of a secure channel between the remote verifier and the attested enclave. After the initial attestation, the device periodically checks proximity to the attested enclave. The established secure channel is contingent on the physical presence of the embedded device on the target machine and it stays active only as long as the device is plugged-in. The act of detaching the device automatically revokes the attested

platform without any interaction with a trusted authority. Thus, our solution enables secure *offline* enrollment and revocation.

To use our solution, enclave developers use a simple API that facilitates communications between the enclave and the device.

Security assumptions. In our solution, the ProximiKey device is a trusted component. We deem this choice reasonable since it implements only the strictly necessary functions and therefore it a has significantly smaller software TCB, attack surface, and complexity compared to a general-purpose commodity OS. We assume that its issuer certifies each embedded device prior to its deployment and such certification can take place fully offline.

Concerning the security of the ProximiKey device we employ the same adversary model introduced in Section 3 for enclaves. While the user's device and its private keys are never exposed to the attacker, another similar device can be in the physical possession of the attacker, which has as much time as she wants to fully compromise it (run arbitrary code and extract keys).

4.2 Example Use Cases

Our solution is targeted to scenarios where the benefits of more secure attestation outweigh the deployment cost of a simple embedded device. Here, we outline three example cases.

Data center. In our first example, we consider a cloud platform provider that attaches ProximiKey to a server in a specific data center and makes the public key of the connected device known to the users of the service. Our approach is particularly well suited to cloud computing models where customers rent dedicated computing resources like entire servers. In such a setting, our solution ensures that the cloud platform customer outsources data and computation to a server that resides in a specified location. Enforcing location may be desirable to meet increasing data protection regulation that defines how and where data can be stored, even if protected by TEEs such as SGX. Revocation (e.g., when a server is relocated to another data center or function) can be realized by merely detaching ProximiKey.

Permissioned blockchain. Our second case is a setting in which a trusted authority initializes a set of validator nodes for a permissioned and SGX-hardened blockchain. The trusted authority issues one ProximiKey for each organization that operates one of the validator nodes which allows secure attestation of the validator platforms. Organizations are free to upgrade their computing platforms by attaching the ProximiKey to a new platform which automatically revokes the old platform without the need to interact with a trusted authority. Furthermore, since ProximiKey can only be active on one platform at the time, such a deployment enables the authority to control the identities used in (Byzantine) blockchain consensus process.

HSM-protected keys. Our last case is the management of HSM-protected keys from an attested enclave. Such deployment enables the secure and flexible realization of various access control policies, implemented as attested enclaves. ProximiTEE guarantees that only an enclave in the proximity of the HSM can control its keys. Such solution provides a high level of protection because, at no point in time, the HSM keys are directly accessible by the enclave (which may be vulnerable to side-channel attacks) or by the untrusted OS.

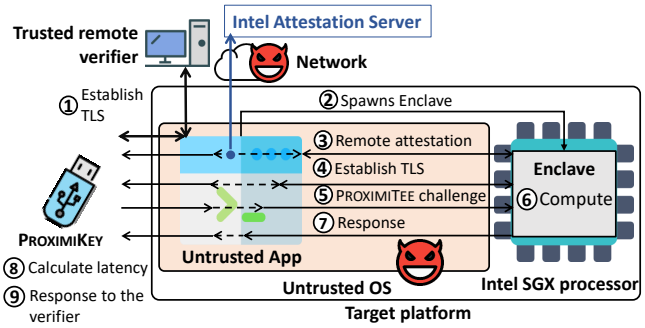

Figure 4: ProximiTEE attestation. The remote verifier establishes a secure channel to the ProximiKey device that first attests the enclave and then verifies its proximity.

4.3 Solution Details

Now, we explain the ProximiTEE attestation mechanism in detail.

I. Attestation protocol. Figure 4 illustrates the attestation protocol that proceeds as follows:

① The remote verifier establishes a secure channel (e.g., TLS) to the certified ProximiKey. An assisting but untrusted user-space application facilitates the connection on the target platform acting as a transport channel between the remote verifier and the ProximiKey (and later also the enclave). As part of this first step, the remote verifier specifies which enclave should be executed.

② The untrusted application creates and starts the attestation target enclave.

③ ProximiKey performs the standard remote attestation to verify the code configuration of the enclave with the help of the IAS server or using a custom DCAP procedure (see Section 2). In the attestation protocol, the device learns the public key of the attested enclave.

④ ProximiKey establishes a secure channel (e.g., TLS) to the enclave using that public key.

⑤ ProximiKey performs a distance-bounding protocol that consists of n rounds, where each round is formed by steps ⑤ to ⑧. At the beginning of each round ProximiKey generates a random challenge r and sends it to the enclave over the TLS channel.

⑥ The enclave increments the received challenge by one $(r+1)$.

⑦ The enclave sends a response $(r+1)$ back to the ProximiKey over the TLS channel.

⑧ ProximiKey verifies that the response value is as expected (i.e., $r+1$) and checks if the latency of the response is below a threshold (T_{con}). Successful proximity verification requires that the latency is below the threshold for at least $k \times n$ responses, where $k \in (0,1]$ is a percentage of the total number of responses n.

⑨ If proximity verification is successful, the ProximiKey notifies the remote verifier over the TLS channel (constructed in step ①). The verifier starts using the ProximiKey TLS channel to send messages to the enclave.

II. Periodic proximity verification. After the initial connection establishment, the ProximiKey device performs *periodic* proximity verification on the attested enclave. ProximiKey sends a new

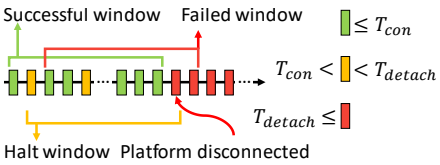

Figure 5: Sliding window for periodic proximity verification with three different types of challenge-response latencies.

random challenge r at frequency f, verifies the correctness of the received response and measures its latency. The latest w latencies are stored to a sliding window data structure, as shown in Figure 5.

As elaborated in Section 5 there are three types of latencies in the presence of relay attacks. The first type of response is received faster than the threshold T_{con} (green in Figure 5), these responses can only be produced if no attack is taking place. In the second type of response the latency exceeds T_{con}, but it is below another, higher threshold T_{detach} (yellow), these are sometimes observed during legitimate connections and sometimes during relay attacks. And third, the latency is equal to or exceeds T_{detach} (red), these latencies are only observed while a relay attack is being performed. Given such a sliding window of periodic challenge-response latencies, we define the following rules for halting or terminating the connection:

1. *Successful window: no action.* If at least k responses have latency $\leq T_{con}$ and none of the response have latency $\geq T_{detach}$, the current window legitimate and ProximiKey keeps the connection active.

2. *Halt window: prevent communication.* If one of the responses have latency $\geq T_{detach}$, we consider the current window a "halt window," and ProximiKey stops forwarding data to the enclave until the current window is legitimate again.

3. *Failed window: terminate channel.* If two or more responses have latencies $\geq T_{detach}$, we consider the current window a "failed window" and ProximiKey terminates the communication and thus revokes the attested platform.

4.4 Security Analysis

Attestation security. To analyze the security of our hardened attestation mechanism, we must first define successful attestation. We say that the attestation is successful when the remote verifier establishes a connection to the correct enclave that i) has the expected code measurement and ii) runs on the computing platform to which the ProximiKey device is attached.

The task of establishing a secure channel to the correct enclave can be broken into two subtasks. The first subtask is to establish a secure channel to the correct ProximiKey device. This is achieved using standard device certification. We assume that the adversary cannot compromise the specific ProximiKey used. If the adversary manages to extract keys from other ProximiKey devices, he cannot trick the remote verifier to connect to a wrong enclave, as the remote verifier will only communicate with a pre-defined embedded device.

The second subtask is to establish a secure connection from ProximiKey to the correct enclave. For this, we use proximity verification. ProximiKey verifies the proximity of the attested enclave through steps ⑤ to ⑧ of the protocol. These steps essentially check two things. First, through step ⑦, whether the messages are received from the correct enclave. This verification is performed by checking the correctness of the decrypted message, and it relies on the assumption that the attacker cannot break the underlying encryption and hence only the enclave that has access to the key that was bound to the attestation could have produced a valid reply. Second, through step ⑧, whether the ProximiKey and the enclave are in each other's proximity. This check relies on the assumption that a reply from a remote enclave will take more time to reach the ProximiKey than a reply from the local enclave.

We evaluate the second aspect experimentally. In particular, we simulate a powerful relay-attack adversary that is connected to the target platform with fast network connection. To consider the best case for the adversary, we make several assumptions in his favor. For example, we assume that he can instantly perform all computations needed to participate in the proximity verification protocol. However, he cannot break cryptographic hardness assumptions. We define the adversary's success as the event in which proximity verification succeeds with an enclave that resides on the attacker's platform and denote the probability of such event P_{adv}. We define the legitimate success as the event in which proximity verification succeeds with an enclave that resides in the target platform and denote its probability P_{legit}.

In Section 5 we show that it is possible to find parameters ($n = 50$, $k = 0.3$ and $T_{con} = 186\mu s$) that make proximity verification very secure ($P_{adv} = 3.55 \times 10^{-34}$) and reliable ($P_{legit} = 0.999999977$).

Revocation security. To analyze the security of the periodic proximity verification which we use for platform revocation, we must first define what it means for the attacker to break the periodic proximity verification. The purpose of the periodic proximity verification is to prevent cases where the user detaches the ProximiKey device from the attested target platform and attaches it to another SGX platform before the previously established connection is terminated. Since we consider an adversary who does not have physical access to the target platform (recall Section 3.1), we focus on benign users and exclude scenarios where the ProximiKey would be connected to multiple SGX platforms with custom wiring or rapidly and repeatedly plugged in and out of two SGX platforms.

We define the periodic proximity verification as broken if the adversary can manage to keep the previously established connection alive within a "short delay" after the ProximiKey was detached from the attested target platform. For most practical purposes we consider a delay of 10 ms as sufficiently short. We denote the adversary's success probability in breaking the periodic proximity verification as P'_{adv}. A false positive for periodic attestation is the event where the connection to the legitimate enclave is terminated, and the attested platform is revoked despite the ProximiKey being connected to the target platform. We denote the probability that this happens during a "long period" as P'_{fp}. We consider an example period of 10 years sufficiently long for most practical deployments.

In Section 5 we experimentally shows that revocation can be secure ($P'_{avd} = 3.55 \times 10^{-34}$) and reliable ($P'_{fp} = 1.6 \times 10^{-4}$) while consuming only a minor fraction of the available channel capacity.

5 EXPERIMENTAL EVALUATION

In this section, we describe our implementation and evaluation.

5.1 Implementation

We implemented a complete prototype of the ProximiTEE system. Our implementation consists of two components: i) ProximiKey embedded device prototype, and ii) ProximiTEE enclave API which enables any application enclaves to communicate with the ProximiKey device and execute the proximity verification protocols.

ProximiKey. Our embedded device prototype is based on Cypress EZ-USB FX3 USB 3.0 prototyping board that is equipped with an 32-bit 200 MHz ARM9 core. The board communicates with the target platform over a native USB 3.0 connection that provides up to 5 Gbps of bandwidth. FX3 provide direct memory access (DMA) out of the box through its API for efficient communication with the connected platform. We use the ARM mbed TLS [18] cryptographic library for the TLS. The limited set of cipher suites in our implementation uses 128-bit AES (CTR mode) for encryption, AES-HMAC as the message authentication code, Curve25519 for Diffie-Hellman key exchange and SHA256 as the hash function. Our prototype implementation is approximately 200 lines of code, and the code size of the TLS library is around 3.6 KLoC.

ProximiTEE enclave API. The ProximiTEE API for application-specific enclaves is written in C++ using the Intel SGX API. The API uses native SGX crypto library for the TLS implementation, and it is around 200 lines of code.

5.2 Evaluation Focus: Internet Relay

For the purposes of our evaluation, we make the distinction between two types of relay attacks. In the first type, the adversary redirects the attestation *over the Internet* to another platform that is under his physical control, and therefore in a *different location*. As we explained in Section 3.2, such relay attack amplifies the adversary's capabilities the most, as he can now attack the attested enclave using physical side-channels, he has unlimited time to launch digital side-channels, or he can wait for the discovery of new attack vectors.

In the second type of relay attack, the adversary redirects the attestation to another *co-located platform*, like another server on the same server rack. In most cases, attestation relay to a co-located platform does not improve the adversary's chances of attacking the enclave, because typically the adversary has similar control over the co-located platform. The only exception is privilege escalation in cases where the adversary has user privileged on the target platform and system privileges on the co-located platform.

Next, we focus on demonstrating that an inexpensive ProximiTEE prototype can be configured to prevent the first (and typically more dangerous) type of relay attacks with very strong security and robustness. Later, in Section 5.8, we discuss the second type of relay.

5.3 Experimental Setup

To demonstrate that ProximiTEE prevents relay attacks (over the Internet) we performed two types of experiments. First, we tested the legitimate attestation execution with ProximiTEE and measured the challenge-response latencies between our prototype and the target platform. Second, we *simulated* a relay attack, where the adversary redirects the attestation to another platform.

Assumptions and optimizations. To consider the best possible case for the adversary, we made several generous assumptions in his favor, when designing our experimental setup and post-processing of our measurement:

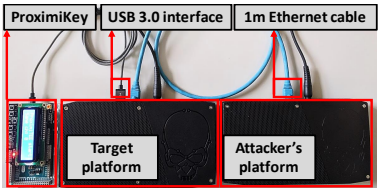

Figure 6: Our experimental setup consists of the ProximiKey device prototype, the target platform, the attacker's platform and the connection interfaces between them.

1. *Single network hop.* Since we do not want to make any assumptions about the precise network path that the relayed attestation needs to travel, we connected the adversary's platform to the target platform via a direct 1-meter Ethernet cable as seen in Figure 6. With such setup, our goal is to simulate the most direct connectivity and the best possible latency that the adversary could achieve in relay attacks that take place over the Internet. In most realistic attacks, the adversary would need to relay the attestation over multiple network hops which increases the round-trip latency significantly.

2. *Instant protocol computation.* Since the adversary might have a faster processor on his platform than the one the one we used in our experiments, we simulated an adversary who is able to perform all computations needed for the proximity verification protocol instantly. Instant replies were simulated by fixing the randomness for the challenges and having precomputed responses for that randomness on the attacker's machine.

3. *Packet forwarding optimizations.* Since the adversary controls the OS on the target platform, he can perform software-based optimizations to reduce the packet forwarding delay. We experimented with several such optimizations. First, we tested the standard `ping` tool which gave a latency of around 380 μs for one-meter Ethernet connection. After that, we used the `ping` tool in so called flood mode and measured a reduced average network latency of around 153 μs (command `ping -s 300 -af`). Flood mode achieves faster round-trip time as the it forces the OS to fill up the network queue of the kernel. Based on these measurements, we chose to simulate an attacker that fills the kernel's network queues (on both platforms) similar to the flood mode to minimize latency. We also tested other possible OS-level optimizations, but did not observe material reduction in measured latencies, and thus in our experiments we only use the kernel queue filling.

4. *Infinitely fast network interface.* Since the adversary's platform might have a faster network interface hardware than the one used in our experiments, we chose to simulate an adversary that has infinitely fast network interface. In our experimental setup, both the target platform and the adversary's platform have identical network interfaces. We assume (in the favor of the adversary) that the transmission time spent on the wire is negligible and most the the round-trip latency is due to processing the in the network interface. This allows us to simulate an adversary with infinitely fast network interface by first performing latency measurements and then in a post-processing phase cutting down all the measured latencies by

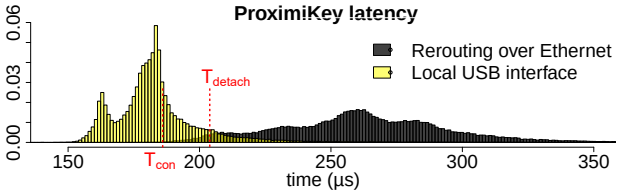

Figure 7: Latency distributions for legitimate challenge-response rounds (left) and simulated relay attack (right).

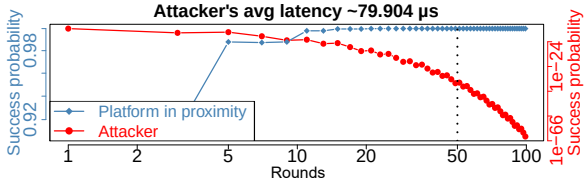

Figure 8: Parameter tuning: the attacker's success probability P_{adv} and the legitimate success probability P_{legit} for different number of rounds n given a fixed k.

half. Note that the target platform's network interface cannot be replaced by the attacker has he does not have physical access to it.

Experiments. We conducted our experiments on three SGX platforms: two Intel NUC NUC6i7KYK mini-PCs and one Dell Latitude laptop, all equipped with SGX-enabled Skylake core i7 processors and Ubuntu 16.04 LTS installed on them. To measure latencies we used FX-3's GPIO pins that provides 100 nanosecond level accuracy. We performed a total of 20 million rounds of the protocol for normal attestations and simulated attacks and measured the challenge-response latencies for each. We measure all of them inside the EZ-USB FX3 code. For cross-validation, we tested the ProximiKey with the high precision oscilloscope and witnessed identical timing patterns.

5.4 Latency Distributions

Figure 7 shows our main experimental result. The histogram on the left represents the challenge-response latencies in the legitimate proximity verification. The histogram on the right shows latencies in a simulated attack (including a post-processing phase where we reduce the adversary's measured network latencies to half to accommodate the assumption of the attacker's infinitely fast network interface).

As can be seen from Figure 7, the vast majority of the benign challenge-responses take from 145 to 250μs (average is 185μs, 95% of samples are in between 150μs and 200μs). The vast majority of the round-trip times in the simulated attack take from 200 to 750 μs (average is 264μs, 95% of samples are in between 209μs and 650 μs). Hence, the average delay of our simulated adversary is only 80μs. To put this into perspective, even the highly-optimized network connections between major data centers in the same region exhibit latencies from one millisecond upwards [3] which is one order of magnitude more than in our simulated setup.

Besides the latency observed on the side of the embedded device, we measured the time required to compute responses to received challenges on the side of the target platform. We repeated these test on three different SGX platforms and observed results that varied from 6 to 10 μs. We also measured if the computational load of the target platform influences the time required to compute responses. Under maximum system load (all 8 cores busy), the maximum observed time increased to 20 μs. Under moderate system load (1 or 2 cores busy), we experience no notable increase in the required computation time. For completeness, we provide such additional measurement results in Appendix B.

5.5 Initial Proximity Verification Parameters

As explained in Section 4.3, the initial proximity verification is successful when at least fraction k of the n challenge-response latencies are below the threshold T_{con}. Now, we explain our strategy for setting these parameters based on the above results.

There are five interlinked parameters that one needs to consider: (i) the legitimate connection latency threshold T_{con}, (ii) total number of challenge-response rounds n, (iii) the fraction k, (iv) attacker's success probability P_{adv} that should be negligible, and (v) the legitimate success probability P_{legit} that should be high. We find suitable values for these parameters in the following order:

1. We start with the threshold T_{con}. The higher T_{con} is, the higher the legitimate success probability P_{legit} becomes, on the other hand, a too high value for T_{con} also makes P_{adv}, the attacker's success probability, high. Therefore, we are after a suitable value for T_{con} that keeps P_{legit} high while minimizing P_{adv} over a varied number of rounds n.
2. Based on such T_{con}, we pick a fraction k such that it maximizes the legitimate success probability P_{legit} and reduces the attacker's success probability P_{adv}.
3. Given T_{con} and k, we evaluate P_{adv} and P_{legit} over a varied number of rounds n and choose the minimum number of rounds that provides the required probabilities, since the fewer rounds, the faster the initial attestation is.

Main result. Figure 8 shows the legitimate enclave's success probability P_{legit} and the attacker's success probability P_{adv} with different number of rounds. Based on our experiments we set T_{con} = 186μs (see Figure 7), the threshold fraction $k = 0.3$ and the number of rounds $n = 50$ which yields a very high legitimate success probability $P_{legit} = 0.999999977$ and a negligible attacker's success probability $P_{adv} = 3.55 \times 10^{-34}$. For completeness, we provide the full details of of this parameter tuning process in Appendix B.

5.6 Periodic Proximity Verification Parameters

For periodic proximity verification we have two main requirements. First, the attacker's success probability P'_{adv} must be negligible. Recall that P'_{adv} refers to an event where the device is detached but the connection is not terminated sufficiently fast. Second, the probability of false positives P'_{fp} should be very low. P'_{fp} refers to an event where the connection is terminated when the device is still attached. Next, we explain the three-step process to set up parameters T_{detach}, w and f for the periodic proximity verification:

1. We find out a suitable latency T_{detach} that define the yellow or red round in Figure 5. The yellow window defines the round of challenge response latency between T_{con} and T_{detach}, while the red window defines a latency more than T_{detach}. Hence, the probabilities $\Pr[T_{con} \leq \mathcal{L}_{legit} \leq T_{detach}] = \Pr[legit \in \text{yellow}]$, and $\Pr[\mathcal{L}_{legit} \geq T_{detach}] = \Pr[legit \in \text{red}]$ should be very low. $\mathcal{L}_{legit}$ and $\mathcal{L}_A$ denote the latency of the legitimate enclave running on the platform in proximity and remote attacker platform's latency respectively.

2. Based on the threshold T_{detach}, we select a suitable sliding window size w to minimize the attacker success probability P'_{adv} to a negligible quantity.

3. We fix a suitable frequency f for the periodic challenges. A high f value terminate the communication very fast, leaving very small attacking window.

Main result. Based on the above strategy, we set the periodic proximity verification parameters as follows: $\Pr[A \in \text{success window}] = P'_{adv} = P'_{fn} = 3.55 \times 10^{-34}$, $\Pr[legit \in \text{success window}] = 0.999999977$ and $\Pr[legit \in \text{failed window}] = P'_{fp} = \Pr[legit \in \text{red}]^2 = 1.6 \times 10^{-4}$ and $T_{detach} = 205\mu s$ (see Figure 7). If at least two latencies above T_{detach} are received, the ProximiKey terminates the connection and revokes the platform. The average downtime due to false positives occurring during a connection of 10 years is around 2 minutes. We provide the full details of the parameter tuning in Appendix B.

5.7 Performance Analysis

In addition, we evaluated the following two performance metrics:

1. *Start-up latency.* The initial proximity verification takes 2 ms. The complete connection establishment including attestation and TLS handshake takes less than 1 second.

2. *Operational latency and data overhead.* Our solution adds around $200\mu s$ of additional latency for TLS and transport over the native USB interface of the FX3. The data overhead is around 80 bytes per packet for the header and the MAC. Execution of the periodic ProximiTEE protocol with 83 rounds/second requires around 156.14 KBytes/s of data which is only 2.4×10^{-3}% of the USB 3.0 channel capacity.

5.8 Preventing Relay to Co-Located Platform

The main purpose of our experimental evaluation was to show that our inexpensive ProximiTEE prototype can effectively prevent relay attacks where the adversary redirects the attestation to another platform that is under his physical control in a *different location*. Next, we discuss whether ProximiTEE can prevent attestation redirection a *co-located* platform, like another server on the same server rack.

If the two co-located platforms are connected through traditional networking technologies like Ethernet (as in our experiments), our evaluation already shows that such relay attacks can be effectively prevented, using a simple and inexpensive embedded device like our prototype. However, in some modern data centers, computing platforms are connected with faster inter-connect technologies like InfiniBand connections [1] that can enable latencies as lows as $7\mu s$ [20].

The ability to distinguish relay attacks depends on three key factors. The first is the latency of the channel through which the relay is performed (e.g., $7\mu s$ for InfiniBand). The second is the time required to compute responses to challenges on the target platform (e.g., $6-10\mu s$ in the SGX platforms that we tested). And the third is how much variance the round-trip times between the embedded device and the target platform have (e.g., $10-20\mu s$ in our USB 3.0 prototype). The local communication variance and the response computation time should be less than the relay latency, to enable robust proximity verification.

We conclude that our simple prototype cannot prevent all possible relays to co-located platforms when very fast inter-connect technologies like InfiniBand are used. To address such relay attacks, one needs a faster and more accurate embedded device that exhibits less variance. For example, PCIe connected FPGAs can have latencies as lows $1\mu s$ [5]. Besides better embedded device, one can also increase the number of distance-bounding protocol rounds and reduce the success probability for legitimate attestation P_{legit}.

6 DISCUSSION AND RELATED WORK

Addressing emulation attacks. During the last year, several micro-architectural attacks like Spectre [16] and Meltdown [19] that leveraged subtle side effects of transient execution optimizations on modern CPUs were discovered. ForeShadow [8], a Meltdown variant, was the first attack that managed to extract production attestation keys from SGX processors.

After Meltdown and Foreshadow, Intel issued a microcode update that addressed them, but soon after that other attacks named microarchitectural data sampling (MDS) by Intel were discovered [23, 30, 32]. Although the authors of the MDS attacks have not yet been able to verify their attack against the production Quoting Enclaves, it appears [30, 32] that the MDS attacks may enable attestation key extraction. Given such recent history of several micro-architectural attacks, leakage of attestation keys from SGX processors becomes a threat that should be considered. Proximity verification alone is not sufficient to prevent relay attacks against an adversary that has leaked but not yet revoked attestation keys, as such adversary can *emulate* a valid enclave locally on the target platform. To prevent such attacks, one must prevent the adversary's ability to emulate enclaves on the target platform.

In Appendix A we present a solution that does this using a *secure boot-time initialization* with the help of the trusted embedded device. In our solution, we combine secure initialization and local attestation with periodic proximity verification to enable convenient revocation. Our solution can be seen as a novel variant of the well-known TOFU principle with security, deployment and renovation benefits. Due to space constraints, we defer the details of this solution to Appendix A.

Trusted path. Recent research has proposed trusted path solutions like Fidelius [11] where a trusted embedded device is used to capture user input and send it securely to an attested *local* enclave. Such solutions are vulnerable to relay attacks where confidential user input is sent to a *remote* enclave on the adversary's platform. ProximiTEE could be integrated to solution like Fidelius to prevent relay attacks and to improve their security.

Extension to other TEEs. Our approach could be applied to other TEEs as well. The critical requirements for the TEE is that it must support programmable operations that can be executed sufficiently fast. One TEE that meets these requirements is ARM TrustZone.

DRTM proximity verification. Presence attestation [36] enables proximity verification DRTM-based TEEs [22]. The TEE shows an image that is captured by a trusted camera and communicated to a

remote verifier. The same approach cannot be used with SGX since it lacks trusted path for secure image output.

7 CONCLUSION

Relay attacks have been known for a decade, but their implications to modern TEEs like SGX have not been carefully analyzed. In this paper we have presented the first such analysis and shown that attestation redirection increases the adversary's ability to attack an attested enclave. We have also proposed ProximiTEE as a solution to prevent relay attacks using proximity verification. Our experimental evaluation is the first to show that proximity verification can be made secure and reliable for TEEs like SGX. As an additional contribution, we have also presented a novel boot-time initialization solution for addressing a stronger emulation attacker who has leaked attestation keys.

REFERENCES

[1] 2018. Infiniband Trade Association. https://www.infinibandta.org/.
[2] 2018. Tiny Core Linux, Micro Core Linux, 12MB Linux GUI Desktop, Live, Frugal, Extendable. https://distro.ibiblio.org/tinycorelinux/.
[3] Sachin Agarwal. 2018. Public Cloud Inter-region Network Latency as Heat-maps. https://medium.com/@sachinkagarwal/public-cloud-inter-region-network-latency-as-heat-maps-134e22a5ff19
[4] Adil Ahmad, Byunggill Joe, Yuan Xiao, Yinqian Zhang, Insik Shin, and Byoungyoung Lee. 2019. OBFSCURO: A Commodity Obfuscation Engine on Intel SGX. NDSS'19.
[5] Inc. Algo-Logic Systems. 2019. Low Latency PCIe Solutions for FPGA. https://www.algo-logic.com/sites/default/files/PCIe.pdf.
[6] Stefan Brands and David Chaum. 1993. Distance-Bounding Protocols. In *EUROCRYPT '93*.
[7] Ferdinand Brasser, Urs Muller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software Grand Exposure: SGX Cache Attacks Are Practical. In *USENIX WOOT'17*.
[8] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *USENIX Security'18*.
[9] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086.
[10] Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. 2018. Cachequote: Efficiently recovering long-term secrets of SGX EPID via cache attacks. *TCHES* 2018, 2 (2018).
[11] Saba Eskandarian, Jonathan Cogan, Sawyer Birnbaum, Peh Chang Wei Brandon, Dillon Franke, Forest Fraser, Gaspar Garcia Jr, Eric Gong, Hung T Nguyen, Taresh K Sethi, et al. 2019. Fidelius: Protecting User Secrets from Compromised Browsers. In *S&P'19*.
[12] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. 2001. Electromagnetic analysis: Concrete results. In *CHES'01*.
[13] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Adi Shamir, and Eran Tromer. 2016. Physical key extraction attacks on PCs. *Commun. ACM* 59, 6 (2016).
[14] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache attacks on Intel SGX. In *EuroSec'17*.
[15] Simon Johnson and Intel. 2017. Intel SGX: EPID Provisioning and Attestation Services. https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services.
[16] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *S&P'19*.
[17] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *USENIX Security'17*.
[18] ARM Limited. [n. d.]. SSL Library mbed TLS / PolarSSL. https://tls.mbed.org/
[19] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. *USENIX Security'18*.
[20] Jiuxing Liu, Balasubramanian Chandrasekaran, Jiesheng Wu, Weihang Jiang, Sushmitha Kini, Weikuan Yu, Darius Buntinas, Pete Wyckoff, and Dhabaleswar K Panda. 2003. Performance comparison of MPI implementations over InfiniBand, Myrinet and Quadrics. In *SC'03*.
[21] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. 2017. ROTE: Rollback Protection for Trusted Execution.. In *USENIX Security'17*.
[22] Jonathan M McCune, Bryan J Parno, Adrian Perrig, Michael K Reiter, and Hiroshi Isozaki. 2008. Flicker: An execution infrastructure for TCB minimization. In *ACM SIGOPS Operating Systems Review*.
[23] Marina Minkin, Daniel Moghimi, Moritz Lipp, Michael Schwarz, Jo Van Bulck, Daniel Genkin, Daniel Gruss, Frank Piessens, Berk Sunar, and Yuval Yarom. 2019. Fallout: Reading Kernel Writes From User Space. *arXiv preprint arXiv:1905.12701* (2019).
[24] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. Cachezoom: How SGX amplifies the power of cache attacks. In *CHES'17*.
[25] Bryan Parno. 2008. Bootstrapping Trust in a Trusted Platform. In *HotSec'08*.
[26] Ashay Rane, Calvin Lin, and Mohit Tiwari. 2015. Raccoon: Closing Digital Side-Channels through Obfuscated Execution.. In *USENIX Security'15*.
[27] Sajin Sasy, Sergey Gorbunov, and Christopher W Fletcher. 2017. ZeroTrace: Oblivious memory primitives from Intel SGX. In *NDSS'17*.
[28] Vinnie Scarlata, Simon Johnson, James Beaney, and Piotr Zmijewski. 2018. Supporting Third Party Attestation for Intel SGX with Intel Data Center Attestation Primitives. https://software.intel.com/sites/default/files/managed/f1/b8/intel-sgx-support-for-third-party-attestation.pdf.
[29] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *S&P'15*.
[30] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. 2019. ZombieLoad: Cross-Privilege-Boundary Data Sampling. *arXiv:1905.05726* (2019).
[31] Adi Shamir and Eran Tromer. 2004. Acoustic cryptanalysis. *presentation available from http://www. wisdom. weizmann. ac. il/ tromer* (2004).
[32] Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2019. RIDL: Rogue In-Flight Data Load. In *2019 IEEE Symposium on Security and Privacy (SP)*.
[33] Zhenghong Wang and Ruby B Lee. 2006. Covert and side channels due to processor architecture. In *ACSAC'06*.
[34] Dan Wendlandt, David G. Andersen, and Adrian Perrig. 2008. Perspectives: Improving SSH-style Host Authentication with Multi-path Probing. In *USENIX ATC'08*.
[35] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *S&P'15*.
[36] Zhangkai Zhang, Xuhua Ding, Gene Tsudik, Jinhua Cui, and Zhoujun Li. 2017. Presence Attestation: The Missing Link in Dynamic Trust Bootstrapping. In *CCS '17*.

A ADDRESSING EMULATION ATTACKS

We consider attestation key extraction from SGX processors difficult and rare, in contrast to the previously considered relay attacks that require only OS control or other malicious software on the target platform. However, the recently demonstrated Foreshadow attack [8] that exploited the Meltdown vulnerability [19] showed how to extract attestation keys from SGX processors. Although, Intel has the possibility to issue microcode patches that address processor vulnerabilities like Meltdown and the processor's microcode version is reflected in the SGX attestation signature, new vulnerabilities like the ZombieLoad attack [30] may be discovered. Before microcode patches are deployed, in rare occasions, leaked but not revoked attestation keys may be available to the adversary.

In this appendix, we consider such stronger adversary that has leaked attestation keys and present a hardened attestation solution based on boot-time initialization.

A.1 Emulation Attack

Adversary model. We consider an *emulation attacker* has all the capabilities of the relay attacker (cf. Section 3) and additionally has obtained at least one valid (not yet revoked by Intel) attestation key from any SGX platforms but the target platform. The adversary might obtain an attestation key by attacking one of his processors or by purchasing an extracted key from another party.

The emulation attack. In the attack, the adversary uses a leaked attestation key to emulate an SGX-processor on the target platform. Since the IAS (or any other attestation service) successfully attests

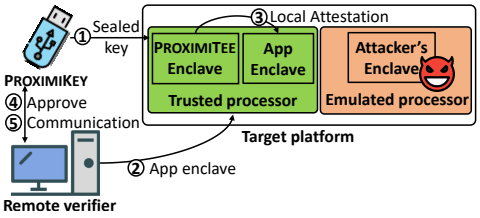

Figure 9: ProximiTEE boot-time attestation. After the boot-time initialization (refer to Figure 10) the ProximiTEE enclave executes a local attestation with the verifier uploaded application-specific enclave.

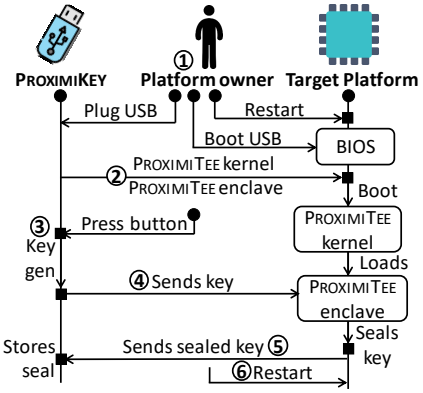

Figure 10: Boot-time initialization. The ProximiKey uses a minimal kernel Linux image to boot and load ProximiTEE enclave on the target platform and seal a platform specific secret to the ProximiKey memory.

the emulated enclave, it is impossible for the remote verifier to distinguish between the emulated enclave and the real one.

Emulation attack implications. The emulation attack allows the adversary to fully control the attested execution environment and thus break the two fundamental security guarantees of SGX, enclave's data confidentiality and code integrity, and to access any secrets provisioned to the emulated enclave. Since the OS is also under the control of the attacker, any attempted communication with the real enclave will always be redirected to the emulated enclave.

A.2 Boot-Time Initialization Solution

Proximity verification alone cannot protect against the emulation attacker, as the locally emulated enclave would pass the proximity test. Therefore, we describe a second hardened attestation mechanism that leverages secure boot-time initialization and is designed to prevent emulation attacks. This solution can be seen as a *novel variant* of the well-known TOFU principle and the main benefits of our solution over previous variants is that it simplifies deployment and increases security. Additionally, when such attestation is used in combination with our previously described periodic proximity verification, our solution enables secure offline revocation.

Security assumptions. Our security assumptions regarding the target platform are as described in Section 3. The only difference is that in this case we assume that the UEFI (or BIOS) on the target platform is trusted.

Solution overview. Figure 9 illustrates an overview of this solution. During initialization, that is depicted in Figure 10, the target platform is booted from the attached device that loads a minimal and single-purpose ProximiTEE kernel on the target device. In particular, this kernel includes no network functionality. The kernel starts the ProximiTEE enclave, which shares a secret with the device. This shared secret later bootstraps the secure communication between ProximiKey and the ProximiTEE enclave. *The security of the bootstrapping relies on the fact that the minimal kernel will not perform enclave emulation at boot time.* The ProximiTEE enclave will later be used as a proxy to attest whether other (application-specific) enclaves in the system are real or emulated and on the same platform.

Boot-time initialization. The boot-time initialization process is performed only once. This process is depicted in Figure 10 and it proceeds as follows:

① The platform owner plugs ProximiKey to the target platform, restarts it to BIOS and selects the option to boot from ProximiKey.

② ProximiKey loads the ProximiTEE kernel and boots from it. The ProximiTEE kernel starts the ProximiTEE enclave.

③ The user presses a button on ProximiKey to confirm that this is a boot-initialization process. This step is necessary to prevent an attack where the compromised OS emulates a system boot.

④ ProximiKey sends a randomly generated key $\mathcal{K}$ to the ProximiTEE enclave.

⑤ The enclave returns the sealed key $\mathcal{S}$ corresponding to the key $\mathcal{K}$ ($\mathcal{S} \leftarrow \texttt{Seal}(\mathcal{K})$) to ProximiKey that stores the key and the seal pair ($\mathcal{K}$,$\mathcal{S}$) on its flash storage.

⑥ ProximiKey blocks further initializations, sends a restart signal and boots the platform with the normal OS.

Attestation process. After initialization the target platform runs a regular OS. The attestation process is depicted in Figure 9 and proceeds as follows:

① ProximiKey sends the seal $\mathcal{S}$ to the ProximiTEE enclave that unseals it and retrieves the key $\mathcal{K}$. ProximiKey and the ProximiTEE enclave establish a secure channel (TLS) using $\mathcal{K}$.

② The remote verifier uploads a new application-specific enclave on the target platform.

③ The ProximiTEE enclave performs local attestation (see Section 2) on the application-specific enclave that binds its public key to the attestation.

④ The ProximiTEE enclave sends the measurement and the public key of the application-specific enclave to ProximiKey. ProximiKey establishes a secure channel to the application-specific enclave and sends the measurement of the enclave to the remote verifier. The remote verifier then approves the communication to the application-specific enclave.

⑤ The remote verifier checks that the measurement of the application-specific enclave is as expected. If this is the case, it can communicate with the enclave through ProximiKey.

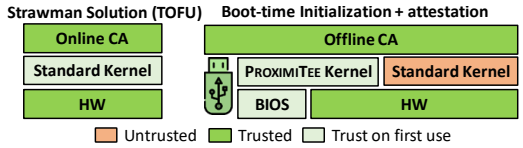

Figure 11: TCB comparison. Trusted components in a common TOFU solution and our boot-time solution.

Following communication. Similar to our previous solution, after the initial attestation all the communication between a remote verifier and the enclave is mediated by the ProximiKey that periodically checks the proximity of the attested enclave and terminates the communication channel in case the embedded device is detached.

A.3 Security Analysis

In this attestation mechanism, the task of establishing a secure communication channel to the correct enclave can be broken into three subtasks. The first subtask is to establish a secure channel to the correct ProximiKey device. In our solution, this is achieved using standard device certification. Recall that the adversary cannot compromise the specific ProximiKey used.

The second subtask is to establish a secure communication channel from ProximiKey to the ProximiTEE enclave. ProximiKey shares a key with an enclave that is started by the trusted ProximiTEE kernel, hence at a time in which the attacker could not emulate any enclave. ProximiKey knows when secure initialization takes place because the user (platform owner) indicates this by pressing a button which is an operation that the adversary cannot perform. The ProximiTEE enclave seals the key during initialization. Different SGX CPUs cannot unseal each other's data, and therefore even if the adversary has extracted sealing keys from other SGX processors, she cannot unseal the key and masquerade as the legitimate ProximiTEE enclave.

The third subtask is to establish a secure communication channel from the ProximiTEE enclave to the application-specific enclave. The security of this step relies on SGX's built-in local attestation. An adversary in possession of leaked sealing attestation keys from other SGX processors, cannot produce a local attestation report that the ProximiTEE enclave would accept, and therefore the adversary cannot trick the remote verifier to establish a secure communication channel to a wrong enclave.

A.4 Comparison to TOFU

Our second attestation mechanism is a novel variant of the well-known "trust on first use" principle. In this section we briefly explain the main benefits of our solution over common TOFU variants.

Smaller TCB size and attack surface. Figure 11 illustrates a comparison of trusted components and attack surface between a common TOFU solution where a trusted authority (CA) certifies enclave keys (cf. Section 3.3) and our boot-time initialization mechanism. In the TOFU solution, the standard and general-purpose OS needs to be trusted on first use and the CA needs to remain online for enrollment of new SGX platforms. In our solution, a significantly smaller and single-purpose kernel needs to be trusted on first use. Additionally, we require trust on the BIOS (or UEFI). In our solution, the CA can remain offline when a new platform is enrolled.

Reboot instead of re-install. Our solution requires that the target platform is rebooted once from ProximiKey. In most TOFU solutions, the target platform requires a clean state which is difficult to achieve without reinstall that makes deployment difficult.

Secure offline revocation. When boot-time initialization is combined with the previously explained periodic proximity verification, our solution provides an additional property of secure offline revocation that requires no interaction with the CA. Such property is missing from previous TOFU solutions.

A.5 Implementation

We implemented a complete prototype of our second attestation mechanism. On top of our previous ProximiTEE implementation (see Section 5.1), the boot-time initialization solution requires the ProximiTEE kernel. We have modified an image of Tiny Core Linux [2] and used it as the boot image for our boot-time initialization. The image size of our modified Linux distribution is 14 MB (in contrast to 2 GB standard 64 bit Linux images build on the standard kernel). Our image supports bare minimum functionality and includes `libusb`, `gcc`, Intel SGX SDK, Intel SGX platform software (PSW), and Intel SGX Linux driver. The ProximiTEE enclave is a minimal enclave that uses a simple serial library to communicate with the ProximiKey and local attestation mechanism to attest any application-specific enclave.

B FURTHER EVALUATION DETAILS

In this appendix we provide further details on how we find suitable parameter values for the initial and periodic proximity verification, and describe additional experimental results such as the effect of system load on proximity verification execution.

B.1 Initial Proximity Verification

In Section 5.5, we outlined a three-step approach to determine suitable parameter values for proximity verification. Here, we provide further details on each of these steps.

1. Finding suitable threshold T_{con}. Finding a suitable threshold T_{con} is a non-trivial task. A very low threshold requires a high number of the challenge-response rounds, since the protocol requires at least a fraction k of the observed responses to be less or equal to T_{con} and a low threshold has very low cumulative probability value in the latency distribution (see Figure 13). Conversely, a very high threshold value enables some latencies measured during an attack to be classified as legitimate replies, hence increasing the chances of the attacker to break the proximity verification. To address this challenge, we perform a trial over multiple threshold candidates to evaluate their viability.

Figure 12 shows the legitimate success probability P_{legit} for different number of rounds ($n \in \{10,20,50,100\}$). We iterate through multiple threshold times ($T_{con} \in \{183\mu s, 184\mu s, 185\mu s, 186\mu s, 189\mu s\}$), and $186\mu s$ provides high success ratio for different values of k ($P_{legit} = 0.9\{7\}77$ ($n = 50$) and $P_{legit} = 0.9\{15\}29$ ($n = 100$)), where $0.9\{n\}x$ denotes $0.n$-times 9 followed by x.

We test T_{con} up until $186\mu s$ because as can be observed in Figure 7 for these values we observe extremely small occurrences (1.33×10^{-3})

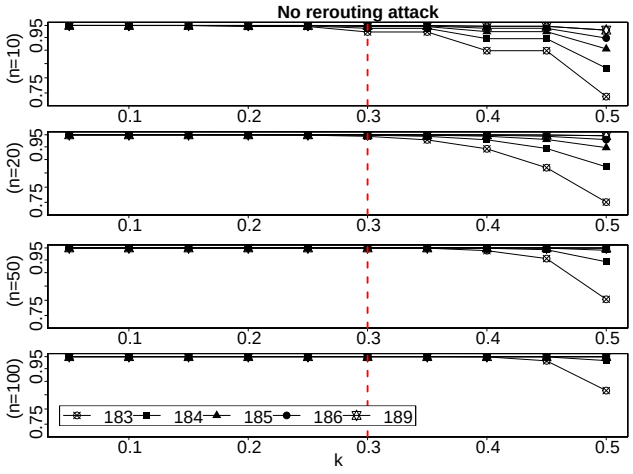

Figure 12: Legitimate attestation success probability for different T_{con} values. The chosen value T_{con} = 186 μs gives success probability 0.999999977 for number of trials at least 15 out of $n = 50$ rounds when $k = 0.3$.

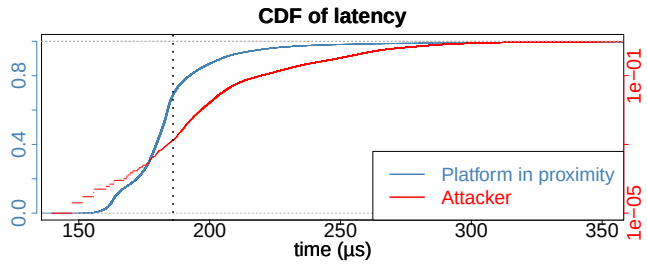

Figure 13: Cumulative distribution function for latencies. We set the threshold T_{con} at 183 μs which has a cumulative probability of 0.693 in the experiment where no rerouting attack takes place with probability of 1.33×10^{-4}.

of latency responses during an attacking scenario. It is possible to increment the latency further to improve the success probability, but doing so will start increasing the probability for the attacker as well. After that, we estimate that any latency value less than or equals to the threshold T_{con} appears with the cumulative probability of $p_{\mathcal{H}} = \Pr[144 \le x \le 186] = \sum_{i=144}^{186} \Pr[x = i] = 0.693$ (where 144 μs is the smallest latency experienced).

The attacker's success probability for a single round is the cumulative probability sampled from the attacker's distribution (the grey histogram in Figure 7) $p_{\mathcal{A}} = \Pr[x \le 186] = \sum_{i=160}^{183} \Pr[x = i] = 1.33\times10^{-4}$.

Now, for both cases (simulated attack and benign case) we can model the complete challenge-response protocol of n rounds as a Bernoulli's trial where we look for at least kn responses within 186 μs out of n. We can write this cumulative probability as a binomial distribution:

$$\Pr[x \ge nk] = \sum_{i=nk}^{n} \binom{n}{i} (p)^i (1-p)^{n-i}; \text{ where } p \in \{p_{\mathcal{H}}, p_{\mathcal{A}}\}$$

2. Choosing a suitable fraction k. The next step of the evaluation is to find a suitable fraction k based on the threshold time T_{con}. Note

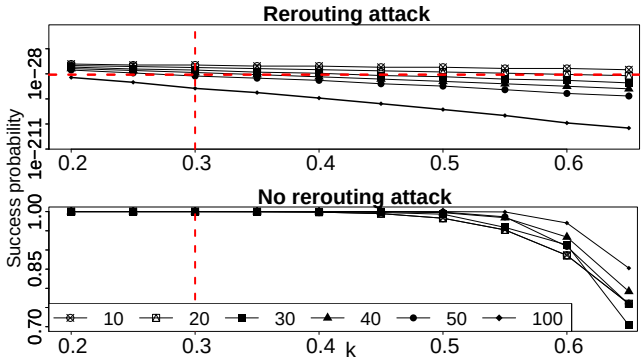

Figure 14: Finding suitable fraction k. The graph shows the legitimate enclave's success probability in an ideal scenario and the attacker's success probability in rerouting attack scenario with varying k.

that both the success probability of the attacker and the legitimate enclave is calculated as the cumulative probability from a binomial distribution (from nk to n). Hence, we require to choose a suitable value of k that maximizes P_{legit} while minimizing P_{adv}.

We calculate two graphs that are depicted in Figure 14 where the x-axis denotes k, and the y-axis denotes attacker's success probability P_{adv} and legitimate success probability P_{legit}, respectively, while using $T_{con} = 186\mu s$. We observe a sharp decrease in the legitimate success probability at $k = 0.3$. Hence, fix $k = 0.3$ to achieve the maximum P_{legit}. Additionally, in the graph of attacker's success probability, the red horizontal line is placed at $10^{-30} \approx 2^{-100}$. Hence we propose to choose any round configuration bellow this horizontal line, where $n \ge 40$. With number of rounds set to $n = 50$ and $k = 0.3$, we have $P_{legit} = 0.99999997$ and $P_{adv} = 3.55\times10^{-34}$. Similar result could be also observed in Figure 14 where the success probability of the legitimate enclave decreases significantly after $k = 0.55$ for $T_{con} = 186\mu s$.

3. Generalizing the number of rounds n. Figure 7 extends this analysis to the general number of challenge-response rounds spanning from $n = 2$ to 100. Here we compute the probability of attacker returning the reply within $186\mu s$ for at least $k = 0.3$ fraction of challenges. The y-axis denotes the attacker's success probability which diminishes overwhelmingly with the increasing number of challenges (keeping the fraction constant at $k = 0.3$).

B.2 Periodic Proximity Verification

In Section 5.6 we outlined a three-step approach for finding suitable parameters for the periodic proximity verification that we use for revocation. Here, we provide further details on each of these steps.

1. Finding suitable threshold T_{detach}. We set the threshold T_{detach} to 205 μs. We choose this value as we experience very small samples from the timing distribution (refer to the 'yellow' distribution Figure 7) where no rerouting attack takes place. While in the attacker's distribution, the cumulative probability of the response occurring between T_{con} and T_{detach} is $Pr[T_{con} \le \mathcal{L}_A \le T_{detach}] = \sum_{i=186}^{205} \Pr[\mathcal{L}_A = i] = 3.2\times10^{-2}$. Using T_{detach}, we can now define the

challenge response rounds in Figure 5 for a *single round* as following:

$$\Pr[\mathcal{L}_{legit} \leq T_{con}] = \Pr[legit \in \text{green}] = 0.693$$
$$\Pr[T_{con} < \mathcal{L}_{legit} < T_{detach}] = \Pr[legit \in \text{yellow}] = 0.208$$
$$\Pr[\mathcal{L}_{legit} \geq T_{detach}] = \Pr[legit \in \text{red}] = 1.83\times10^{-3}$$
$$\Pr[\mathcal{L}_{A} \leq T_{con}] = \Pr[A \in \text{green}] = 1.33\times10^{-3}$$
$$\Pr[T_{con} < \mathcal{L}_{A} < T_{detach}] = \Pr[A \in \text{yellow}] = 0.032$$
$$\Pr[\mathcal{L}_{A} \geq T_{detach}] = \Pr[A \in \text{red}] = 0.966$$

2. Finding suitable sliding window size w. Sliding window size is analogous to that of the number of rounds n. We keep the size of the sliding window as $w = n = 50$ as it only requires the ProximiKey to remember the past 50 interactions and achieve high probability for the legitimate enclave and negligible success probability for the attacker. Similar to the previous approach, only if 15 out of 50 ($k = 0.3$) challenge-response round where responses are within 186 μs, ProximiTEE yields success probabilities as the following:

$$\Pr[A \in \text{success window}] = P'_{adv} = P'_{fn} = 3.55\times10^{-34}$$
$$\Pr[A \in \text{failed window}] = \Pr[A \in \text{red}]^2 = 0.933$$
$$\Pr[legit \in \text{success window}] = 0.99999997$$
$$\Pr[legit \in \text{failed window}] = P'_{fp} = \Pr[legit \in \text{red}]^2 = 3.34\times10^{-6}$$

The probability that a halt window event occurs for a legitimate application-specific enclave running on the platform in proximity is $\Pr[legit \in \text{red}] \approx 7.09\times10^{-3}$. The ProximiKey halts all the data communication to the target platform until the next periodic proximity verification.

If two or more than two latencies $\geq 205\mu s$ (T_{detach}) are received, the ProximiKey terminates the connection and revoke the platform. The downtime that can happen as a result of false positive during a connection of 10 years is around 2 minutes.

3. Finding suitable frequency f. The frequency f determines how fast the connection is terminated in case the ProximiKey device is detached. Note that the ProximiKey takes around 500 μs on average to issue a new random challenge in the legitimate case. Hence, by performing a round of the protocol as soon as the previous is over, we achieve the maximum attainable average frequency of ~ 2000 rounds per second. We use this frequency as it consumes only 156.14 KB (2.4×10^{-3}% of the USB 3.0 channel capacity) and allows the communication channel to be halted on average after $200\mu s$ of the start of a relay attack and terminated in $1000\mu s$ or 1 ms.

B.3 Additional Experimental Results

Effects of core pinning. We executes the ProximiTEE enclave application pinning to specific CPU cores (using the command `taskset [COREMASK] [EXECUTABLE]`). Core pinning forces the operating system to use a specific set of CPU core(s) to execute a program. CPU pinning may significantly bring down execution time due to the elimination of core switching and ability to reuse L1 and L2 cache. Figure 15 illustrates the effect of CPU core pinning vs. no pinning. We experience negligible effect by core pinning. Hence we conclude that the attacker won't gain any advantage by CPU core pinning.

Effects of CPU load. Figure 16 shows the enclave execution times with varying degree of CPU stress testing. We used `stress-ng` to stress different number of CPU cores. We experienced a minor slowdown with the increasing number of busy CPU cores. But the slowdown is insignificant. For example, as shown in the Figure 16, we experienced a shift of $12\mu s$ when all the 8 CPU cores are busy executing the benchmark software. Also, note that the load introduced by the benchmark is a sustained load on all the CPU cores which is much more demanding for the CPUs compared to the CPU loads introduced by real-life applications. In that scenarios, the deviation would be even lesser. We conclude that proximity verification for SGX enclaves is reliable even under high system load. In rare cases of extreme system load, proximity verification might fail, but this is an availability concern, not a security threat.

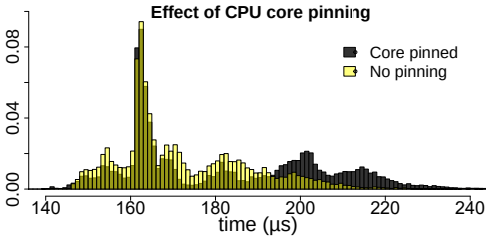

Figure 15: Effect of CPU core pinning on the enclave application. Restricting the enclave application to a specific core has a very minor effect on the observed latency.

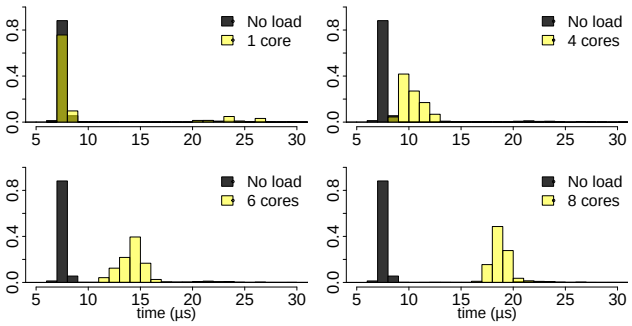

Figure 16: Effect on enclave execution time with different number of stressed CPU cores. Increase system load has a minor effect on observed enclave execution time.