Dedicated Security Chips in the Age of Secure Enclaves

Kari Kostiainen, Aritra Dhar, and Srdjan Capkun | ETH Zurich

Secure enclave architectures have become prevalent in modern CPUs. Enclaves provide a flexible way to implement various hardware-assisted security services. But special-purpose security chips can still have advantages. Interestingly, dedicated security chips can also assist enclaves and improve their security.

Trusted computing base (TCB) minimization is one of the most fundamental computer security principles. The main idea is to reduce the amount of software and hardware that needs to be trusted for the secure operation of a particular application. A common technique to achieve TCB minimization is to run the application inside a trusted execution environment (TEE). The TEE protects the application's execution despite any other compromised software on the same system.

One TEE implementation approach that has recently gained significant popularity is to realize the TEE by enhancing the main CPU of the computing platform with new features, such as special instructions and access-control checks. Intel's software guard extension (SGX), designed for the x86 architecture, is a prime example of such a TEE. In SGX, the CPU ensures that no other process can access the memory of the protected application, which is called an *enclave*. By doing this, SGX guarantees that enclaves enjoy execution integrity and that their data remains confidential. Several other TEE designs exist, too. ARM TrustZone is a popular TEE architecture that is used in many commercial mobile devices, while Sanctum[1] serves as a good example of a research TEE system. For simplicity, we focus on Intel's SGX and use it as a case study to discuss the strengths and limitations of enclaves.

SGX-style enclaves are a powerful security primitive. They are programmable, and thus developers can implement almost arbitrary hardware-protected security services by using them. This is in contrast to previous secure elements, such as trusted platform modules (TPMs), that support only a fixed set of operations. Enclaves are also fast, as they run on the main CPU of the computing platform, compared to significantly slower security elements, including smart cards. Furthermore, enclaves are cheap since they require no additional hardware, in contrast to expensive separate coprocessors, such as hardware security modules. This combination of programmability, high performance, and low cost makes enclaves an attractive way to deploy various hardware-assisted security services. Indeed, after a decade of research and development into secure enclaves, the first large-scale commercial deployments are now starting. For example, Microsoft's Azure Confidential Computing service uses SGX enclaves to protect customer data in the cloud.

Digital Object Identifier 10.1109/MSEC.2020.2990230
Date of current version: xxxxxx

The wide adoption of enclave architectures in modern CPUs is probably the most prominent trend in hardware-assisted security during the past decade. However, there is also another, more subtle trend appearing. Recently, computing service providers, such as Google, and computer manufacturers, including Apple, have started to enhance their systems with special-purpose security chips. Google's cloud servers have a security chip called *Titan* in them,[2] while Apple's computers come with the T2 security chip.[3]

At first glance, these two trends seem almost contradictory. If enclaves enable arbitrary hardware-protected security services, why do we still need dedicated security chips? In this article, we discuss the rationale behind this situation. We explain the benefits of dedicated security chips and outline two of our research projects where we designed such chips. These projects showcase an interesting new pattern, one where special-purpose security chips assist enclaves and thus improve their security.

Dedicated Security Chips

Computing platform providers have recently added new security chips to their systems. We look at two examples.

Google Titan

Titan[2] is a security chip implemented as a low-power microcontroller on Google's purpose-built server platforms. The Titan chip communicates with the main CPU via the Serial Peripheral Interface, and it interposes between the boot firmware flash and Platform Controller Hub (PCH).

One of the main functionalities that Titan implements is secure boot. When the server machine is powered up, Titan executes code, known as *boot read-only memory* (*ROM*), from its embedded ROM. This code is immutable and thus implicitly trusted. The boot ROM code loads Titan's firmware from the embedded flash and verifies its integrity using a digital signature. Once Titan's firmware is securely verified and running, it can authenticate the boot process of its host. Titan blocks the PCH's access to the firmware flash until it has cryptographically confirmed the content of the flash, and then it releases the lock and allows the verified boot firmware to configure the machine and activate the boot loader, which subsequently verifies and loads the operating system (OS). Such an iterative process enables precise control over which system software is booted.

Apple T2

Apple's latest PCs come with a security chip called *T2*[3] that also supports secure boot. When a machine with the T2 chip is turned on, T2 executes code from its ROM. This code verifies the next step of T2's own boot process. Once T2 is fully running, it can verify the Unified Extensible Firmware Interface, which will ensure that only an authorized kernel will be booted on the host CPU. Besides secure boot, T2 provides other security features, such as protecting users' fingerprint values and making sure that the microphone is disconnected from the main CPU when the laptop's lid is closed.

Specific Security Objectives

Both Titan and T2 implement secure boot. Secure boot is also a good example of a security mechanism that is outside the security objectives of SGX, which was designed to provide a specific set of protections,[4] including detecting an integrity violation of an enclave instance, safeguarding the confidentiality of an enclave's data, isolating enclaves from each other, and ensuring that an enclave's execution always starts from an authorized location. The overall goal of these protections can be loosely summarized as enabling secure computation on untrusted computing platforms. Because these safeguards do not include OS integrity verification, platform providers have added dedicated security chips, including Titan and T2, to implement such

Secure Boot in TrustZone

ARM TrustZone is a processor-based trusted execution environment architecture that is commonly used on smartphones. The main idea of TrustZone is to implement two separate execution modes on the main CPU. All untrusted software, such as the operating system (OS) and third-party apps, is executed in the *normal world,* while applications that need protection are run in a separate execution mode called the *secure world*. The processor and memory controllers ensure that any process in the normal world cannot access the secure world.

TrustZone can enable secure boot.[5] A mobile device can be configured such that when the device is powered up, the main CPU starts executing implicitly trusted code that is loaded from the ROM in the secure world. This code can then verify the normal-world boot loader before the CPU starts executing the main boot sequence of the normal-world OS. Many smartphone manufacturers implement this approach.

functionality. Disconnecting the microphone from the main CPU is another example of a security feature that is not provided by enclaves. (Other TEEs, such as the ARM TrustZone architecture, can accommodate secure boot. See "Secure Boot in TrustZone.")

Security Weaknesses

Besides limited objectives, enclaves have security weaknesses. Since enclaves and untrusted code share the same CPU, they can be susceptible to side-channel leakage and microarchitectural attacks. The recently discovered Spectre and Meltdown vulnerabilities showed how transient execution could leak information across isolation boundaries. The same idea was successfully applied to extract secret keys from SGX enclaves in the Foreshadow attack.[6] While specific attacks can be, and have been, mitigated (e.g., Intel's microcode updates include Spectre and Meltdown patches), side channels and microarchitectural attacks continue to be a concern for enclaves. The root cause is that modern processors are extremely complex systems that have been optimized through decades. Enclave support was added on top of many layers of performance optimizations, and now, in hindsight, one can easily say that this approach was not the ideal foundation for strong isolation. In this regard, dedicated security chips have a clear advantage over enclaves.

Another security challenge is the rich interface between the untrusted OS and the enclave. Enclaves must interact with the OS in many ways. For example, enclaves communicate by sending and receiving messages through the OS. Enclaves also need to safely pause their execution for interrupts. While enclave architectures provide a coarse-grained memory isolation primitive at the hardware level, developers need to ensure that the interface is protected on the software level. Common implementation tasks include sanitization of buffers and safety checks for pointers. Because such checks are tedious, several enclave runtimes, such as the Open Enclave software development kit, have been designed to assist enclave creators. However, recent research has shown that many such enclave runtimes have classical memory-safety vulnerabilities.[7] Since dedicated security chips do not need equally extensive interaction with the OS, the interface toward the untrusted OS is easier to protect tightly.

Other Security Services?

To summarize our discussion so far, enclaves do not implement all useful security mechanisms, and they also have significant security issues. Dedicated security chips can address both of these concerns. Obviously, the security services that can be implemented as dedicated security chips are not limited to the previously discussed examples. Which other services could be implemented as special-purpose security chips? Titan and T2 are integrated security chips that are permanently attached to the computing platform. Are there also use cases that would benefit from plug-and-play security tokens? In the rest of this article, we explore these questions by examining two examples from our recent research. Our first example is trusted path,[8] and after that, we focus on platform identification and remote attestation.[9]

Trusted Path

The main goal of SGX is to enable secure computation for enclaves. Such enclaves do not easily lend themselves to secure user interaction. The main reason for this is that, in an architecture such as SGX, enclaves communicate with input and output (IO) devices through the untrusted OS. When an enclave needs to receive user input, the OS must pass data from an input device, such as a keyboard, to the enclave. When an enclave creates user output, the OS must forward data to an output device, such as the display. Such a TEE design means that a compromised OS can easily modify any user inputs and outputs. Indeed, trusted path—a secure channel from the human user to a trusted application, such as the enclave—is outside the security objectives of SGX.[4]

User input manipulation can have severe consequences. If a malicious OS modifies the user input that is provided to a financial enclave, the enclave can be tricked to perform unauthorized payments. If enclaves are used to implement hardened medical devices and industrial systems, user input modifications may cause serious safety and health risks. Also, any enclave that needs user passwords or similar credentials is difficult to implement securely.

Design Principles

Several projects have explored the idea of complementing computing platforms with dedicated hardware modules for building a trusted path. While proposing extra hardware may be easy, the more difficult and interesting question is what exactly should the added hardware do? To answer this question, we look at previous approaches, examine their limitations, and identify design principles for trusted path implementation.

The first approach that we look at is transaction confirmation.[10] In such a solution, the user first completes the interaction, such as a payment, by interacting with the user interface (UI) of the untrusted platform that may be manipulated by the OS or browser. Once this is done, the user is expected to confirm his or her input, such as a payment value and account number, through a separate hardware token to detect and prevent any possible modifications. This approach has two main problems. The first is the fact that the user now has to interact with two separate devices and two UIs, which reduces usability. The second and more severe problem is that the extra confirmation step is vulnerable to user habituation. *Habituation* refers to behavior where the user begins, after a few successful transactions, confirming new payments without verifying their correctness. These observations lead us to the first design principle.

Principle 1. Out-of-context security confirmations have a high cognitive load and risk of user habituation. Thus, user interaction should be protected in the context of the normal UI.

The next approach that we examine is input signing.[11] Here, the main idea is to use a simple hardware device that sits between a user input device, such as a keyboard, and the untrusted computing platform. The device intercepts every keypress, or similar user input event, and sends a signed trace of each one to the trusted application. Input signing conforms to our first principle because the user does not have to interact outside the main UI. However, such solutions are vulnerable to attacks, where the adversary manipulates the user's input by showing false information on the output channel. That is, output manipulation leads to user input integrity violations. One example is a fake-typo attack. Assume that the user types in the value "10," but the adversary shows "1" on the screen. The user is likely to think he or she mistyped and may press "0" again. As a result, a signed trace of "100" will be sent to the trusted application. This simple attack leads us to our next design principle.

Principle 2. User input and output integrity cannot be considered in isolation. Both must be protected simultaneously.

The last approach that we examine is secure overlays. Fidelius[12] is an example research system that follows this approach. In Fidelius, one device intercepts keyboard presses and signs them for the trusted application, while another device intercepts the High-Definition Multimedia Interface (HDMI) output signal and modifies it with secure overlays of security-critical UIs, such as payment web forms. Fidelius addresses the previously mentioned problems but is still vulnerable to a different type of user input manipulation that we call *early submission attack*. This attack is possible because Fidelius protects input from a keyboard but not a mouse. While this may seem to be only a functional limitation, it turns out to be a security problem. Assume again that the user intends to submit the value "10." Once the user has typed "1," the untrusted OS or browser generates a fake mouse event that submits the web form. Mouse inputs could be disabled altogether, but such a naive solution hurts usability. Now we can state our last design principle.

Principle 3. All user input modalities must to be protected simultaneously.

ProtectIOn System

Given these principles, we designed a new trusted-path system called *ProtectIOn*.[8] In the following, we focus on the use case where the trusted path is established between the user and a remote web server. That is, we want to protect interactions where the user completes and submits a security-critical web form. The same solution could be easily modified to create a trusted path between the user and a local enclave, as well.

Figure 1 provides an overview of the ProtectIOn system. The central component of the solution is a low-complexity embedded device called *IOHub* that intercepts keypresses from a keyboard and movement events from a mouse, tracks mouse movements, and draws secure overlays. When the user visits a webpage that contains a protected web form, the remote server sends a QR code to the untrusted browser, as shown in Figure 2(a). The QR code contains a specification of the protected web form signed by the server to prevent its modification in the browser. By using QR codes, we enable communication from the webserver to the IOHub device via an unmodified browser, which significantly simplifies deployment. By periodically examining the HDMI signal, IOHub detects the QR code on the screen, decodes it, and verifies its signature. After that, as illustrated in Figure 2(b), IOHub renders the protected web form as an overlay on top of the HDMI ~~frame~~ that it receives from the untrusted OS. This step ensures that the security-critical UI elements are correctly presented to the user, and thus the output integrity of the protected web form is preserved.

IOHub tracks mouse movement events, and when the mouse pointer enters the secure overlay, as pictured in Figure 2(c), IOHub dims the rest of the screen to focus the user's attention on the secure overlay. Such protection is needed to prevent the user from following a possible fake mouse cursor, drawn by the untrusted browser or OS, elsewhere on the screen. Dimming parts

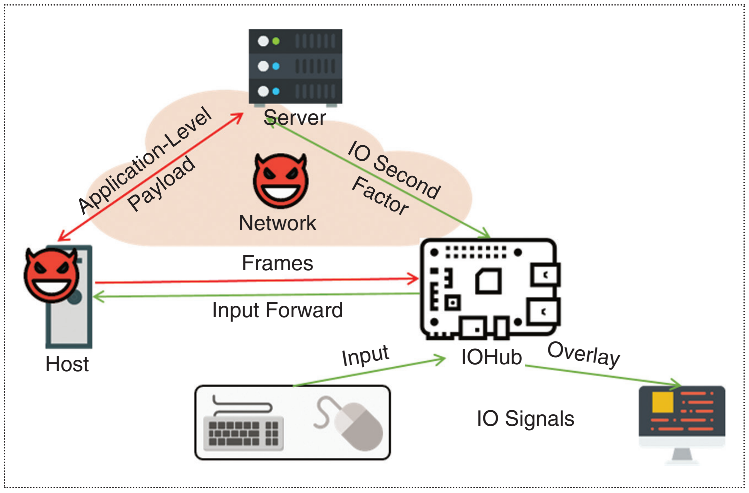

Figure 1. The ProtecIOn system.

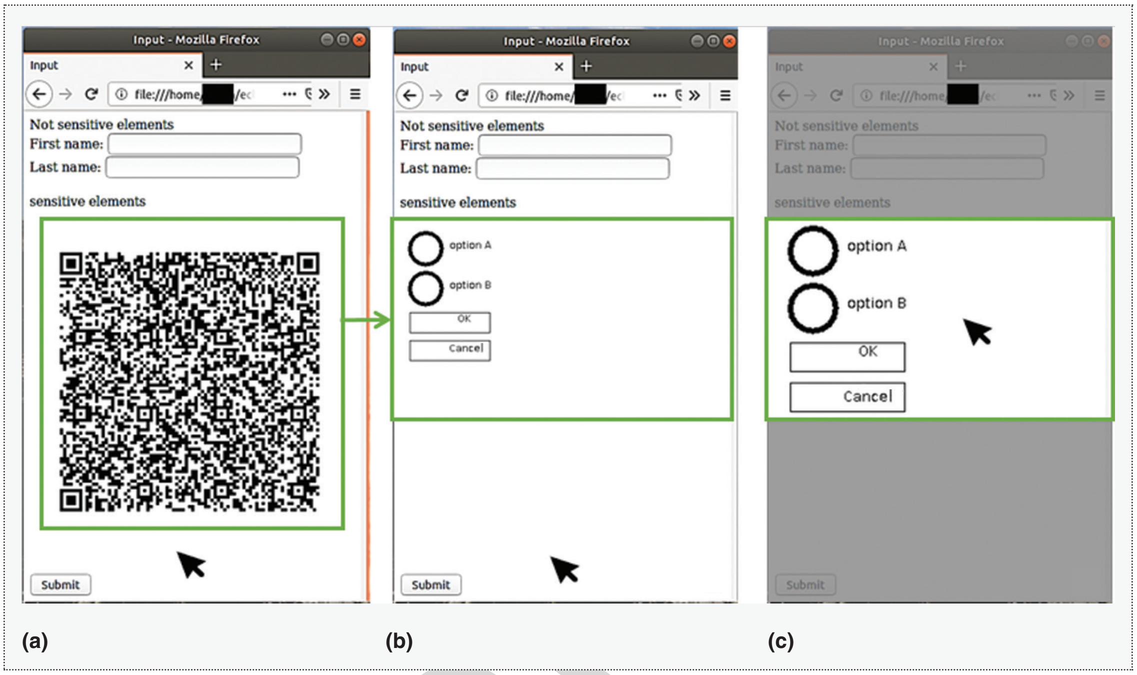

Figure 2. The ProtectIOn UI, including (a) the attacker's view, (b) the user's view on the monitor, and (c) focusing the user's attention.

of the screen has been shown to be an effective way to focus the user's attention to the correct cursor.[13]

While the user interacts with the protected web form, IOHub intercepts all input events, and when the user clicks on the "submit" button, IOHub signs all inputs and sends them to the server via the untrusted browser (e.g., by encoding them to a requested URL). Since the "submit" button is part of the protected overlay and all mouse clicks are intercepted and signed by the IOHub device, early submission attacks are not possible. The server verifies the signed user inputs it receives. If input confidentiality is needed, the user can be required to trigger a secure attention sequence, such as Control-Alt-Delete, before entering any secrets. The untrusted OS and the browser cannot observe sensitive data on the secure overlay since the information is rendered by IOHub and never accessible to them. Such a design complies with our three principles. Principle 1 is met because the user interacts only with the main UI. Secure overlays support Principle 2, and mouse tracking combined with input trace signing ensures that all input modalities are protected (Principle 3).

ProtectIOn Prototype

We implemented a prototype of the IOHub device as a combination of Raspberry Pi and Arduino boards and a simple HDMI interceptor. Our prototype shows that the proposed functionalities are feasible to implement with a small TCB on low-cost hardware. Full details of the ProtectIOn system are explained in our recent paper.[8]

Attestation and Platform Identification

Remote attestation is a key feature of enclave architectures such as SGX. In remote attestation, an external verifier checks that an enclave was constructed as expected. To do this, the verifier sends a challenge to the attested CPU. The CPU signs the challenge together with a previously recorded measurement of the enclave's code by using a processor-specific attestation key. The signature can also include application-specific data, such as the public key of the enclave, that enables secure communication with the attested enclave. The attestation key is part of a group signature scheme managed by Intel. The signed attestation statement is then sent back to the verifier. If the signature can be authenticated correctly, the remote verifier knows that the attested enclave was correctly constructed and runs the expected code inside a legitimate SGX processor. The remote attestation process enables distant verifiers to detect enclave integrity violations before provisioning secrets to, or accepting signed messages from, the enclaves.

Relay Attacks

The previously outlined remote attestation protocol is a useful security mechanism, but it also has a well-known problem. An adversary that controls the OS on the attested platform can easily redirect the attestation challenge to another platform that computes the response.

The verifier cannot notice such relay attacks since the SGX attestation mechanism is based on a group signature scheme, and all processors from the same group produce indistinguishable signatures. (Even if attestation used traditional digital signatures, it would be very difficult, in practice, for the verifier to know which signing key corresponds to which CPU.) In other words, SGX guarantees detection of an enclave's integrity violation, but identification of the computing platform on which the attested enclave is running is not part of its security objectives.

Relay attacks have been known for a long time. Parno identified them in the context of TPM attestation more than a decade ago and called them *cuckoo attacks*.[14] However, the full implications of relay attacks have not been well understood. Since attestation can be redirected only to another legitimate processor that executes exactly the same attested enclave code, it may appear that such relay attacks do not have noteworthy negative implications. Our analysis shows that such a belief is misguided.

Relay attack implications. Many computer systems, such as servers at data centers, use multiple layers of protection. These safeguards may include using TEEs to protect certain applications and other defenses, such as running software components at different privilege levels, physically protecting access to the computing platform, and frequent patching of discovered vulnerabilities. The main implication of relay attacks is that they increase the adversary's ability to attack the attested enclave by circumventing many such protections.

Our first observation is that by redirecting the attestation to the adversary's platform, the enemy enables physical side-channel attacks, such as acoustic, electric, and electromagnetic monitoring, that otherwise would not be possible to mount on the victim's platform. Such side channels have been shown to be effective and inexpensive means to extract secrets,[15] and hardening enclaves against all possible physical side channels is difficult. Relay attacks can also enable privilege escalation. In cases where the adversary has compromised only the user space application that manages the enclave on the victim platform, the application can redirect the attestation to the attacker's platform, where the opponent controls the OS, as well. In such cases, the relay enables digital side-channel attacks that require system privileges.

The third and perhaps most subtle implication of relay is that it can enable software-based side-channel attacks that otherwise would not be possible due to the timing of certain events. One example is a scenario where the victim platform OS is compromised at the time of attestation and secret provisioning, and the attested enclave is hardened against known digital side-channel attacks (e.g., using tools like Raccoon[16]). After secret provisioning, the OS compromise is detected, and the platform is cleaned and patched. Later, a new side-channel attack vector (that is not prevented by the used tools) is discovered. If the adversary performed redirection during attestation and the secret was provisioned to the attacker's machine, the new ~~side channel~~ is exploitable. Without the relay, the attack is not possible. In this case, the attestation relay eliminates the security benefit of good platform maintenance. Finally, we note that attacks based on leaked attestation keys (e.g., ones obtained through the Foreshadow attack[6]) are independent of relaying. If the adversary has obtained a valid attestation key, he or she can emulate an SGX processor on the target platform and steal any secrets that are provisioned to it.

Trust on first use. A commonly suggested solution to relay attacks is the principle of trust on first use (TOFU). In one example solution, a platform-specific enclave generates a key pair and exports the public part of the key for certification right after a fresh OS installation. When remote verifiers need to attest to other enclaves on that platform, they can first authenticate the certified enclave, which, in turn, performs local attestation of the target enclave (SGX provides a local attestation primitive that cannot be relayed). Such a TOFU solution has several problems. The first is a large temporary TCB, as a complete general-purpose OS needs to be trusted during the first boot. The second is difficulty in deployment because a fresh OS reinstallation is not always possible. The third is the need for online certification authorities, which increases their attack surface, assuming that the certification process is automated. Further problems are discussed in Dhar et al.[9]

ProximiTEE System

Parno identified relay attacks more than a decade ago and, at the same time, suggested proximity verification as a solution.[14] The main idea was to verify that the attested CPU was in the proximity of the verifier device, which should prevent attestation redirection to remote platforms. Proximity verification using an external device overcomes the previously listed main problems of TOFU approaches. First, the TCB is small, as the verifier device can be a single-purpose one and thus very simple. Second, the adoption of such a solution is easy since there is no need to fully reset the target platform. And third, the authority that certifies the verifier device can remain entirely offline.

Although the idea of proximity verification sounds simple to realize, the research efforts that followed failed to implement secure proximity verification. The main reason for the failure was that TPM, the secure element commonly available at the time, supported only fixed identification operations, such as digital signatures. TPM signatures may take up to one second, which enables the redirected attestation request to travel a long distance, and thus TPMs were simply too slow for secure proximity verification.

Because SGX enclaves are programmable, it is possible to implement proximity verification protocols that leverage simple operations, such as exclusive-OR, that enable fast challenge-response rounds. Based on this observation, we designed a hardened SGX attestation scheme called *ProximiTEE*.[9] Our solution uses a simple embedded device called *ProximiKey* that is attached to the target platform through a local communication interface, such as a USB. In hardened remote attestation, the verifier first establishes a secure channel, a ProximiKey, whose public key the verifier learns from its issuer. ProximiKey performs standard remote attestation on the local enclave, establishes a secure Transport Layer Security channel to the enclave, and verifies its proximity by using a simple distance-bounding protocol that consists of repeated and fast challenge-response rounds. If such proximity verification succeeds, ProximiKey facilitates the creation of a secure channel between the remote verifier and attested enclave.

Proximity verification security. While the preceding design is mostly straightforward, the more interesting aspect is whether such a system prevents relay attacks, in practice. To answer this question, we implemented our solution using a USB prototyping board and simulated a strong attack scenario where the adversary performs a relay to another SGX platform that is connected to the target platform through a 1-m-long Ethernet wire. We also assumed that the adversary was able to perform all protocol computation instantaneously. Figure 3 shows the results from our experiments, where we measured both the legitimate and relayed challenge-response latencies. The vast majority of the benign latencies range from 145 to 250 μs, while the attack round trips take from 200 to 750 μs. The average delay of our adversary is only 80 μs. (To put this in perspective, even the highly optimized network connections between major data centers in the same region exhibit latencies from 1 ms upward.)

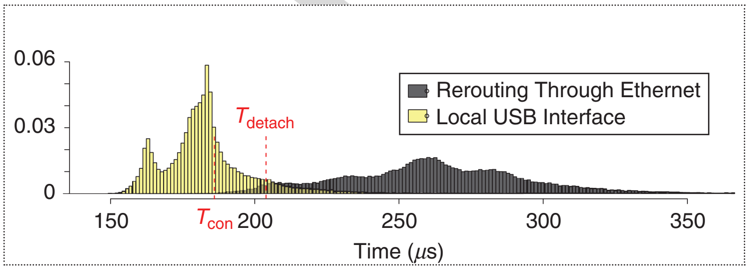

Figure 3. The ProximiKey latency distributions for legitimate challenge-response rounds and a simulated relay attack.

As can be seen from Figure 3, these two latency distributions are distinguishable. Our analysis confirms that it is possible to set protocol parameters (the number of challenge-response rounds, latency threshold, and so on) such that very fast relay attacks can be detected with high probability, and legitimate attestations fail only with a negligible likelihood. The full details of the ProximiTEE system and our analysis can be found in our recent paper.[9]

Discussion

Now that we have seen two commercial integrated security chips (Titan and T2) and two user-attachable research tokens (ProtectIOn and ProximiTEE), we can discuss their deployment aspects, security benefits, and previous comparable solutions.

Deployment Options

Integrated security chips, such as Titan and T2 are, obviously, limited to deployments by major service and platform providers that have the ability to design and build their own systems. ProximiTEE is an example of a plug-and-play security token that can be attached to the target platform through a standard interface, such as a USB. Deployment of security solutions is a feasible option for a larger set of service providers. For example, cloud-computing providers can enhance off-the-shelf servers with ProximiKey tokens and communicate their public keys to their clients to support more secure attestation. Another interesting use case is the setup of a permissioned blockchain, where every consensus node is hardened with SGX enclaves. The trusted authority that appoints the consensus nodes can issue a ProximiKey token to each organization that operates one node.

Also, ProtectIOn could be deployed as a plug-and-play security module. Such deployment enables service providers, such as voting authorities and banks, to increase the security of their services without restricting users' choice of client platform. In medical and industrial domains, an externally attached ProtectIOn module can improve the security of safety-critical systems, even when modifications to the computing platform itself are prohibited due to strict regulations. Alternatively, ProtectIOn could be deployed as an integrated security chip such that its functionality is implemented as part of the integrated keyboard, mouse, and display controllers.

Security Benefits

Titan and T2 are chips that enable security functionality, such as a secure boot, that is not provided by enclaves.

Thus, the operation of Titan and T2 is largely orthogonal to the operation of enclaves. In comparison, ProximiTEE is a security chip that is designed to work in collaboration with enclaves and improve their security guarantees by enabling secure TEE identification for hardened remote attestation. ProtectIOn is a solution that can either assist enclaves or operate independently of them. One possible usage for ProtectIOn is to enable a trusted path from the user to a local enclave, which can communicate securely with remote servers. In such a deployment, ProtectIOn works together with an enclave. Alternatively, ProtectIOn could be employed to create a trusted path from the user to a remote server without the use of enclaves. Such a deployment is beneficial when the risk of microarchitectural attacks on enclaves is considered too high, for example.

Dedicated Chips and Early TEEs

Some previous TEE designs have leveraged dedicated security chips to implement a TEE that can be attested. AMD's Secure Virtual Machine technology is one such example, where the TEE is created as follows: the CPU measure code in a specific memory region enables direct memory access protections for that region, disables interrupts, records the measurement into a platform configuration register of a TPM chip, and, finally, begins executing the measured code. Essentially, this sequence of events provides a clean execution of the measured code without restarting the whole platform, and therefore this technique is often called *late launch*. The main role of the dedicated chip (a TPM, in this case) is to securely record the code that was launched so that an external verifier can check its integrity by using attestation.

Future computing platforms are likely to combine various units, including CPUs, GPUs, tensor processing units, field-programmable gate arrays (FPGAs), and more. Similar to the current enclave architectures that enhance CPUs with secure execution capabilities, other processing units, such as GPUs and FPGAs, will need secure computation, too. There are already several ongoing research efforts that explore the design of such TEEs.[17]

Our research on trusted path (the ProtectIOn system[8]) highlights that IO devices need secure communication with enclaves. Similarly, other peripherals, such as GPS units and fingerprint sensors, would benefit from secure communication with enclaves. Protected communication between TEEs and other platform components requires authentication, enclave identification, and access-control mechanisms. The ARM TrustZone architecture has limited support toward this direction. In TrustZone, hardware components, such as memory controllers, can make coarse-grained access-control decisions based on the CPU's execution mode.[5] Extending this paradigm for more fine-grained access control and secure intercomponent communication is one promising direction. We envision future computing platforms where enclaves, peripherals, and special-purpose security chips can communicate and work together to provide a rich set of hardware-assisted platform security services. ■

References

1. V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. 25th USENIX Security Symp.*, 2016, pp. 857–874. doi: 10.5555/3241094.3241161.
2. "Titan in depth: Security in plaintext," Google, Mountain View, CA. Accessed on: Mar. 2020. [Online]. Available: https://cloud.google.com/blog/products/gcp/titan-in-depth-security-in-plaintext
3. "About the Apple T2 security chip," Apple Inc., Palo Alto, CA. Accessed on: Mar. 2020. [Online]. Available: https://support.apple.com/en-us/HT208862
4. F. McKeen et al., "Innovative instructions and software model for isolated execution," in *Proc. 2nd Int. Workshop Hardware and Architectural Support Security and Privacy*, 2013. doi: 10.1145/2487726.2488368.
5. J.-E. Ekberg, K. Kostiainen, and N. Asokan, "The untapped potential of trusted execution environments on mobile devices," *IEEE Security Privacy*, vol. 12, no. 4, pp. 29–37, July–Aug. 2014. doi: 10.1109/MSP.2014.38.
6. J. Van Bulck et al., "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proc. 27th USENIX Conf. Security Symp.*, 2018, pp. 991–1008. doi: 10.5555/3277203.3277277.
7. J. Van Bulck et al., "A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes," in *Proc. 2019 ACM SIGSAC Conf. Computer and Communications Security*, pp. 1741–1758. doi: 10.5555/3277203.3277277.
8. A. Dhar, E. Ulqinaku, K. Kostiainen, and S. Capkun, "ProtectIOn: Root-of-trust for IO in compromised platforms," in *Proc. Network and Distributed System Security Symp.*, 2020. doi: 10.14722/ndss.2020.24112.
9. A. Dhar, I. Puddu, K. Kostiainen, and S. Čapkun, "ProximiTEE: Hardened SGX attestation by proximity verification," in *Proc. 10th ACM Conf. Data and Application Security and Privacy*, 2020, pp. 5–16. doi: 10.1145/3374664.3375726.
10. A. Filyanov, J. M. McCuney, A. Sadeghiz, and M. Winandy, "Uni-directional trusted path: Transaction confirmation on just one device," in *Proc. 2011 IEEE/IFIP 41st Int. Conf. Dependable Systems & Networks*, pp. 1–12. doi: 10.1109/DSN.2011.5958202.
11. A. Dhar, D-Y. Yu, K. Kostiainen, and S. Capkun, "Integrikey: End-to-end integrity protection of user

input," 2017. [Online]. Available: https://eprint.iacr.org/2017/1245

12. S. Eskandarian et al., "Fidelius: Protecting user secrets from compromised browsers," in *Proc. IEEE Symp. Security and Privacy*, 2019, pp. 264–280. doi: 10.1109/SP.2019.00036.
13. L.-S. Huang, A. Moshchuk, H. Wang, S. Schechter, and C. Jackson, "Clickjacking: Attacks and defenses," in *Proc. 21st USENIX Conf. Security Symp.*, 2012, pp. 413–428. doi: 10.5555/2362793.2362815.
14. B. Parno, "Bootstrapping trust in a trusted platform," in *Proc. 3rd Conf. Hot Topics Security*, 2008, pp. 1–6. doi: 10.5555/1496671.1496680.
15. D. Genkin, L. Pachmanov, I. Pipman, A. Shamir, and E. Tromer, "Physical key extraction attacks on PCs," *Commun. ACM*, vol. 59, no. 6, pp. 70–79, 2016. doi: 10.1145/2851486.
16. A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital side-channels through obfuscated execution," in *Proc. 24th USENIX Conf. Security Symp.*, pp. 431–446. doi: 10.5555/2831143.2831171.
17. S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on GPUs," in *Proc. 12th USENIX Conf. Operating Systems Design and Implementation*, 2018, pp. 681–696. doi: 10.5555/3291168.3291219.

Kari Kostiainen is a senior scientist at ETH Zurich. His research interests include trusted computing, mobile security, secure user interaction, and blockchain security. Kostiainen received a Ph.D. from Aalto University, Espoo, Finland. Contact him at kari.kostiainen@inf.ethz.ch.

Aritra Dhar is a Ph.D. student at ETH Zurich. His research interests include secure user interaction, trusted computing, anonymous networks, and program analysis. Dhar received an M.S. from the Indraprastha Institute of Information Technology, Delhi, India. Contact him at aritra.dhar@inf.ethz.ch.

Srdjan Capkun is a full professor at ETH Zurich. His research interests include wireless security, secure positioning, trusted computing, and blockchain technologies. Capkun received a Ph.D. from École Polytechnique Fédérale de Lausanne, Switzerland. He is a Fellow of ACM. Contact him at srdjan.capkun@inf.ethz.ch.

> The TEE protects the application's execution despite any other compromised software on the same system.

> If enclaves enable arbitrary hardware-protected security services, why do we still need dedicated security chips?

> Enclaves do not implement all useful security mechanisms, and they also have significant security issues.

> Out-of-context security confirmations have a high cognitive load and risk of user habituation.

“

User input and output integrity cannot be considered in isolation.

”

“

Remote attestation is a key feature of enclave architectures such as SGX.

”