DeviceVeil: Robust Authentication for Individual USB Devices Using Physical Unclonable Functions

Kuniyasu Suzaki*, Yohei Hori*, Kazukuni Kobara*, Mohammad Mannan†
*National Institute of Advanced Industrial Science and Technology, Japan
†Concordia University, Canada

***Abstract*—The Universal Serial Bus (USB) supports a diverse and wide-ranging set of device types. To enable ease of use, USB devices are automatically detected and classified by common operating systems, without any authentication. This *trust-by-default* design principle can be easily exploited, and led to numerous attacks in the past (e.g., Stuxnet, BadUSB, BadAndroid), specifically targeting high-value organizations. Administrators' efforts to prevent these attacks may also be threatened by unscrupulous users who may insert *any* USB device, or malicious users (inside attackers) who may try to circumvent OS/kernel-enforced protection mechanisms (e.g., via OS replacement). The root causes of USB attacks appear to be the lack of robust authentication of individual USB devices and inadequate tamper-proofing of the solution mechanism itself. We propose `DeviceVeil` to address these limitations. To authenticate individual USB devices, we utilize the tamper-proof feature of Physical Unclonable Functions (PUFs); PUFs extract unique features from physical characteristics of an integrated circuit (IC) at a reasonable cost (less than 1 USD). To make our authentication mechanism robust, we implement it as a small hypervisor, and protect it by a novel combination of security technologies available in commodity PCs, e.g., Trusted Platform Module (TPM), customized secure boot, and virtualization support. The OS disk image with all user data is encrypted by a key *sealed* in TPM and can be decrypted by the hypervisor only. Customized secure boot allows the loading of the legitimate hypervisor and OS kernel only. The hypervisor enables pre-OS authentication to protect the trust-by-default OS from USB attacks. The chain of trust continues from power-on to the insertion of a USB device and disallows all illegitimate USB devices. DeviceVeil's PUF authentication takes about 1.7 seconds during device insertion.**

***Index Terms*—Individual Device Authentication, Hypervisor, Physical Unclonable Function, Secure Boot, Trusted Computing**

I. Introduction

Current USB standards (2.0 or 3.0) support several sophisticated mechanisms but follow the *trust-by-default* [69] design principle, connecting many types of devices with heterogeneous protocols, including, Human Interface Devices (HID), Mass Storage Class (MSC), Media Transfer Protocol (MTP) and Picture Transfer Protocol (PTP). The ubiquity and diversity of USB have enabled many high-profile attacks. For example, Stuxnet [9], [32] relied on a USB storage device, and went beyond network air-gap to compromise Iranian SCADA/nuclear systems. BadUSB [43] camouflaged USB storage as a USB keyboard and inserted malicious commands.

USB attacks are also perpetrated by insiders. The most famous insider attack that uses USB is possibly the Edward Snowden incident in 2013 [80]. Another example is, Benesse, a Japanese education company, which announced the leak of 29 million customer records by a company engineer in 2014 [25]. Benesse apparently had a prevention mechanism enforced by Windows ActiveDirectory against illegal use of Mass Storage Class USB devices. However, the engineer allegedly used MTP/PTP of his smartphone to bypass the protection, highlighting the difficulty of designing protection mechanisms that can withstand malicious insiders. In addition, an insider can replace a hardened OS/kernel enforcing the protection with a backup tool, and then restore the system after connecting an illegal USB device (e.g., to copy sensitive files). The insider can also load an unauthorized kernel with the Linux *kexec* syscall [46], and thus can bypass OS-bound defenses including secure boot. However, complete blocking of USB devices (e.g., by physical means) is a non-solution, as many machines routinely connect peripheral devices by USB alone, e.g., a USB keyboard and mouse, although these devices can be easily exploited, as evident from BadUSB [43].

Despite these high-profile attacks, security awareness regarding USB devices is still low. CompTIA [10] reported that 17% of the dropped unknown USB storage devices were plugged into PCs in several US business districts. Tischer et al. [70] dropped 297 USB storage devices in a university campus (UIUC), and found that HTML files in 45% of the devices were opened. These careless and accidental insertions also must be addressed, especially, in an enterprise solution.

To counter these attacks, the USB 3.0 Type-C specification includes PKI-based authentication with tamper-proof hardware [72]; each product would hold a certificate issued by the USB Implementers Forum or an intermediate CA to protect counterfeit. The specification states that "Products should provide protected tamper-resistant operation and storage for the private keys to prevent them from being read." Nevertheless, tamper-resistant mechanisms remain unspecified. Another drawback of the specification is that the proposed authentication is effective only for each *product type*, but not for individual devices. Most access control mechanisms also support product-level only (e.g., ActiveDirectory and SELinux). Solutions that support individual device authentication, mostly rely on USB serial numbers (e.g., USBSec [78] and GoodUSB [66]). However, USB serial numbers can be easily modified to defeat such weak authentication; e.g., see the attack [44] on USB Raptor [74], which uses USB serial numbers to lock/unlock Windows. Therefore, it is clearly evident that a tamper-proof hardware mechanism is required to implement any robust device authentication.

Such robust device authentication technologies have been available for smart cards for more than a decade now [1], [37]. For example, Subscriber Identity Module (SIM) cards for mobile phones use Secure Element [52] as a tamper-proof technology for individual device authentication. The specification of Secure Element is openly defined by GlobalPlatform [19], and the chip is available from several secure semiconductor vendors (e.g., Gemalto, NXP). Unfortunately, these device authentication mechanisms based on hardware tamper-proof technologies are relatively costly and accessible to some vendors for specific devices (e.g., mobile phones), and IT administrators cannot modify SE to use in commodity PCs.

Software tamper-proof technologies can also be used to protect a device authentication solution. For example, Intel SGX (Software Guard Extensions) offers software tamper-proofing; however, SGX cannot handle intermediate device recognition between a USB device and the OS because SGX offers Ring-3 isolated execution environment named ”enclave” and does not have direct access to devices. Virtualization can also be used to implement tamper-proof hypervisors, e.g., HyperSentry [4], HyperSafe [79], TrustVisor [38] and SecVisor [53]. Unfortunately, they are not designed to protect OS replacement attacks, such as the Linux *kexec* syscall [46] from the userland by insiders.

We propose `DeviceVeil` to authenticate individual USB devices using Physical Unclonable Functions (PUFs) with a hardened thin hypervisor. PUFs do not require additional tamper-proofing, and extract uniqueness from physical characteristics of integrated circuits. The setting of PUFs can be managed by enterprise administrators and a challenge-response authentication pair can be reset even if the pair is lost/compromised. The thin hypervisor is strengthened by security technologies in commodity PCs, including Trusted Platform Module (TPM), secure boot, and virtualization. Our novel combination of these techniques compensates shortcomings of individual components. The disk image of the OS is encrypted by a key *sealed* in the TPM chip, and the key is extracted by the hypervisor depending only on the trusted boot technology. However, trusted boot cannot prevent the re-installation attack, which can be mitigated by secure boot by allowing the legitimate kernel only. The hypervisor enables pre-OS authentication in an isolated environment and protects the existing OS that follows the *trust-by-default* [69] design principle of USB authentication. DeviceVeil is primarily targeted for enterprise environments, where a trusted administrator can configure employee machines and initialize all allowed USB devices.

Contributions and Challenges:

1) *Individual USB device authentication:* DeviceVeil is the first solution that authenticates individual USB devices by PUF, and thus connecting USB device authentication to physical characteristics of the USB IC. The use of PUF provides robust hardware-based identification against tampering, e.g., unlike device serial number, which can be easily duplicated, or the use of per-device unique secret keys, which can be extracted. In addition, PUF derives different IDs from different sampling points, and the setting of PUF authentication is controlled by the administrators in an enterprise environment (i.e., the USB device manufacturers are not involved in the authentication process). One challenge for PUF implementation is the cost of an individual device. We estimate the hardware cost of our PUF design is less than 1 USD (apparently reasonable even for low-cost USB devices).
2) *Tamper-proofing against inside attackers:* DeviceVeil introduces tamper-proofing into both the USB device and the software components on a client PC. We utilize a novel combination of TPM, secure boot, and virtualization technology—compensating each other's shortcomings. DeviceVeil encrypts the entire OS including user data, and the encryption key is *sealed* in the TPM chip, i.e., can be extracted from the TPM when the Platform Configuration Register (PCR) values are extended with the correct hash of the hypervisor only. Our customized secure boot allows loading of the hypervisor and the OS kernel only. The hypervisor includes a pre-OS authentication mechanism and protects the existing OS. Thus, malicious users cannot reinstall another OS, remove the DeviceVeil hypervisor, or access content from the OS through unauthenticated USB devices.
3) *Vendor/OS independence:* DeviceVeil uses the vendor-independent tamper-proof technology, PUFs, allowing enterprise administrators to configure challenge-response pairs. In addition, DeviceVeil hypervisor runs only one commodity OS and protects it from unauthenticated USB devices. Authentication is enforced within the hypervisor before the OS can interact with the actual USB device. DeviceVeil also utilizes readily-available security technologies in commodity PCs, aiding deployability.
4) *Implementation:* We have implemented a prototype of DeviceVeil by reusing and modifying the code from several projects, including BitVisor [56] (for the thin hypervisor), DeviceDisEnabler [62] (parts of TPM management), and Zuiho [87] (USB-PUF authentication). The overhead comes from PUF authentication during device insertion, an average of 1.7 seconds, and zero performance penalty during the device use.

II. Attack Types and Threat Model

In this section, we summarize common attack types against USB and provide our threat model.

A. Attack types

When a USB device is inserted, the OS recognizes the USB device based on its descriptor (e.g., vendor ID, class ID, serial number), and loads a suitable device driver automatically. This *trust-by-default* [69] design principle facilitates most attack types exploiting USB.

BadUSB [43] utilizes a vulnerability in USB firmware and customizes it to camouflage as a USB keyboard, i.e., identifying the device class as a Human Interface Device (HID). The camouflaged keyboard inputs malicious commands. Another similar attack is USBProxy [59] that uses the USB On-The-Go specification [73] to mimic another (allowed) device type. USB On-The-Go enables a USB device to switch the roles of a host and device (for the case of a one-to-one connection). USBProxy exploits this functionality to enable MITM attacks.

The diverse set of protocols as allowed in USB standards, are also leveraged in some attacks. As mentioned in Section I, MTP/PTP has allegedly been used to bypass the protection for MSC at Benesse [25]. Furthermore, some smartphones offer "USB Debug Mode", not protected by ActiveDirectory [77]. BadAndroid [5] spoofs a USB-Ethernet adapter from an Android phone to capture the traffic from a connected computer.

All USB devices typically include a device descriptor to advertise their features; the descriptor offers information such as the vendor ID, product ID, class ID, serial number, etc. The USB specifications do not require a mandatory, hard-coded serial number, and thus some products use the same serial number, or no serial number at all, e.g., USB keyboard/mouse. Most USB access control tools like ActiveDirectory usually do not consider the USB serial number because their purpose is to allow/disallow certain product types (based on device descriptor); note that ActiveDirectory can be configured to use serial numbers.

B. *Threat model and assumptions*

DeviceVeil has two primary goals: disallow any unauthorized USB device, and protect the OS kernel and DeviceVeil itself from being tampered. We protect DeviceVeil against different types of inside attackers, including the following. (a) Attackers with admin privileges of the OS (not the system administrator who sets up DeviceVeil), may use the kexec command to load a separate Linux kernel without restrictions (e.g., no active defense against USB attacks). Kexec is a kernel function to load another kernel from the running kernel without the help of BIOS/UEFI and bootloader, which can also be used as a jailbreak mechanism [47]. From kernel version 3.17, kexec has an option for allowing only kernels with verifiable digital signatures. (b) Insiders may also reinstall the OS on the target PC. They can recover the previous/legitimate OS if they keep the disk image (to avoid detection). Backup tools can also help such attacks by allowing the seamless/instant restore feature. (c) Attackers may also try to install a hypervisor that allows direct access to USB devices through I/O virtualization (e.g., Intel VT-d) to a guest OS. The host OS/hypervisor does not interfere with such direct access.

Security assumptions for DeviceVeil include the following.

1) The adversary can have physical control of the machine after the secure setup of DeviceVeil. We assume that administrators deploy PCs with secure BIOS, keep the BIOS firmware updated, and protect the BIOS using strong passwords that are not shared with a regular user (see also Section VIII).
2) The adversary has root/admin privileges of the OS, but he cannot replace the hypervisor/kernel as they are protected by secure boot (only signed updates issued by the administrator are allowed).
3) The adversary can have physical control of USB devices, including legitimate USB devices with PUF. He can exploit the USB controller and modify the USB device descriptor (i.e., device class and serial number) as in BadUSB. However, we assume PUF is unmodified and cannot be moved to another USB device (i.e., PUF is integrated into the USB's IC).
4) DeviceVeil does not control the content in USB devices, and as such cannot protect the host OS against USB malware in legitimate USB devices. Also, DeviceVeil cannot prevent information leakage through an authorized USB device. Confidentiality and integrity of contents must be protected by other means (e.g., encryption or access control).
5) The adversary can change TPM configuration (e.g., reset ownership via jumper settings), but cannot launch attacks that require significant lab efforts (chip imaging/decapping). Note that TPM reset will not leak the TPM-sealed disk encryption key (unsealed only with specific PCR values). We also assume the TPM firmware is also kept up-to-date (cf. [23]).

III. Current Countermeasures

Several countermeasures for USB attacks have been proposed and implemented in academic literature and commercial solutions. We categorize them as USB access control, USB authentication, and tamper-proofing, and discuss a few examples closely related to our work. For a comprehensive analysis of all major defensive solutions, see Tian et al. [69].

A. *USB access control*

ActiveDirectory and SELinux are the most popular access control mechanisms on Windows and Linux, respectively. ActiveDirectory has a group policy for USB devices and allows access control for each product or device class, but cannot distinguish individual devices. SELinux also offers similar access control in Linux. Solutions that support individual device authentication, mostly rely on USB serial numbers (e.g., USBSec [78] and GoodUSB [66]). However, USB serial numbers can be easily modified, and thus such weak authentication can be defeated (see e.g., [44]).

USBFILTER [68] and Cinch [2] introduce packet filtering based security mechanisms for USB communication. Packet filtering can allow fine-grain access control and prevent attacks. USBFILTER is implemented as a reference monitor and is statically compiled and linked into the Linux kernel to avoid being unloaded; it uses a database of rules for USB packets for each application, and filters the applications accordingly. Cinch also utilizes packet filtering, but it uses two KVM virtual machines, where the first VM is sacrificial while the other one runs a guest OS. The sacrificial VM uses I/O virtualization and connects to USB devices directly. There is an *enforcer* between the sacrificial VM and the guest OS, and packets are filtered and encrypted. However, as packet filtering involves checking all communications to/from a USB device, significant overhead is incurred; e.g., USBFILTER suffered 17.6% overhead when 100MB data was transferred, and Cinch reduced the I/O throughput from 3.4Gbps (direct) to 2.1Gbps for USB-SSD.

SandUSB [34] and USBWall [27] use a small computer between a USB device and a host PC, which monitors the USB's behaviors. SandUSB uses a Raspberry Pi2 with a module to relay USB packets from a USB device to the PC. SandUSB scans and analyzes USB packets using USB-Mon [83]. USBWall uses a BeagleBone Black (BBB) and runs USBProxy [59] to act as middleware to enumerate the devices on behalf of the PC. USBWall has no mechanism to filter USB protocols, and simply works as an isolated environment.

B. USB authentication

Most USB device authentication targets device class identification, instead of individual devices (as in USB 3.0 Type-C specifications). Some solutions use the serial number of a USB device but the serial number is easily modified. Kanguru's FlashTrust [28] introduces a digitally signed firmware to protect against BadUSB, but it is not designed for individual authentication. GoodUSB [66] offers device authentication mediated by a human user. The user makes "the final decision" from the physical form of the USB device, i.e., GoodUSB implicitly trusts its users, and thus cannot prevent inside attackers or unscrupulous users.

Kells [7] and ProvUSB [67] offer host authentication based on a TPM to limit USB devices to connect only to legitimate host computers. They require a special USB device with computation capability (e.g., an ARM CPU). The device also includes public key issued by the TPM on the host machine. If the authentication (i.e., the public key verification) fails, the device exposes dummy flash storage. The deployment cost may be non-negligible due to the need for an on-board CPU per USB device.

C. Tamper-proofing

Robust and tamper-proof authentication is essential to prevent insider attacks, but most countermeasures are installed only in a kernel or hypervisor, which can be bypassed by several attack vectors such as kernel replacement via kexec, or installation of an unprotected hypervisor/OS. On the other hand, many current PCs are equipped with UEFI secure boot [81] that prevents the loading of unsigned/unauthorized bootloaders or kernels. For example, Microsoft Windows Production CA allows booting only signed Windows bootloader and kernel. DeviceVeil utilizes the secure boot mechanism and TPM sealing to enable tamper-proofing, which resembles Windows BitLocker to some extent.

IV. Background

As discussed in Section I in the explanation of USB 3.0 Type-C specification and Secure Element of SIM card, a hardware-based tamper-proof mechanism is needed to implement robust individual device authentication. In addition, tamper-proofing and OS independence are required. These requirements are fulfilled by PUF and a novel combination of security technologies in commodity PCs, i.e., TPM, secure boot, and virtualization.

A. PUF: Physical Unclonable Function

PUFs use the delay of the electric signal or the initial state of memory, which differs on each chip due to small variations resulting from the manufacturing process. These features allow deriving unique secrets from individual physical characteristics of ICs. When the sampling points are changed, PUFs create a different ID from the device. PUFs utilize hardware intrinsic features and do not require any special mechanism for hiding secret keys in tamper-proof hardware. PUFs are used for DRM, key generation, and device authentication (see e.g., [16], [30], [33], [36], [49], [61], [63]).

1) Fuzzy extractor: PUFs use fragile physical characteristics of ICs and have an intrinsic limitation. Since PUFs are sensitive to noise and always produce bit errors in response, a correction technique is required. In 2004, Dodis et al. [13] proposed circuits, known as a *fuzzy extractor*, to correct bit errors by extracting uniform random bits. The fuzzy extractor is implemented in PUF authentication and creates error correction data known as *helper data* [6]. The helper data is passed to the circuits when the key is used for authentication. While the redundant information calculated during verification may include noise, the original redundant information (calculated at the initialization, and used at the verification phase as helper data) is noiseless. The noise in redundant information in the verification phase is corrected with the helper data.

2) PUF device authentication: The key created by PUF and fuzzy extractor can be used for authentication [16], [61]. Note that, pure PUF circuits are passive and lack memory, and some PUF authentication circuits include an additional cipher mechanism [30], [36], [49]. Some current PUF authentication mechanisms are based on arbitrary string encryption/decryption and include the circuits implementing the ciphers (see later sections for DeviceVeil).

B. TPM and Trusted Boot

Trusted Platform Module (TPM) is a secure chip available on many commodity PCs. DeviceVeil utilizes platform integrity and sealing/unsealing of TPM.

1) Platform integrity: The measurement of platform integrity is the core concept of trusted boot consisting of multiple phases: measuring and extending cryptographic hash values of hardware and software components. When the system is powered on, the immutable bootstrap code (CRTM: Core Root of Trust for Measurement) measures the hash value (e.g., through SHA-1 in TPM 1.2) of the BIOS and *extends* it in the PCR (Platform Configuration Register) of TPM before transferring control. The TCG-BIOS also measures the hash values of peripheral devices, option ROMs, and the bootloader, and extends them in the same manner. The same method is implemented in the bootloader and kernel, and thus enables a *chain of trust*.

2) Sealing/Unsealing: Sealing/unsealing of TPM is limited by the measured PCR values to load a key in a TPM. Windows BitLocker uses this mechanism to limit access to the key when PCRs show the values which are extended at a legitimate booting (i.e., the boot procedures and devices remain original). In order to offer a FDE (Full Disk Encryption) to Windows, BitLocker separates the storage into the non-encrypted and encrypted partitions, as the non-encrypted partition is used by the bootloader before the secret key is unsealed from the TPM. Sealing/unsealing of TPM is also used for other applications to confirm the correctness of the machine, boot sequence and BIOS/UEFI.

C. Secure Boot

Although trusted boot can measure platform integrity, it cannot prevent kexec and reinstallation attacks (even if the disk remains encrypted). UEFI secure boot [81] can be used to prevent booting unauthorized kernels (e.g., not signed by a trusted authority). Secure boot requires two keys and two databases. The Platform Key (PK), typically set by the manufacturer, is the encryption key for the Key Exchange Key (KEK). KEK is the encryption key for Authorized DB (db) and Unauthorized DB (dbx), which are databases for public-keys issued by a CA, and include hashes for authorized and unauthorized modules (e.g., bootloaders, hypervisors or kernels). Unfortunately, most PCs do not allow customizing the keys and databases of secure boot. As a workaround, many Linux distributions use a pre-bootloader signed by Microsoft (e.g., Fedora's shim.elf [55]).

D. Virtualization Technology (Hypervisor)

Most secure hypervisors function as the trusted computing base (TCB) to protect a guest OS. Some solutions leverage trusted computing, and guarantee the chain of trust starting from power-on. For example, HyperSentry [4] and Hyper-Safe [79] protect the hypervisor itself, and TrustVisor [38] and SecVisor [53] protect the hypervisor, guest kernel and applications. However, they are not secure against reinstallation and kexec attacks. Unauthorized loading kexec kernels can be avoided by disallowing the kexec syscall in the allowed/signed kernel, and reinstallation attacks can be prevented by secure boot. However, secure boot is not used by common hypervisors, possibly because most PCs do not allow changing the keys and databases in the UEFI. Fortunately, a few PCs (e.g., Lenovo ThinkPad T460s) allow customizing secure boot, and DeviceVeil's prototype takes advantage of such flexibility.

V. DESIGN OF DEVICEVEIL

DeviceVeil authenticates individual USB devices using PUFs and offers robustness against inside attackers. Figure 1 shows DeviceVeil's components. DeviceVeil unseals a key from the TPM and decrypts the encrypted disk to boot the OS. The DB for USB device detection and PUF authentication is also decrypted by the key in the TPM. The hypervisor and OS kernel are authenticated by the secure boot.

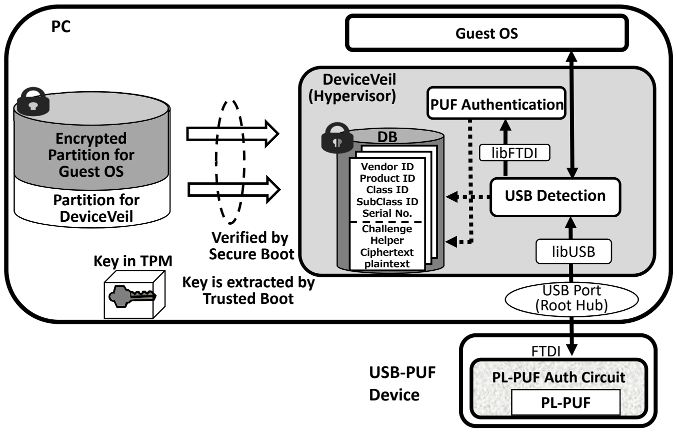

Fig. 1. DeviceVeil design overview.

DeviceVeil authenticates a legitimate USB device with PUF, and then exposes it to the OS, as pre-OS authentication. The pre-OS authentication can be implemented in two options.

1) Hosting the cryptographic key: the key created by PUF is stored in a verifier on a host PC that is used to create ciphertext from plaintext on demand. The drawback is that the key is disclosed if the verifier is compromised.
2) Database for plaintext and ciphertext: The verifier does not have the cryptographic key created by PUF, but it has a database of plaintext and ciphertext pairs created by the key. This prevents key disclosure but requires the database to be created in advance.

We use option (2) to avoid storing a long term key, and to restrict the misuse of a challenge-repose pair exposed via e.g., USB bus sniffing (each pair is used once). Also, if the challenge-repose pairs are different on each PC, the compromise of a PC with its database has limited impact on the device's authentication (e.g., the device can be safely used in other PCs). If the hypervisor selects option (1) hosting cryptographic/PUF key, it must select a different encryption key, generated by PUF circuits of a device. The different encryption key generates different ciphertext at each time on each PC and can limit USB bus sniffing attacks. However, keys generated by PUFs for all PCs need to be tested for correctness because PUFs are sensitive to noise.

A. Pseudo-LFSR PUF (PL-PUF)

DeviceVeil customizes Pseudo-LFSR PUF (PL-PUF [24]) to avoid shortcomings of current PUF technologies. PL-PUF is a delay-based PUF but outputs multiple and variable responses. The structure is based on the Linear Feedback Shift Register (LFSR). PL-PUF composes larger combinational logic than normal delay-based PUFs, and it efficiently outputs multiple bits in parallel used for variable ID. However, the size of the PL-PUF authentication circuit is reasonably small compared to memory-based PUFs. Furthermore, the challenge-response mapping of PL-PUF is variable, depending on the active duration of the circuit, i.e., a single PL-PUF behaves as if it has multiple PUF cores.

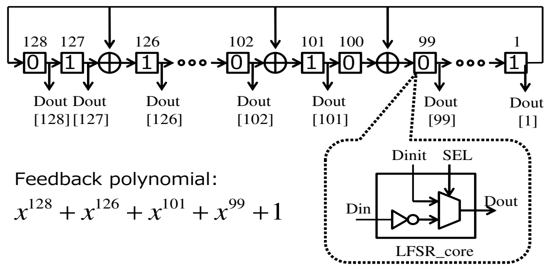

Fig. 2. The structure of PL-PUF authentication circuit (128 bits). It consists of 128 inverters and 3 XOR gates and creates a feedback polynomial.

Figure 2 shows the structure of our 128-bit PL-PUF with the primitive feedback polynomial. The core logic is not a register but an inverter, and thus our PL-PUF is composed of a single combinational circuit. The output will oscillate since the output of the last core (*Dout(1)*) is fed back to the top-most core. The output value depends on the speed of the feedback signal, which is significantly affected by the device variation. As a result, the output is device-dependent. PL-PUF realizes a challenge-response pair as shown in Figure 2. The challenge is a 128-bit initial value given to the core logic (*Dinit*), and the response is a 128-bit output from the core logic (*Dout*) after a certain time of feedback.

PL-PUF creates a multiple bit response that is prone to burst errors. DeviceVeil uses the Reed-Solomon (RS) error correction for the fuzzy extractor to deal with burst errors. For authentication, DeviceVeil uses a temporal database of plaintext and ciphertext pairs generated by using AES (the AES key is created by PL-PUF to avoid the leakage of the key). The plaintext and ciphertext pairs are created with the AES key in advance by the administrator for each individual device. Also, each challenge-response pair should be used only once to prevent USB bus sniffing attacks.

B. Hypervisor with USB-PUF authentication

DeviceVeil is designed on top of BitVisor [56], a type-1 thin hypervisor. BitVisor is based on a para-pass-through architecture and does not prevent hardware interrupts from the USB ports to the OS. Therefore, DeviceVeil disguises a dummy device to the OS until the USB-PUF authentication is completed in the isolated hypervisor environment. The isolated environment also hosts a database with device records used for USB device detection and challenge-response pairs (i.e., ciphertext and plaintext pairs) used for PUF authentication. The database is also encrypted by a TPM-sealed key, and an inside attacker (e.g., the user of the machine) cannot access this key; the key and the plaintext (decrypted) database are available only to DeviceVeil's hypervisor, and inaccessible from the OS.

We enforce PUF authentication by default. To support non-PUF USB devices as an exception, an administrator can configure DeviceVeil to use only vendor ID, product ID, class ID, and serial number. This weak authentication can co-exist during the transition, but we strongly advise against relying on it.

C. Hypervisor with TPM and Secure Boot

To avoid bypassing DeviceVeil's protection, we encrypt the partition of the OS and seal the encryption key in TPM NVRAM, accessible only to DeviceVeil (loaded via trusted boot). Thus, the OS cannot boot without loading the unmodified DeviceVeil binary. Note that although the encryption key stays in RAM, only the hypervisor has access to it (i.e., making it unavailable to the OS, or OS-resident memory extraction tools).

Even if trusted boot measures platform integrity and the disk is encrypted by a TPM-protected key, a reinstallation attack is still possible (i.e., kexec). DeviceVeil compensates this problem with UEFI secure boot, by disallowing unauthorized kernels. An administrator can customize a secure boot to specify legitimate kernels with no kexec support (allowing updates for properly signed kernels). However, the hypervisor must be updated with the active participation of the administrator (to set up the trusted boot to extract/reseal the encryption key in TPM). We assume our thin hypervisor will require updates very infrequently (unlike an OS kernel). Note that, the possibility of an I/O virtualization attack is restricted by the fact that DeviceVeil is loaded first, and it occupies CPU virtualization, and thus other kernels cannot override it.

D. Deployability

DeviceVeil mostly uses security mechanisms included in many commodity PCs (e.g., TPM, secure boot, and virtualization) to help deployability. Although not all consumer PCs come with these components, but enterprise IT admins can buy PCs with appropriate support. In addition, DeviceVeil requires new hardware "PUF" (similar to several other smart USB solutions such as [7], [67]). For the novel PUF-authentication mechanism, we limit the PUF cost to be reasonable (approx. under 1 USD), to include a broad set of USB devices.

The cost of maintenance depends on the usage model. To allow administrators to remotely manage the allowed list of devices (i.e., DeviceVeil's protected database) on a client PC, DeviceVeil must mandate strong authentication of admin machines/accounts (e.g., via remote attestation). This extra effort is necessary for scalability if devices are added/removed frequently. If the authorized devices are changed infrequently, the simple per-PC/device deployment may also be reasonable. Our prototype implementation follows this usage scenario. Note that current DeviceVeil allows users to update the kernel with a proper signature, and thus makes updates easy to deploy.

VI. Implementation

DeviceVeil's hypervisor is implemented on BitVisor [56] with DeviceDisEnabler [62] (parts of TPM management), and USB-PUF authentication is implemented on Zuiho [87] which is a platform for PUF security evaluation. Newly added and deleted code for DeviceVeil are about 6,000 and 500 LOC respectively. The main part of the added code is PL-PUF authentication. Note that, DeviceVeil's TCB also contain libFTDI (unmodified: 55,000 LOC).

TABLE I
RESOURCES USED BY PL-PUF CIRCUITS. RESOURCES ARE SLICE OF FPGA, LOOK UP TABLE (LUT), FLIP FLOP (FF), BLOCK RAM (BRAM) AND OPERATION TIME (μSEC).

	Slice	LUT	FF	BRAM	μsec
PL-PUF	130	260	256	0	0.33
RS Enc	181	333	236	2	1.37
RS Dec	7,990	15,086	2,323	1	23.17
AES	2,774	5,405	812	0	0.58(Enc)/1.08(Dec)
Total	12,774	21,459	6,282	7	25.44(Enc)/25.94(Dec)

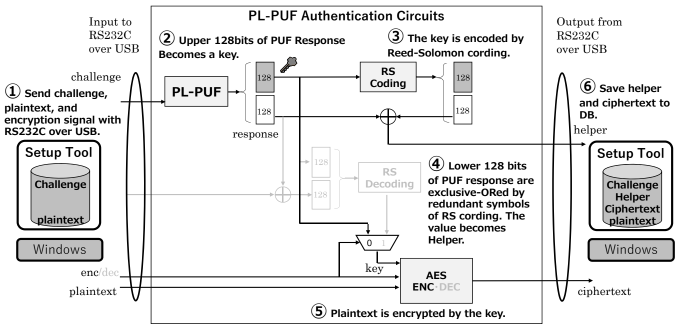
PL-PUF Authentication Circuits
Input to RS232C over USB
Output from RS232C over USB
② Upper 128bits of PUF Response Becomes a key.
③ The key is encoded by Reed-Solomon cording.
challenge
① Send challenge, plaintext, and encryption signal with RS232C over USB.
PL-PUF
128
128
response
RS Coding
128
128
⑥ Save helper and ciphertext to DB.
helper
Setup Tool
Challenge
plaintext
Setup Tool
Challenge
Helper
Ciphertext
plaintext
RS Decoding
④ Lower 128 bits of PUF response are exclusive-ORed by redundant symbols of RS cording. The value becomes Helper.
0
1
Windows
Windows
key
enc/dec
plaintext
AES
ENC·DEC
ciphertext
⑤ Plaintext is encrypted by the key.

Fig. 3. Setup Tool (Enrollment) and PL-PUF authentication circuits. Light gray circuits are not used by enrollment.

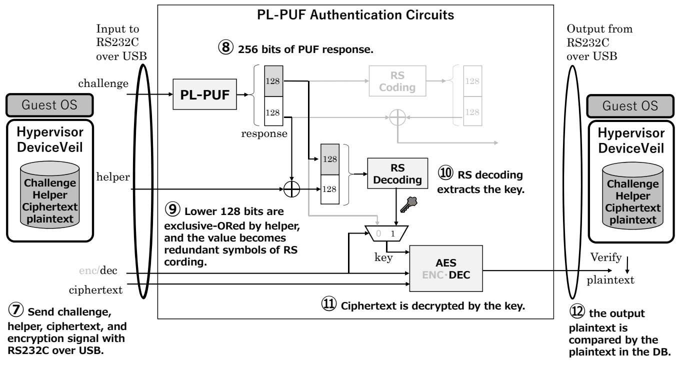
PL-PUF Authentication Circuits
Input to RS232C over USB
Output from RS232C over USB
⑧ 256 bits of PUF response.
challenge
PL-PUF
128
128
response
RS Coding
128
128
Guest OS
Hypervisor DeviceVeil
Challenge
Helper
Ciphertext
plaintext
Guest OS
Hypervisor DeviceVeil
Challenge
Helper
Ciphertext
plaintext
helper
128
128
RS Decoding
⑩ RS decoding extracts the key.
⑨ Lower 128 bits are exclusive-ORed by helper, and the value becomes redundant symbols of RS cording.
0
1
key
AES
ENC·DEC
enc/dec
ciphertext
Verify
plaintext
⑪ Ciphertext is decrypted by the key.
⑦ Send challenge, helper, ciphertext, and encryption signal with RS232C over USB.
⑫ the output plaintext is compared by the plaintext in the DB.

Fig. 4. DeviceVeil authentication (verification of the hypervisor) and PL-PUF authentication circuits. Light gray circuits are not used in authentication.

A. Zuiho PL-PUF Authentication Circuits

The Zuiho board comes with an FPGA chip, Spartan-3A DPS3400A, and our PL-PUF authentication circuits are implemented on it. The circuits consist of PL-PUF, RS encoding, RS decoding, AES encryption/decryption, etc. The circuits for PL-PUF are based on [24], and the RS encoding and decoding utilize MATLAB Communication Toolbox and HDL Coder. For AES encryption/decryption, we utilize the open source Verilog code offered by the Cryptographic Hardware Project [3]. In Table I, we list the hardware resources used by PL-PUF authentication circuits; Logic Block resources include: Slice, Look Up Table (LUT), Flip Flop (FF), and Block RAM (BRAM).

1) Cost of PUF: To evaluate the cost, the size of PL-PUF authentication circuits is compared to the TPM implementation on FPGA [14], [15]. Eisenbarth et al. [15] assume 3,000 Logic Elements (each element consists of a single 4-input LUT connected to a single-bit flip-flop) and 75k RAM. Eguro and Venkatesan [14] assume 27,237 LUTs, 27,076 FFs, and 49 BRAMs. These circuit sizes of TPM implementations are significantly larger than the current implementation of PL-PUF authentication; see Table I. Furthermore, the current implementation of RS encoding/decoding uses the MATLAB Communication Toolbox and HDL Coder, which are not optimized. We can reduce these parts with custom code. We estimate the price of PL-PUF based on the size of the circuit is less than 1 US dollar because the price of TPM is estimated to be 1 US dollar [31] (from 2006). Furthermore, PL-PUF will be integrated into the circuits in a USB IC and does not require additional tamper-proof mechanism as a TPM chip; i.e., the cost will be even lower. Thus, price-wise, PL-PUF should not be an issue for most USB devices. Especially, DeviceVeil is assumed to be used in enterprise environments, which can absorb the extra cost for security gain.

2) Setup and Authentication: PL-PUF authentication circuits are set up by a tool we call *PUF-Acquisition*, which works as a database enrollment tool on Windows. The database is used by PUF authentication in the hypervisor. The setup tool and hypervisor use USB-UART interface to communicate with the PUF. The procedures of the Setup Tool (Enrollment) and DeviceVeil authentication (hypervisor's verifier) are explained below. Note that we assume that suitable challenge data is selected to generate a robust key.

Setup Tool (Enrollment): Figure 3 outlines the PL-PUF database generation procedure using the setup tool. It runs on Windows and communicates with the PL-PUF authentication circuits using RS232C protocol over FTDI-USB serial. As prerequisite data, the setup tool has challenge data for PL-PUF and plaintext for encryption/decryption by the key created by the PL-PUF. The steps are as follows.

1) The setup tool sends a challenge (256 bits), plaintext, and encryption signal to the PL-PUF authentication circuits with RS232C protocol over USB.
2) The challenge goes to PL-PUF and becomes a 256-bit response. The upper half (128 bits) of the response is used as the encryption key.
3) The key is encoded by Reed-Solomon error correction code, and 128 redundant bits are generated (16 symbols).
4) The redundant RS symbols are masked (XORed) with the lower half (128 bits) of the PUF response so that the redundant symbols do not leak any secret. The masked value is used as helper data, which is outputted to the setup tool.
5) Inputted plaintext is encrypted by the encryption key. The ciphertext is outputted to the setup tool.
6) The setup tool saves the helper and ciphertext values to the DB for PUF authentication.

DeviceVeil Authentication (Verification of the Hypervisor): Figure 4 outlines the PL-PUF authentication procedure. The DeviceVeil hypervisor sends a challenge, helper, and ciphertext

TABLE II
THE INPUT DATA STRUCTURE FOR DEVICEVEIL'S DB, USED FOR USB DEVICE DETECTION AND PUF AUTHENTICATION. CATEGORIES OF VERIFICATION, FLAGS, AND OPTIONS ARE SHOWN.

Category	Flag (1 byte)	Options when the flag is on.
Vendor ID	1=on,0=off	Vendor ID 2 bytes
Product ID	1=on,0=off	Product ID 2 bytes
Class ID	1=on,0=off	Class ID 2 bytes
Sub-Class ID	1=on,0=off	Sub-Class ID 2 bytes
Serial Number	1=on,0=off	Serial Number (Unicode) 18 bytes
PUF Info	1=on,0=off	Challenge 32 bytes Challenge #2 16 bytes Helper 16 bytes Ciphertext 16 bytes Plaintext 16 bytes

to the Zuiho board, and verifies the received plaintext. The steps are as follows.

7) DeviceVeil sends challenge, helper, and ciphertext to the PL-PUF authentication circuits in the USB device.
8) The challenge goes to PL-PUF and becomes a 256-bit response.
9) The helper data is XORed with the lower 128 bits of the PUF response, resulting in the unmasked redundant symbols (with several errors) of the RS code.
10) The upper 128 bits of the response and redundant symbols go through RS decoding and generate the encryption key.
11) The ciphertext is decrypted by the encryption key.
12) The output plaintext is compared to the plaintext in the DB for PUF authentication. When the comparison succeeds, DeviceVeil exposes the actual USB device to the OS; when failed, the USB device remains hidden.

3) Creating DB for USB Device Detection and PUF Authentication: We summarize the input data structure used by our pre-OS USB device detection and PUF Authentication in Table II. Device meta-data is stored in the DB during setup (Enrollment), encrypted by a TPM-stored key accessible only by the hypervisor (see Figure 1). We use the stored device information (e.g., vendor ID, product ID, class ID, and seal number, provided by the $Get\_descriptor()$ function) in the DB for pre-OS device detection. We get the PUF challenge-response pairs from the Setup Tool (Enrollment) described in Section VI-A and store them in the DB for PUF authentication.

B. Device Detection by Hypervisor

DeviceVeil is based on the para-pass-through architecture of BitVisor and does not prevent hardware interrupts to the OS when a USB device is inserted, but DeviceVeil responds with a dummy USB device during USB-PUF authentication.

1) Communication Structure: We add a USB device detection process to BitVisor's libUSB compatible interface and use libFTDI as the FTDI driver. When a USB device is detected, DeviceVeil plays the role of a USB host, and the PUF authentication process communicates with the PL-PUF circuits with the RS232C protocol on USB-FTDI.

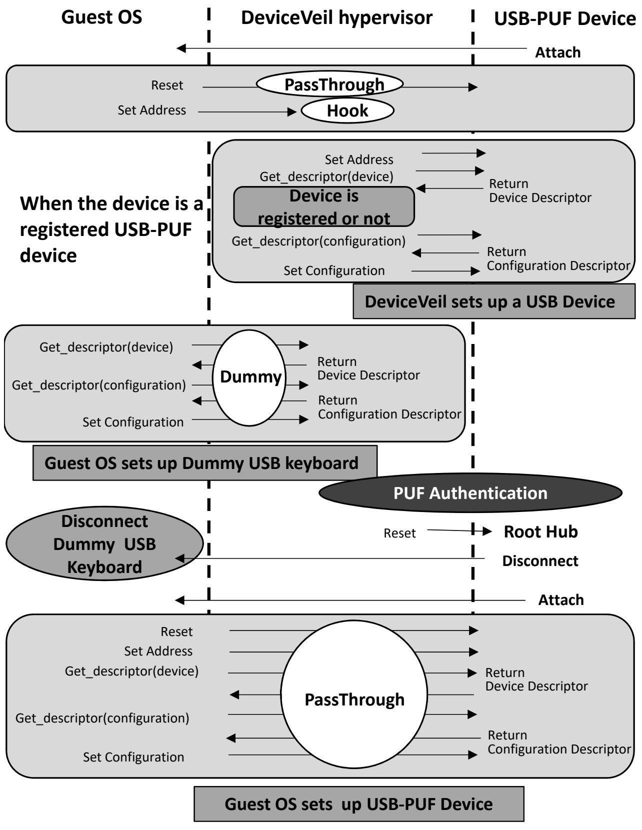
Guest OS
DeviceVeil hypervisor
USB-PUF Device
Attach
Reset
PassThrough
Set Address
Hook
Set Address
Get_descriptor(device)
Return Device Descriptor
Device is registered or not
When the device is a registered USB-PUF device
Get_descriptor(configuration)
Return Configuration Descriptor
Set Configuration
DeviceVeil sets up a USB Device
Get_descriptor(device)
Dummy
Return Device Descriptor
Get_descriptor(configuration)
Return Configuration Descriptor
Set Configuration
Guest OS sets up Dummy USB keyboard
PUF Authentication
Disconnect Dummy USB Keyboard
Reset
Root Hub
Disconnect
Attach
Reset
Set Address
Get_descriptor(device)
PassThrough
Return Device Descriptor
Get_descriptor(configuration)
Return Configuration Descriptor
Set Configuration
Guest OS sets up USB-PUF Device

Fig. 5. Pre-OS USB device detection on DeviceVeil (between the Guest OS and a USB-PUF device)

2) Procedure of Pre-OS USB Detection: Figure 5 shows the procedure for pre-OS USB device detection by DeviceVeil hypervisor. When a USB device is inserted, the "Attach" hardware interrupt occurs. Due to BitVisor's para-pass-through architecture, the OS is informed about the insertion, and DeviceVeil disguises a dummy device until PUF authentication is completed. After the "Attach" interrupt, the OS follows the normal procedure to detect a USB device. The OS waits a certain period (about 100 msec) and issues the "Reset" command. Then the USB device comes to the "default" state and can accept control commands. After that, the "Set Address" command is issued from the OS to the USB device; however, this request is overtaken by DeviceVeil, which issues its own "Set Address" command to the USB device. The USB device then comes into the "address" state.

DeviceVeil sends $Get\_descriptor(device)$ to the device to get device descriptor (e.g., vendor ID, product ID, class ID, serial number). If the device is a registered USB-PUF device, DeviceVeil offers a dummy harmless USB keyboard to the OS and attempts the PUF authentication process. If the device is a registered non USB-PUF device (if explicitly allowed by administrators during the transition), DeviceVeil allows para-pass-through access from the OS immediately. Otherwise, DeviceVeil just offers a dummy device to the OS.

For legitimate PUF-enabled devices, DeviceVeil allows the OS to continue the USB device recognition procedure. The OS sends $Get\_descriptor(device)$ and

TABLE III
COMPARISON BETWEEN DEVICEVEIL AND RELATED PROPOSALS

	USB 3.0 Type-C [72]	Active Directory	GoodUSB [66]	Cinch [2]	USBFILTER [68]	Kells [7]	ProvUSB [67]	**DeviceVeil**
Individual Device Auth.	No	No	Serial Number	No	No	No	No	PUF
Additional Hardware	Tamper-proof storage in USB (Spec-only)	No	No	No	No	ARM in USB	ARM in USB	PUF in USB
Granularity	Device	Device	Device	Packet	Packet	Device	Device	Device
Overhead	Insert time	Insert time	Insert time	Comm. time	Comm. time	Insert time	Insert time	Insert time
Isolation	No	No	No	Yes QEMU/KVM	No	No	No	Yes BitVisor
OS dependence	No	Yes	Yes	No	Yes	Yes	Yes	No
Tamper-proof and hardening	(Good on USB) (Spec-only) Tamper-proof Storage	(Partial on host) Privilege mode	(Partial on host) In Kernel	(Partial on host) In Hypervisor	(Good on host) Trusted Boot SELinux	(Good on host) TPM Auth.	(Good on host) TPM Auth.	(Good on host&USB) Trusted Boot Secure Boot PUF

$Get\_descriptor(configuration)$, but DeviceVeil hooks them and responds with a dummy USB device. After the OS recognizes a dummy USB device, DeviceVeil initiates the PL-PUF authentication process (Section VI-A). If authentication fails, no other action is taken (the OS only sees the dummy USB keyboard that processes no input). If authentication succeeds, DeviceVeil issues the "Reset" command to the USB root hub of the PC. The root hub raises the "Disconnected" interrupt, which goes to the OS. The OS recognizes that the dummy USB device is removed and takes action for disconnection. Afterward, the USB device issues the "Attach" interrupt automatically, which is sent to the OS to establish a USB connection normally. From this time, DeviceVeil forwards all requests as para-pass-through.

If authentication fails, DeviceVeil does not take any further action. The OS keeps the dummy USB keyboard, which processes no input. When the real USB device is disconnected, the dummy USB keyboard is also removed in the manner of USB disconnection.

C. Customizing Trusted/Secure Boot

The OS disk partition is separated from the disk partition for booting DeviceVeil. The OS partition is encrypted by DeviceVeil using the encryption key sealed in the TPM (i.e., unsealed only when the PCR values indicate that unmodified trusted boot and DeviceVeil have been loaded). The current implementation is designed for TPM 1.2 on a ThinkPad T400 (Intel Core2 P8400 2.26GHz) and uses TCG-BIOS and TrouSerS [71]. We use the *TPM_NV_DefineSpace* command to allocate the NVRAM space for the encryption key, bounded by PCR[0-7] (TCG-BIOS) and PCR[12-14] (Trusted GRUB) values. The *TPM_NN_WriteValue* command is used to write the encryption key to NVRAM with these PCR values during the setup stage. The *TPM_NV_ReadValue* command is used to read the encryption key. The read operation is successful only when the bounded PCR values are the same as the registered values. After that, DeviceVeil decrypts the OS disk partition and boots the OS.

To customize secure boot for DeviceVeil and the OS, the Platform Key, Key Exchange Key, Authorized DB, and Unauthorized DB must be replaceable. We use ThinkPad T460s (Intel i5-6300U 2.4GHz) as ThinkPad T400 does not offer secure boot. On the other hand, ThinkPad T460s does not have TPM 1.2. We thus have to use both these machines for testing our prototype (which of course can be integrated into a single, compatible PC). The UEFI of T460s allows customizing the Platform Key, Key Exchange Key, Authorized DB, and Unauthorized DB. In order to boot Linux, we used *shim.elf* [55], a pre-bootloader for GRUB bootloader, hypervisor, and the Linux kernel.

VII. EVALUATION

We compare DeviceVeil with existing solutions, evaluate it against two types of attacks, and measure the performance overhead.

A. Comparison

Table III summarizes the comparison of related proposals with DeviceVeil. Individual USB authentication is achieved by GoodUSB and DeviceVeil. However, GoodUSB depends on serial numbers, which can be easily modified. DeviceVeil uses PUF for individual USB authentication, which ties the authentication process with physical IC characteristics of the USB device. Kells and ProvUSB do not offer individual USB authentication, but they offer individual host authentication and can be combined with individual USB authentication (i.e., DeviceVeil) to achieve mutual authentication. Additional hardware requirement is imposed by USB 3.0 Type-C, and Kells, ProvUSB, and DeviceVeil. Although the cost is cheap on Type-C and DeviceVeil; Kells and ProvUSB require a processor for TPM authentication. Granularity is divided into device level (authenticates a USB device at insert time, causing one-time, low overhead), and packet level (verifies all communication traffic, causing significant runtime overhead). DeviceVeil imposes overhead only at insert-time similar to other implementations except for Cinch and USBFILTER; however, only DeviceVeil offers individual device authentication. Cinch and USBFILTER incur overhead for each packet transferred between a USB device and the host (38% for Cinch and 17.6% for USBFILTER). Cinch and DeviceVeil both use hypervisor for stronger isolation. Type-C, Cinch, and DeviceVeil are OS-independent: Type-C is a standard, and Cinch and DeviceVeil use a hypervisor to run an independent OS. Tamper-proofing is achieved by USBFILTER and DeviceVeil: USBFILTER protects the host with trusted boot and SELinux; DeviceVeil protects the hypervisor and the host with trusted boot, and the USB device with PUF. Overall, DeviceVeil offers comprehensive security at low cost.

B. Attack resistance

We have tested DeviceVeil against two real-world attacks: BadUSB and MTP attack. For BadUSB, we used the code from Caudill [8]. The target USB controller is Phison Electronics Phison 2251-03 (2303), included in Toshiba TransMemory-MX USB 3.0 16GB. BadUSB customizes the firmware on the USB controller and disguises it as a keyboard (HID). However, this malicious BadUSB keyboard is not recognized by DeviceVeil, and the attack fails. For the MTP attack, we used ASUS XenPad 3. Without DeviceVeil, XenPad 3 is detected as an MTP device, and the file system for MTP *jmtpfs* is opened. However, with DeviceVeil, the device is not recognized.

C. Performance

We measured the performance of USB-PUF authentication 100 times on ThinkPad T400. The average time for USB-PUF authentication was 1.705sec (min 1.683sec, max 1.729sec, and std. deviation 0.110sec). For regular use, this delay for authentication appears to be acceptable, considering USB insertion is a manual process.

The target machine is old (Intel Core2 Duo P8400 2.26 GHz) and low performance (Average CPU Mark 1455 [12]). If we can use current relevant machine (ThinkPad T580 with Intel Core i5-8350U 1.70GHz, Average CPU Mark 8189 [11]), we can get better performance because CPU performance is improved by more than 5 times. We know that the CPU improvement does not affect the latency directly, but the latency can be estimated less than 1 seconds from 1.7 seconds (2 times). After authentication, the USB device is accessed as para-pass-through, and thus, incurs no overhead.

VIII. Security Analysis

A. Attacks on BIOS/UEFI

DeviceVeil relies on virtualization, TPM, and secure boot, all of which are set up through BIOS/UEFI. Thus the security of BIOS/UEFI is critical, but some old BIOSes come with default/well-known passwords, which also can be reset by jumper settings when physical access is possible. With improved BIOS/UEFI security in recent times, attacks also became more sophisticated. For example, the System Management Mode (SMM) and sleep mode S3 have been exploited in several attacks (e.g., [17], [26], [82]). Fortunately, these vulnerabilities are also often promptly patched by vendors.

Note that the OS kernel and hypervisor can be made independent of BIOS/UEFI using DRTM (Dynamic Root of Trust Measurement, such as Intel TXT; cf. [85]). The DRTM of GraceWipe utilizes tboot [65] which depends on Intel TXT (Trusted eXecution Technology) and resets the trust chain. GraceWipe uses the secret key stored in the TPM, which is sealed/unsealed by the PCR[17] that is measured by tboot only. The method allows being independent of the measurement from the BIOS/UEFI. However, DeviceVeil cannot depend on DRTM alone, because secure boot and I/O virtualization can be reset if the BIOS/UEFI is compromised. If secure boot is turned off in BIOS, re-installation attacks become possible. If the I/O virtualization is on, a virtual machine may access it through IOMMU.

B. Attacks on Hypervisor

Hypervisors also suffer from vulnerabilities [45], [64]. Since DeviceVeil is a type-I hypervisor and runs only one OS, the attack surface for DeviceVeil is significantly smaller than normal hypervisors with multiple OSes (e.g., Xen). Similarly, cross-VM side-channel attacks [84] are also not a concern. DeviceVeil is based on the para-pass-through architecture and thus, does not suffer from attacks through virtual devices, e.g., VENOM [76] (a vulnerability of floppy emulator in QEMU). However, DeviceVeil also depends on CPU virtualization technologies (i.e., Intel VT-x) as other hypervisors, and may be vulnerable if there are vulnerabilities in such technologies.

C. Attacks on TPM

In contrast to BIOS attacks, DeviceVeil does not care for TPM attacks because they disable booting causing a disadvantage for the attacker, especially for the insider. Even if a side-channel attack is possible, the creation of certain PCR values is difficult. In a TPM reset attack [29], [58], a TPM's LRRESET pin is grounded and initializes PCR values. However, it is still difficult to guess the PCR values to extract the key.

D. Attacks on Secure Boot

As mentioned in Section II, kexec can bypass the secure boot as kexec can load unauthorized kernels bypassing BIOS/UEFI and bootloader [47]. DeviceVeil prevents this attack by using a non-kexec kernel, certified by secure boot. Note that, secure boot could be bypassed by exploiting vulnerabilities in UEFI implementation [17]. The administrator must use an up-to-date, secure UEFI BIOS.

E. Attacks on Memory

The encryption key from the main memory may be extracted by DMA Attacks [54], [60] or cold-boot attacks [22]. These attacks take the memory dump image and search for secret keys (e.g., loaded from a TPM). However, they can be addressed by relocation of secret keys from RAM to other (relatively) safer places, such as SSE registers (AESSE [40]), debug registers (TRESOR [39]), MSR registers (Amnesia [57]), and AVX registers (PRIME [18]), GPU registers (PixelVault [75]). Keys and secrets in RAM can also be protected by other hardware security features in CPUs, such as Intel TSX (Mimosa [21]), and and Intel TXT/AMD SVM (Hypnoguard [86] during ACPI S3 suspension). Note that Intel SGX (Software Guard Extensions) is inapplicable to DeviceVeil although SGX offers encrypted memory and isolated execution, as we require device-level access (ring-0) from the trusted environment (i.e., our hypervisor), but SGX allows only ring-3 (user level) privilege.

Memory attacks can also be launched via software side-channels (e.g., cross-VM side-channel attacks). A disk encryption key is managed by DeviceVeil and stored in the memory allocated to DeviceVeil. DeviceVeil is a type-I hypervisor and runs one OS only, and therefore cross-VM side-channel attacks (e.g., [84]) from the host OS and memory leaks are not applicable to DeviceVeil. However, DeviceVeil's threat model also includes a malicious insider. Such an attacker can use hypervisor forensics tools (e.g., Actaeon [20]) to detect hypervisor from the use of VMCS (Virtual Machine Control Structure) of Intel VT-x. Actaeon requires a memory dump image, but a pure software approach cannot capture the memory properly, according to Graziano et al. [20], who suggested using SMM based memory scanners (e.g., [48]). However, SMM is protected by the security of BIOS/UEFI, which is assumed to be secure by DeviceVeil.

F. Attacks on PUF

General PUF circuits are passive devices and do not keep a state. They are vulnerable to replay attacks. If the DB in the hypervisor of DeviceVeil is disclosed, the data can be exploited to allow unauthorized devices. The current implementation of the DB is protected by encryption (with a TPM-stored key) and we assume that the TPM and DeviceVeil's hypervisor are free of vulnerabilities. To enable replay protections (against any disclosed ciphertext-plaintext pairs), we need to use the one-time challenge-response [36], a common random seed in the PUF authentication circuits and verifier [49], or circuits hosting a key that is decrypted by a PUF created key [30].

A more sophisticated impersonation is modeling attacks [50], [51] that emulate PUF software. Modeling attacks use machine learning to create an algorithm to impersonate the original PUF. However, this type of attack has two drawbacks. The first drawback is that the modeling attacks assume that the challenge and response pairs of target PUF grow only linearly. PL-PUF is based on LFSR that works as a pseudo-random number generator, and thus makes it difficult to model the PUF authentication circuits. The second drawback is that modeling attacks assume that challenge and response pairs are exposed directly to attackers. The PL-PUF authentication circuits do not offer challenge and response pairs directly. A ciphertext is sent with the challenge, and the key in PL-PUF authentication circuits decrypts it to generate the response. The modeling attacks are improved with side-channel attacks [35], but PL-PUF still has the advantage as it uses complex LFSR circuits and only decrypts the AES ciphertext, which difficulty is the same level of AES attacks.

G. Attacks on USB BUS

USB bus has no encryption and is easily sniffed by a USB protocol analyzer. The lost information (e.g., a key) can be used for replay attack or forged authentication [42]. These attacks are not prevented by hardware tamper-proof on a USB device for secret-hiding protection. One solution is encryption of communication, but managing keys for such encryption will be non-trivial and possibly subject to compromise. Even if an encryption key is shared to authenticate a device by some PCs, all PCs must renew the key when a PC loses it. However, PUF allows creating different keys from a device for each authentication [41]. The administrators can set a different key on each PC. Current DeviceVeil uses the database for challenge-response, but it can replace the key created by PUF.

DeviceVeil (hypervisor) issues a "Reset" command to a USB device after the PUF authentication phase and the USB device is visible to the Guest OS via para-pass-through. After PUF authentication, an attacker may attempt to replace the authenticated USB with an illegitimate USB device (i.e., a kind of time-of-check to time-of-use, TOCTOU race condition). However, the USB host controller on the PC detects the physical disconnection of a USB device. DeviceVeil is also aware of the physical disconnection and the PUF authentication starts from the beginning for the newly inserted device.

H. Legacy Devices

For backward compatibility, some users may want to use legacy devices. DeviceVeil offers authentication for legacy devices, but it depends on vulnerable serial numbers. Administrators must evaluate risks from such devices before allowing them, although we strongly recommend using PUF-enabled devices only.

I. Other concerns

DeviceVeil offers strong, one-way device authentication, but mutual authentication is desirable when the host is untrusted. We make the host tamper-proof against unauthorized USB devices and malicious insiders; but it can also integrate host authentication technology (e.g., Kells [7]), depending on the cost and usage scenario.

IX. Conclusions

DeviceVeil is the first solution that authenticates individual USB devices with hardware-based identification circuit, PL-PUF, and hypervisor with a pre-OS authentication mechanism, hardened by TPM and secure boot. It allows connecting only authenticated USB devices and protects the trust-by-default OS and user data against camouflage attacks (e.g., BadUSB), protocol abuses, and unknown USB device attacks. DeviceVeil requires hardware modification, but the estimated cost appears to be reasonable (less than 1 US dollar), and it does not require any on-board processor on the USB device. We believe, DeviceVeil offers a low-cost device authentication technique, mitigating an important security gap in existing USB defense solutions.

REFERENCES

[1] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov, "Cryptographic Processors-A Survey," *Proceedings of the IEEE*, vol. 94, no. 2, 2006.

[2] S. Angel, R. S. Wahby, M. Howald, J. B. Leners, M. Spilo, Z. Sun, A. J. Blumberg, and M. Walfish, "Defending against malicious peripherals with Cinch," in *USENIX Security Symposium*, 2016.

[3] Aoki Laboratory's Cryptographic Hardware Project, "http://www.aoki.ecei.tohoku.ac.jp/crypto/," 2014.

[4] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky, "HyperSentry: Enabling stealthy in-context measurement of hypervisor integrity," in *Conference on Computer and Communications Security*, ser. CCS'10, 2010.

[5] BadAndroid, "https://github.com/tst-zdouglas/badandroid," 2014.

[6] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs," in *Cryptographic Hardware and Embedded Systems*, ser. CHES'08, 2008.

[7] K. R. B. Butler, S. E. McLaughlin, and P. D. McDaniel, "Kells: A Protection Framework for Portable Data," in *Annual Computer Security Applications Conference*, ser. ACSAC'10, 2010.

[8] Caudill's-badUSB, "https://github.com/adamcaudill/psychson," 2014.

[9] T. M. Chen and S. Abu-Nimeh, "Lessons from Stuxnet," *Computer*, vol. 44, pp. 91–93, 2011.

[10] CompTIA, "Cyber secure: A look at employee cybersecurity habits in the workplace," compTIA Whitepaper (2015). https://www.comptia.org/resources/cyber-secure-a-look-at-employee-cybersecurity-habits-in-the-workplace.

[11] CPU Benchmark Intel Core i5-8350U 1.70GHz. (2017) https://www.cpubenchmark.net/cpu.php?cpu=intel+core+i5-8350u+%40+1.70ghz&id=3150.

[12] CPU Benchmark Intel Core2 Duo P8400 2.26 GHz. (2008) https://www.cpubenchmark.net/cpu.php?cpu=intel+core2+duo+p8400+%40+2.26ghz&id=973.

[13] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *International Conference on the Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT'04, 2004.

[14] K. Eguro and R. Venkatesan, "FPGAs for trusted cloud computing," in *International Conference on Field Programmable Logic and Applications*, ser. FPL'12, 2012.

[15] T. Eisenbarth, T. Güneysu, C. Paar, A.-R. Sadeghi, D. Schellekens, and M. Wolf, "Reconfigurable Trusted Computing in Hardware," in *Workshop on Scalable Trusted Computing*, ser. STC'07, 2007.

[16] K. B. Frikken, M. Blanton, and M. J. Atallah, "Robust authentication using physically unclonable functions," in *International Conference on Information Security*, ser. ISC'09, 2009.

[17] A. Furtak, Y. Bulygin, O. Bazhaniuk, J. Loucaides, A. Matrosov, and M. Gorobets, "BIOS and Secure Boot Attacks Uncovered," *(ekoparty)*, 2014.

[18] B. Garmany and T. Müller, "Prime: Private rsa infrastructure for memory-less encryption," in *The Annual Computer Security Applications Conference*, ser. ACSAC '13, 2013.

[19] GlobalPlatform. https://globalplatform.org/.

[20] M. Graziano, A. Lanzi, and D. Balzarotti, "Hypervisor memory forensics," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, ser. RAID'13, 2013.

[21] L. Guan, J. Lin, B. Luo, J. Jing, and J. Wang, "Protecting private keys against memory disclosure attacks using hardware transactional memory," in *IEEE Symposium on Security and Privacy*, ser. SP'15, 2015.

[22] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Cal, A. J. Feldman, and E. W. Felten, "Least we remember: Cold boot attacks on encryption keys," in *USENIX Security Symposium*, 2008.

[23] S. Han, W. Shin, J.-H. Park, and H. Kim, "A bad dream: Subverting trusted platform module while you are sleeping," in *USENIX Security Symposium*, Baltimore, MD, USA, Aug. 2018.

[24] Y. Hori, H. Kang, T. Katashita, and A. Satoh, "Pseudo-LFSR PUF: A compact, efficient and reliable physical unclonable function," in *International Conference on Reconfigurable Computing and FPGAs*, 2011.

[25] K. Ishii and T. Komukai, "A Comparative Legal Study on Data Breaches in Japan, the U.S., and the U.K." in *IFIP TC International Conference on Human Choice and Computers (HCC'16)*, 2016.

[26] C. Kallenberg, J. Butterworth, X. Kovah, and S. Cornwell, "Defeating signed bios enforcement," *MITRE White Paper*, 2014.

[27] M. Kang and H. Saiedian, "USBWall: A novel security mechanism to protect against maliciously reprogrammed USB devices," *Information Security Journal: A Global Perspective*, vol. 26, no. 4, pp. 166–185, 2017.

[28] Kanguru's FlashTrust, "https://www.kanguru.com/storage-accessories/kanguru-flashtrust-secure-firmware.shtml," 2014.

[29] B. Kauer, "OSLO: Improving the Security of Trusted Computing," in *USENIX Security Symposium*, 2007.

[30] T. Kubota, M. Shiozaki, and T. Fujino, "Robust authentication using physically unclonable functions," in *Embedded Security in Cars*, ser. ESCAR'16, 2016.

[31] K. Kursawe, "Trusted Computing and its Applications: An Overview," in *Information Security Solutions Europe*, ser. ISSE'06, 2004.

[32] D. Kushner, "The real story of stuxnet," *ieee Spectrum*, vol. 50, no. 3, pp. 48–53, 2013.

[33] J. W. Lee, D. Lim, B. Gassend, G. E. S. andMarten van Dijk, and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications," in *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525)*, 2004.

[34] E. L. Loe, H.-C. Hsiao, T. H.-J. Kim, S.-C. Lee, and S.-M. Cheng, "SandUSB: An installation-free sandbox for USB peripherals," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Dec 2016, pp. 621–626.

[35] A. N. Mahmoud, U. Rührmair, M. Majzoobi, and F. Koushanfar, "Combined modeling and side channel attacks on strong pufs," *IACR Cryptology ePrint Archive*, 2013.

[36] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Techniques for design and implementation of secure reconfigurable PUFs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. Vol 2, no. 1, 2009.

[37] K. Mayes and K. Markantonakis, *Smart Cards Tokens Security and Applications*. Springer, 2008.

[38] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "TrustVisor: Efficient TCB reduction and attestation," in *IEEE Symposium on Security and Privacy*, ser. SP'10, 2010.

[39] T. Müller, F. Freiling, and A. Dewald, "Tresor runs encryption securely outside ram," in *20th USENIX Security Symposium (USENIX Security 11)*, 2011.

[40] T. Müller, A. Dewald, and F. C. Freiling, "AESSE: a cold-boot resistant implementation of AES," in *the 3rd European Workshop on System Security*, 2010.

[41] M. Nabeel, S. Kerr, X. Ding, and E. Bertino, "Authentication and key management for advanced metering infrastructures utilizing physically unclonable functions," in *Smart Grid Communications*, ser. SmartGridComm'12, 2012.

[42] M. Neugschwandtner, A. Beitler, and A. Kurmus, "A transparent defense against usb eavesdropping attacks," in *European Workshop on System Security*, ser. EuroSec'16, 2016.

[43] K. Nohl, S. Krissler, and J. Lell, "BadUSB-On accessories that turn evil," blackHat USA 2014.

[44] G. Ose, "Exploiting USB Devices with Arduino," blackHat USA 2011.

[45] D. Perez-Botero, J. Szefer, and R. B. Lee, "Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers," in *Workshop on Security in Cloud Computing*, 2013.

[46] A. Pfiffer, "Reducing System Reboot Time With kexec," *OSDL Whitepaper*, 2003.

[47] Phoronix.com, "Secure boot breaks kexec, hibernate support on Linux," news article (Jan. 28, 2013). https://www.phoronix.com/scan.php?page=news_item&px=MTI4NjE.

[48] A. Reina, A. Fattori, F. Pagani, L. Cavallaro, and D. Bruschi, "When Hardware Meets Software: A Bulletproof Solution to Forensic Memory Acquisition," in *Annual Computer Security Applications Conference*, ser. ACSAC'12, 2012.

[49] M. Rostami, M. Majzoobi, F. Koushanfar, D. S. Wallach, and S. Devadas, "Robust and reverse-engineering resilient PUF authentication and key-exchange by substring matching," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, 2014.

[50] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling Attacks on Physical Unclonable Functions," in *Conference on Computer and Communications Security*, ser. CCS'10, 2010.

[51] U. Rührmair, J. Solter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "PUF Modeling Attacks on Simulated and Silicon Data," *IEEE Trans. Info. For. Sec.*, 2013.

[52] Secure Element (SE) Committee. (2007) https://globalplatform.org/technical-committees/secure-element-se-committee/.

[53] A. Seshadri, M. Luk, N. Qu, and A. Perrig, "SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes," in *Symposium on Operating Systems Principles*, ser. SOSP'07, 2007.

[54] R. Sevinsky, "Funderbolt: Adventures in thunderbolt DMA attacks," *Black Hat USA*, 2013.

[55] Shim, "https://github.com/rhinstaller/shim," 2015.

[56] T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai, K. Kono, S. Chiba, Y. Shinjo, and K. Kato, "BitVisor: A thin hypervisor for enforcing I/O device security," in *Conference on Virtual Execution Environments*, ser. VEE'09, 2009.

[57] P. Simmons, "Security through amnesia: A software-based solution to the cold boot attack on disk encryption," in *The Annual Computer Security Applications Conference*, ser. ACSAC '11, 2011.

[58] E. R. Sparks, "A security assessment of trusted platform modules," *Dartmouth College Technical Report*, 2007.

[59] D. Spill, "USBProxy: An open and affordable USB man in the middle device."

[60] P. Stewin, *Detecting peripheral-based attacks on the host memory*. PhD thesis, Technischen Universität Berlin, 2014.

[61] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Design Automation Conference*, ser. DAC'07, 2007.

[62] K. Suzaki, "DeviceDisEnabler: A lightweight hypervisor which hides devices to protect cyber espionage and tampering," blackHat Sao Paulo 2014.

[63] D. Suzuki and K. Shimizu, "The glitch PUF: A new delay-PUF architecture exploiting glitch shapes," in *Cryptographic Hardware and Embedded Systems*, ser. CHES'10, 2010.

[64] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the Hypervisor Attack Surface for a More Secure Cloud," in *Conference on Computer and Communications Security*, ser. CCS'11, 2011.

[65] tboot. (2008) http://tboot.sourceforge.net/. [Online]. Available: http://tboot.sourceforge.net/

[66] D. J. Tian, A. Bates, and K. Butler, "Defending against malicious USB firmware with GoodUSB," in *Annual Computer Security Applications Conference*, ser. ACSAC'15, 2015.

[67] D. J. Tian, A. Bates, K. R. Butler, and R. Rangaswami, "ProvUSB: Block-level Provenance-Based Data Protection for USB Storage Devices," in *Conference on Computer and Communications Security*, ser. CCS'16, 2016.

[68] D. J. Tian, N. Scaife, A. Bates, K. Butler, and P. Traynor, "Making USB great again with USBFILTER," in *USENIX Security Symposium*, 2016.

[69] D. J. Tian, N. Scaife, D. Kumar, M. Bailey, A. Bates, and K. R. Butler, "SoK: "Plug & Pray" Today – Understanding USB Insecurity in Versions 1 through C," in *IEEE Symposium on Security and Privacy*, ser. SP'18, 2018.

[70] M. Tischer, Z. Durumeric, S. Foster, S. Duan, A. Mori, E. Bursztein, and M. Bailey, "Users really do plug in USB drives they find," in *IEEE Symposium on Security and Privacy*, ser. SP'16, 2016.

[71] TrouSerS. (2005) https://sourceforge.net/projects/trousers/.

[72] USB-3.0-Promoter-Group, "Universal serial bus Type-C authentication specification release 1.0 with ECN and Errata," 2017.

[73] USB-On-The-Go, "http://www.usb.org/developers/onthego," 2001.

[74] USB-Raptor. (2014) https://sourceforge.net/projects/usbraptor/.

[75] G. Vasiliadis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Pixelvault: Using gpus for securing cryptographic operations," in *Conference on Computer and Communications Security*, ser. CCS'14, 2014.

[76] VENOM, "http://venom.crowdstrike.com/," 2016.

[77] T. Vidas, D. Votipka, and N. Christin, "All your droid are belong to us: A survey of current Android attacks," in *USENIX Workshop on Offensive Technologies*, ser. WOOT'11, 2011.

[78] Z. Wang and A. Stavrou, "Attestation & authentication for USB communications," in *International Conference on Software Security and Reliability Companion*, ser. SERE'12, 2012.

[79] Z. Wang and X. Jiang, "HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity," in *IEEE Symposium on Security and Privacy*, ser. SP'10, 2010.

[80] A. Washburn, "Snowden smuggled documents from NSA on a thumb drive. https://www.wired.com/2013/06/snowden-thumb-drive/."

[81] R. Wilkins and B. Richardson, "UEFI secure boot in modern computer security solutions," *UEFI Forum*, 2013.

[82] R. Wojtczuk and C. Kallenberg, "Attacks on uefi security," *CanSecWest*, 2015.

[83] P. Zaitcev, "The usbmon: USB monitoring framework," in *Linux Symposium*, 2005.

[84] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-vm side channels and their use to extract private keys," in *Conference on Computer and Communications Security*, ser. CCS'12, 2012.

[85] L. Zhao and M. Mannan, "Gracewipe: Secure and Verifiable Deletion under Coercion." in *Network and Distributed System Security Symposium*, ser. NDSS'15, 2015.

[86] ——, "Hypnoguard: Protecting secrets across sleep-wake cycles," in *Conference on Computer and Communications Security*, ser. CCS'16, 2016.

[87] Zuiho. (2013) http://www.toptdc.com/en/product/sasebo/.