KISS: "Key it Simple and Secure" Corporate Key Management

Zongwei Zhou, Jun Han, Yue-Hsun Lin, Adrian Perrig, Virgil Gligor
CyLab and Carnegie Mellon University, Pittsburgh, Pennsylvania, United States
{stephenzhou, junhan, tenma, perrig, gligor}@cmu.edu

Abstract. Deploying a corporate key management system faces fundamental challenges, such as fine-grained key usage control and secure system administration. None of the current commercial systems (either based on software or hardware security modules) or research proposals adequately address both challenges with small and simple Trusted Computing Base (TCB). This paper presents a new key management architecture, called KISS, to enable comprehensive, trustworthy, user-verifiable, and cost-effective key management. KISS protects the entire life cycle of cryptographic keys. In particular, KISS allows only authorized applications and/or users to use the keys. Using simple devices, administrators can remotely issue authenticated commands to KISS and verify system output. KISS leverages readily available commodity hardware and trusted computing primitives to design system bootstrap protocols and management mechanisms, which protects the system from malware attacks and insider attacks.

Key words: Key Management, Trusted Computing, Isolation, Trusted Path

1 Introduction

As consumers and corporations are increasingly concerned about security, deployments of cryptographic systems and protocols have grown from securing online banking and e-commerce to web email, search, social networking and sensitive data protection. However, the security guarantees diminish with inadequate key management practices, as exemplified by numerous real-world incidents. For example, in 2010 Stuxnet targeted Iranian uranium centrifuges, installing device drivers signed with private keys *stolen* from two high-tech companies [11]. In another incident, the private keys of DigiNotar, a Dutch certificate authority, were maliciously *misused* to issue fraudulent certificates for Gmail and other services [23]. Even high-profile, security-savvy institutions fall prey to inadequate key security, let alone companies with a lower priority for security.

Despite its indisputable significance, *none* of the current corporate key management systems (KMS) – either industrial solutions based on software, or hardware security modules (HSM), or research proposals known to us – provide comprehensive key management *with small and simple trusted computing base (TCB)*. There are at least two significant challenges that lead to the insufficiency of the KMS, as shown in Table 1.

Fine-grained Key Usage Control. A comprehensive life-cycle KMS should enforce *fine-grained key usage control* (i.e., whether an application operated by a user has the permission to access a specific cryptographic key). This problem is exacerbated with the current trend of Bring Your Own Device (BYOD), which allows client devices (e.g.,

Systems	Key Usage Control	Administration Interfaces	TCB	ROT
HSM [18,20,7,16,17]	coarse-grained (application or machine control)	HSM & complex admin dev, non-verifiable	large	HSM, admin dev
SW [15,19,9]	insecure (rely on OS)	keyboard/display, non-verifiable	large	OS
TPM [5,11,2,13,14]	coarse-grained (only application control)	keyboard/display, non-verifiable	large	TPM
KISS	fine-grained (both application and user control)	trusted path & simple admin dev, verifiable	small	TPM, admin dev

Table 1. A comparison between KISS and current key management systems. "HSM", "SW", and "TPM" represent the KMS that are based on HSM, software packages, and TPM seal storage, respectively. "ROT" denotes the root of trust of the systems.

tablets and laptops) to increasingly host both personal and security-sensitive corporate applications and data.

Although commercial HSMs [18, 20, 7, 15, 17] provide high-profile physical protection of cryptographic keys and algorithms, they fail to control key usage requests from outside their physical protection boundary (e.g., the users and applications on other client computers). The attackers can cause key misusage [23] by compromising client computers and submitting fake key usage requests to the HSMs. Some HSMs enable porting key usage applications to an in-module secure execution environment [18, 20]. This method only provides application-level key usage control, and is not scalable due to the limited resources of the dedicated environment. Some HSMs enforce key usage control by accepting requests from client machines that deploy special hardware tokens only. This mechanism is insecure because it cannot block requests from a compromised operating system (OS) or an application on an authenticated machine.

Cost-sensitive companies commonly deploy *key management software* [14, 19, 8] on commodity servers, and rely heavily on the underlying OS services to protect cryptographic keys and management operations. These systems are untrustworthy because modern OSes are large and routinely compromised by malware.

Research proposals (e.g., credential protection systems [5, 10, 2] and hypervisor-based solutions [12, 13]) leverage Trusted Platform Modules (TPM) sealed storage. It assures that the cryptographic keys sealed by an application can only be accessed by the same software. However, this approach is coarse-grained; it does not enforce any user authentication of the sealed keys.

Secure System Administration. A trustworthy KMS should allow benign administrators to securely manage the system and defend against attacks from malicious insiders. It must guarantee the authenticity of the communication between the administrators and the KMS. Otherwise, an adversary can cause unintended key management operations by stealing administrator login credentials, modifying or spoofing the administrator command input or the KMS output (e.g., operation result, system status).

The HSMs usually mandate the administrators to perform management operations via the I/O devices (e.g., keyboard and display) that are physically attached to the modules. For remote administration, they need complicated management software running on a commodity OS or a dedicated administrator device. Both mechanisms signifi-

cantly increase system TCB and thus exposes larger attack surface. For software-based KMS, the I/O interfaces and authentication-relevant devices are controlled directly by the underlying OS, which means that the administrator credentials, input commands, and KMS output can easily be compromised by malware in the OS. Similarly, research proposals [5, 2, 10] do not support trustworthy remote management mechanisms. More importantly, none of KMS solutions provide intuitive ways for administrators to verify the status of the administration interfaces. Without such verification, administrators cannot trust any displayed system output and may mistakenly perform operations.

Contributions. To address the above challenges, this paper presents KISS (short for "Key it Simple and Secure"), a comprehensive, trustworthy, user-verifiable, and cost-effective enterprise key management architecture. Table 1 compares KISS with mainstream KMS and research proposals. Among them, KISS is the first KMS that supports *fine-grained key usage control* based on users, applications, and configurable access-control policies. To do this, KISS isolates authorized corporate applications from the untrusted OS and measures the code identities (cryptographic hash) of the protected applications. KISS also directly accepts user authentication by isolating user-interface devices and authentication relevant devices from the OS. Moreover, KISS enables *secure system administration*, leveraging a simple external device with minimal software/hardware settings. The KISS administrators execute thin terminal software on commodity machines. The thin terminal accepts administrator input via trusted paths, remotely transfers the input to and receives system output from the KISS system. The administrators use the external devices to *verify* the execution of the thin terminal and trusted paths and guarantee the *authenticity* of the input/output.

KISS leverages hypervisor-based isolation to protect the key management software and cryptographic keys from the large untrusted OS, applications, and peripheral devices. The administrators securely bootstrap the KISS system using the simple administrator devices and lightweight protocols, regardless of malware attacks and insider attacks from malicious administrators. These mechanisms together significantly reduce and simplify the KISS TCB, enabling higher security assurance. Because KISS leverages commodity hardware and trusted computing techniques, it is *cost-effective* and makes the wide adoption of KISS in small- and medium-sized business possible, in addition to financial or governmental institutions. KISS showcases how trusted computing technologies achieve tangible benefits when used to design trustworthy KMS.

Paper Organization. First, we describe the KISS attacker model and introduce the background in Sections 2 and 3, respectively. Section 4 describes in detail the KISS system model and administrative policies. In Section 5, we illustrate the KISS hypervisor-based architecture and the simplicity of the external administrator devices. Sections 6, 7, and 8 introduce the detailed mechanisms for system bootstrap, secure administration, and fine-grained key usage control, respectively. We analyze potential attacks on KISS and our defense mechanisms in Section 9. Section 10 discusses KISS extensions with stronger security properties or address real-world application issues. We then compare our solution with related work (Section 11) and conclude the paper.

2 Attacker Model

We consider an adversary that remotely exploits the *software vulnerabilities* of the OS and applications on KMS machines. The adversary can then access any system resources managed by the OS (e.g., memory, chip-set hardware, peripheral devices) and subvert any security services provided (e.g., process isolation or file system access-control). However, we trust the correctness of the key management software, and assume that it cannot be exploited by the adversary. The mechanisms to guarantee the correctness is out of the scope of this paper.

We also consider *insider attacks* from malicious administrators that attempt to leak, compromise, or misuse the cryptographic keys. They can actively issue unauthorized key management operations, intentionally misconfigure the KMS and corporate applications, or steal the administrator devices or credentials (e.g., password, smart cards) of benign administrators. However, benign administrators are trusted to protect their administrator devices and credentials and comply with the KISS protocols.

We do *not* address the following three types of attacks in this paper: (1) physical attacks to the hardware that KISS relies on (e.g., TPM), (2) side-channel attacks to cryptographic keys and algorithms, and (3) denial-of-service attacks. Countermeasures against these attacks are complementary to KISS.

3 Background

This section introduces the technical building blocks of KISS: program isolation [12, 13] and trusted paths [25, 3]. They are implemented based on readily available trusted computing primitives, such as dynamic root of trust for measurement (or *Late Launch*) [1, 9], *remote attestation*, and *sealed storage* [21].

Program Isolation. Recent research contributions [12, 13] demonstrate the capability of removing large commodity OS from the TCB of small program modules. These systems isolate program modules by leveraging a small and trustworthy hypervisor with higher privilege level than the OS. The hypervisor guarantees that the OS, applications, and DMA-capable devices cannot compromise the execution integrity, data integrity, and secrecy of the isolated program modules. The protected code modules are self-contained, and they should not rely on OS services.

Trusted Path. A Trusted Path (TP) is a protected channel providing secrecy and authenticity of data transfers between a user's I/O devices (e.g., keyboard, display, USB devices) and an isolated program trusted by the user. Recent research advances demonstrate the usage of a small, dedicated hypervisor to establish trusted paths, completely bypassing the commodity OS [25, 3]. The hypervisor exclusively redirects the I/O communications (e.g., memory-mapped I/O, DMA, interrupts) of the trusted-path devices to the isolated software module. The TP device drivers are included in the isolated software module and redesigned to communicate with the devices via the hypervisor.

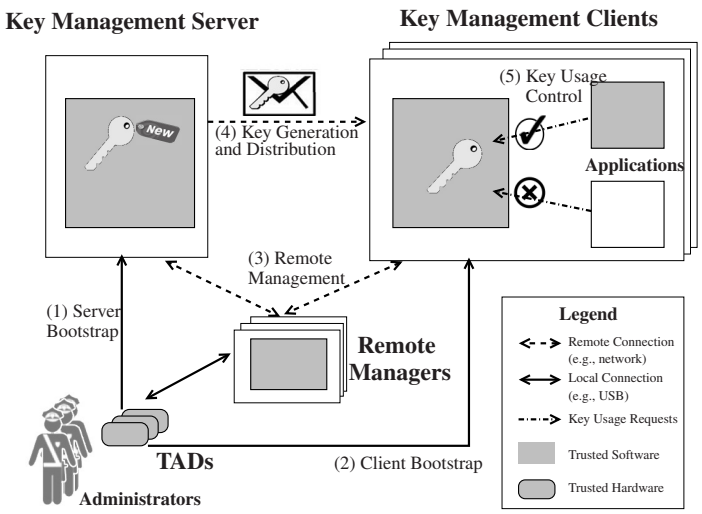


Fig. 1. KISS system model.

4 Overview

Corporate key management in this paper refers to the establishment and usage of cryptographic keys in corporate and distributed environments. In this section, we provide a high-level overview of KISS system entities and model, and demonstrate how this model enables scalable and hierarchical enterprise key management.

4.1 System Entities

Figure 1 shows the four major entities in the KISS system.

Key Management Server (KISS Server). A commodity server machine that executes the key management software to perform server-side key life-cycle operations (e.g., key generation, registration, backup, revocation, de-registration, and destruction).

Key Management Clients (KISS Clients). Distributed machines (e.g., employees' desktops or corporate web servers) that install the KISS client software to receive cryptographic keys from the KISS server and use the keys to provide services to corporate applications. For example, On employees' desktops, the cryptographic keys stored in the KISS client software can be used to encrypt confidential documents. For a corporate web server, the keys are used to authenticate the outgoing network traffic.

Remote Managers (KISS Managers). Commodity machines used by KISS administrators to perform remote management. These machines install the KISS manager software to securely transfer administrative commands to and receive system output from the KISS server or clients.

Trusted Administrator Devices (KISS TAD). Small, dedicated devices that are directly connected (e.g., via USB) to the KISS server or clients for local administration, or connected with the KISS managers for remote management.

4.2 System Model

Figure 1 also demonstrates a basic workflow of bootstrapping and using the KISS system. In Steps (1) and (2), administrators install and execute the KISS software on the server or clients, and perform bootstrap protocols to establish cryptographic channels between the server software, client software and TADs. We design our system to protect the server/client software and the channel keys against malware attacks (see Section 5). The bootstrap protocols must be performed by a quorum of administrators to defend against malicious insider attacks. Each participating administrator use his/her TAD to confirm that the KISS bootstrap process succeeds. After bootstrap, the KISS server software starts recording subsequent system operations in a tamper-evident audit log, which helps the administrators detect insider attacks. Section 6 illustrates the KISS bootstrap protocols, cryptographic channel establishment, and audit log in detail.

In Step (3), the administrators remotely manage the KISS server/client software, leveraging their TADs and KISS managers. The KISS system protects the manager software (acting as a thin terminal) and user-interfaces devices (e.g., keyboard, and display) against malware attacks from the KISS manager OS. The administrators can securely input commands and review system output via the KISS manager user interfaces. The administrators use their TADs to authenticate the outgoing commands, and verify the authenticity of the operation results back from the KISS server/client software. Section 7 describes the remote management process and how our design significantly reduces KISS TCB.

In Step (4), new cryptographic keys (which are our key management products) are generated in the KISS server and securely distributed to the clients via the cryptographic channels established in step (2). In Step (5), the KISS client software protects the distributed keys, and handles key usage requests from various applications. KISS enables more fine-grained control of key usage than previous key management systems and proposals. It isolates the applications (similar to the isolation of KISS server software from the server OS) and measures their code identities. It also provides protected channels between authentication devices and the KISS client software, so that the KISS client software can directly authenticate the users of the applications. If the requests are from authorized users (e.g., company employees) and corporate applications (e.g., corporate document editors), the KISS client software uses the corresponding cryptographic keys to process the requests (e.g., decrypt confidential documents). The KISS client software rejects any key usage request from unauthorized users (e.g., visitors that are not allowed to read any confidential document) or applications (e.g., personal web browsers, media players). Section 8 describes the detailed mechanisms of our fine-grained key usage control.

The KISS client is necessary for collecting application and user information to perform key usage control. By receiving keys from the server, it also supports offline key usage, which reduces the key access latency and allows key usage when network connections are unavailable (e.g., while traveling on flights). However, offline key usage increases the risk of key abuse (e.g., when client machines are stolen). Companies might enforce special key usage policies to reduce this risk, such as requiring client machines to periodically obtain key usage permissions from the KISS server. Note that KISS can easily be modified to serve as the key usage control front end of the HSM.

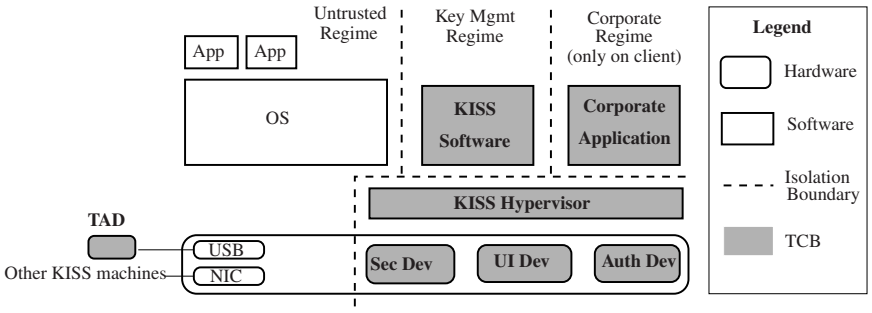


Fig. 2. System architecture for KISS client, server, and manager. Sec Dev is the hardware (e.g., TPM) that provides trusted computing primitives. UI Dev denotes the user-interface devices, such as a keyboard and a display. Auth Dev is the device used for authentication (e.g., fingerprint scanner, and keypad). The KISS machines communicate via the network interface cards (NIC), and connects with TADs via USB interfaces.

The KISS server software receives approved key usage requests from the clients, and securely transfers them to the HSM on the server machines via trusted paths. Both the cryptographic keys and algorithms are always protected inside HSM.

5 System Architecture

In this section, we introduce the unified hypervisor-based architecture for the KISS server, client, and manager, and the hardware/software settings of TAD. We demonstrate how our architectural design significantly reduces and simplifies the TCB of the whole system, which is necessary for achieving high security assurance.

5.1 KISS Server, Client, and Manager

KISS server, client, and manager share the same architecture, hence we only illustrate the KISS client in detail here. As shown in Figure 2, the KISS hypervisor is a thin layer of software running in a higher privilege than the commodity OS of the client. Unlike commercial hypervisors/virtual machine monitors (VMM) (e.g., VMware Workstation, Xen), the KISS hypervisor does not virtualize all hardware resources or support the concurrent execution of multiple OSes. Thus, the code base of the KISS hypervisor is orders of magnitude smaller and demonstrably simpler than an OS or a full-functioning hypervisor/VMM. The TCB of a KISS client is only the hypervisor, the client software, the corporate applications that utilize the keys, and some commodity hardware (e.g., Sec, Auth, and UI Dev in Fig. 2). The KISS hypervisor is dedicated to three main tasks:

Isolation. The KISS hypervisor divides the client to three isolated software regimes, which are lightweight "virtual machines" as described in Section 3. The key management regime runs the KISS client software and stores all cryptographic keys during its run time. We also leverage TPM sealed storage to protect the cryptographic keys at rest.

Each authorized application that uses the keys is isolated in its own corporate regime. The untrusted regime consists of the commodity OS, other applications, and devices.

Trusted Paths. When the administrators locally manage the client machine, the hypervisor establishes trusted paths between the client software and the UI Dev or Auth Dev (Figure 2). The trusted paths protect the administrator command input and the client software output and safeguard the user authentication credentials. We defer the detailed explanation to subsequent sections.

Key Usage Control. The hypervisor helps the KISS client software to collect the identifier of the corporate applications and users that request key usage. When isolating the corporate applications in corporate regimes, the KISS hypervisor computes a cryptographic hash of the corporate application code and static data, and transfers the hash value as application identifiers to the KISS client software. The hypervisor also establishes trusted paths between the authentication-relevant devices and the KISS client software, for user authentication. Section 8 describes the key usage control procedure.

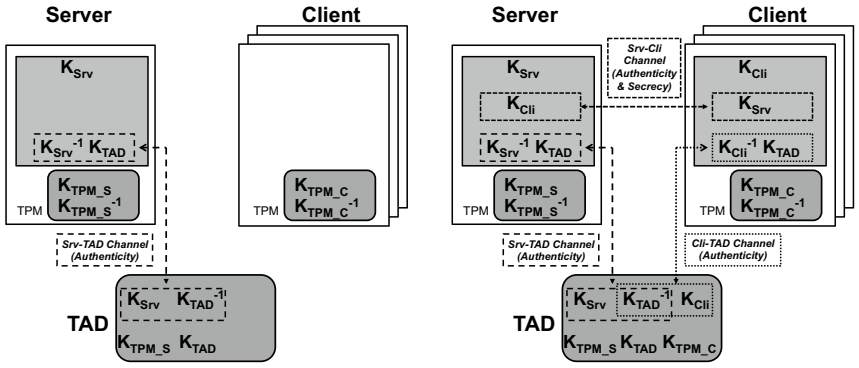
5.2 TAD

TAD is a small, dedicated, embedded device that assists system administration, both locally and remotely. TAD employs much simpler software/hardware than the typical administrator devices in current KMS. TAD does not need a full user-interface hardware for the key management command input and system output. Instead, the administrator can leverage the trusted paths provided by the KISS hypervisor on the server, client or manager. TAD does not implement complicated key management software to interpret operation input/output. These are directly handled by the KISS server/client software. During remote management, the KISS manager software only collects and transfers administrator input to server/client, and receives returning operation results.

TAD implements software for the KISS bootstrap protocol, standard cryptographic primitives, remote attestation protocol, and necessary hardware drivers (note that the USB driver code is included, but not in the TCB). The TAD software is responsible for three tasks: (1) performing server/client bootstrap; (2) remotely attesting to the KISS server, client, and manager software; and (3) authenticating the administrator input and verifying the authenticity of the server/client output. To meet these functional requirements, TAD includes only a low-end CPU, small on-chip RAM and flash storage, a USB controller, a few buttons, a minimal display to show hexadecimal values, and a physical out-of-band channel receiver (e.g., QR code scanner).

6 System Bootstrap

In this section, we introduce the lightweight KISS bootstrap protocols. These protocols allow a quorum of administrators to verify that the "known good" KISS software is executing on the server/clients, and to establish cryptographic channels between their TADs, the server software and the client software. These channels (depicted in Figure 3) are used in secure system administration and key life-cycle operations. The bootstrap protocols are resilient against malware and insider attacks.



(a) Server Bootstrap (b) Client Bootstrap and Registration

Fig. 3. Cryptographic channels established during KISS bootstrap. Before the bootstrap, the server and clients only have their TPM keys, and TADs has no pre-injected keys.

1. TPM $\xrightarrow{OOB} TAD_i$: K_{TPM_S}
2. TAD_i : Generates device key pair $\{K_{TAD_i}, K_{TAD_i}^{-1}\}$
3. $TAD_i \rightarrow$ Server : $\{\mathsf{C}_i, K_{TAD_i}\}$, where C_i lists the configurations of the Server, e.g., # of involved administrators N, and quorum threshold t.
4. Server : Gathers N messages from TAD_i before timeout, late launches HYP and Server (their measurement is stored in TPM).
5. Server : Checks that all C_i are consistent, and $N \geq t$, generates Server key pair $\{K_{Srv}, K_{Srv}^{-1}\}$
6. Server $\rightarrow$ TPM : Stores the measurement of $\{K_{Srv}, C_i, \Lambda = \{K_1, \cdots, K_N\}\}$
7. $TAD_i \rightarrow$ TPM : Nonce R_i
8. TPM $\rightarrow$ Server : Signature $S_i = \{\mathsf{R}_i, \mathsf{M}\}_{K_{TPM\_S}^{-1}}$, where M is the measurement of $\{HYP, Server, K_{Srv}, \mathsf{C}_i, \Lambda\}$.
9. Server $\rightarrow TAD_i$: ID_i, S_i, Λ, K_{Srv}, where ID_i is a unique identifier for TAD_i
10. TAD_i : Verifies S_i and M, checks $K_{TAD_i} \in \Lambda$, $\#(\Lambda) = N$, and stores K_{Srv}

Fig. 4. KISS Server Bootstrap Protocol. Each administrator possesses a TAD_i.

6.1 Server Bootstrap

During the KISS server bootstrap, a quorum of administrators execute authentic KISS server software and establish the Srv-TAD cryptographic channel (Figure 3(a)). Our lightweight server bootstrap protocol needs minimal administrator involvement. It does not require pre-sharing secrets in TAD (e.g., vendor-injected device private keys). After the bootstrap, the server software starts recording subsequent system operations in a tamper-evident audit log, which help the administrators detect insider attacks.

Bootstrap Protocol. Figure 4 illustrates the server bootstrap protocol. Before the protocol begins, we assume that the administrators creates the necessary configuration file, C_i, of the KISS server software independently and store them in TADs. The C_i includes the number of participating administrators, N, a quorum threshold, t, and other necessary server parameters. In Step 1, each administrator gathers the information of the hardware root of trust, i.e., the TPM public key K_{TPM_S} of the server, via a trusted

out-of-band (OOB) channel. We suggest a secure and practical OOB channel, in which K_{TPM_S} is encodes as a tamper-evident physical label, e.g., an etched QR code on TPM chip surface. Each TAD_i securely attains K_{TPM_S} by scanning the QR code.

After that, each TAD_i generates a device key pair, $\{K_{TAD_i}, K_{TAD_i}^{-1}\}$, and sends C_i along with the public key, K_{TAD_i}, to the server (Steps 2 and 3). In Steps 4–6, the server executes the KISS hypervisor and server software via late launch primitives [1, 9] Late launch resets a special Platform Configuration Register (PCR) of the TPM, and stores the cryptographic measurement of the HYP and the server software in this register for further remote attestation. After that, the server software generates a key pair, $\{K_{Srv}, K_{Srv}^{-1}\}$, and a key list, Λ, by receiving the public keys, K_{TAD_i}, from all participating TADs. The server software stores the measurement of K_{Srv}, C_i, and Λ into other PCRs of the TPM. The accumulated measurement, together with its signature generated by TPM attestation keys (linked with the TPM private key, $K_{TPM\_S}^{-1}$), are sent to the verifier during remote attestation (Step2 7– 9).

Upon receiving the attestation response, TAD verifies the signature using K_{TPM_S}, and trusts the authenticity of the accumulated measurement, M (Step 10). TAD recomputes M using its pre-installed knowledge (e.g., cryptographic hash of HYP and server software, configuration file C_i), the received K_{Srv} and Λ. If the verification succeeds, TAD trusts that an authentic hypervisor/server instance is executing on the KISS server with the appropriate configurations, and that the server instance has the server private key and a correct list of TAD public keys. TAD also verifies that its own public key is included in the public key list, Λ, and the number of keys in Λ equals to the number of participating administrators. If all verification passes, TAD notifies its administrator via the display. The only task that each administrator needs to perform is to visually check that all TADs display verification success messages. KISS introduces an additional computational overhead (e.g., remote attestation and quorum checking) compared to traditional system bootstrapping. However, we argue that this cost is acceptable, considering the security guarantees it achieves.

Audit Log. During the server bootstrap, malicious administrators may inject spurious configuration files with a small quorum threshold, or even forge administrator public keys. These administrators are then capable of passing the quorum check that is necessary for any key management operations. In KISS, the server software maintains an operation log to record all of the system administration operations, including bootstrap operations. This helps legitimate administrators/auditors detect any insider attacks during the server bootstrap. In addition, the audit log helps relaxes the quorum control and improves system usability. Becasue all key management operations are held accountable, KISS may allowing a smaller number of administrators or even merely one to perform operations.

The audit log is stored in the untrusted regime. The KISS server software maintains an aggregated hash of the log entries in the TPM non-volatile memory (NVRAM). The TPM NVRAM access-control (similar to sealed storage) ensures that only KISS server software can access/update that hash, Note that frequent NVRAM updates are impractical on TPM. To minimize NVRAM updates, we leverage an update mechanism that is similar to the PCR-NVRAM two stage update technique presented in [16]. During the

audit procedure, the auditor verifies the integrity of the log by recomputing the aggregated hash and comparing it with the hash stored in TPM NVRAM.

6.2 Client Bootstrap and Registration

Bootstrapping a KISS client is similar to the server bootstrap. A quorum of administrators verifies the authenticity of the KISS hypervisor, client software, and its configuration file. The client software securely sends its public key, K_{Cli}, to each of the participating TADs, and collects the device public keys K_{TAD_i} (generated during the server bootstrap). The configuration file sent to the client software differs from the one established during the server bootstrap. It contains the server public key, K_{Srv}, and the client-side system parameters (e.g., access-control policies of key usage, user authentication information, and the corporate application information). These client-side configurations are used in the fine-grained key usage control (See Section 8). Upon a successful client bootstrap, TADs establish Cli-TAD cryptographic channels with the KISS client, which allows subsequent client administration.

The administrators then register the client to the server by sending the client software public key, K_{Cli}, to the server software, via Srv-TAD cryptographic channels. This establishes the Srv-Cli cryptographic channel (see Figure 3(b)). This channel diffs from the Srv-/Cli-TAD channel in that it provides both secrecy and integrity protection to the data transferred between the server and the clients (e.g., KISS product keys).

7 Secure System Administration

This section describes how the KISS administrators perform local and remote operations using their TADs and remote managers. Unlike traditional KMS, our remote management mechanism introduces a very small TCB that consists of a thin terminal, the KISS hypervisor, the user-interface devices on KISS manager, and the simple TADs. In addition, it enables flexible administrative policies for better usability.

Secure Local Management. Administrators physically present at the KISS server or client connect the TADs directly with the machines to perform management. TADs first perform remote attestation to verify that the connected KISS machine is executing the desired hypervisor, software, and trusted paths. Thus, any command input (or KISS system output) is securely directed to (or displayed by) the KISS server/client software. The administrators also use the TADs to authenticate their command input, by allowing the KISS server/client to display the command input with its digest (a cryptographic hash, $H(input)$) to the administrators. The alleged digest $H(input)$ is sent to the TADs via untrusted USB connection. The administrator confirms that the digest value displayed on his/her TAD is identical to the one on the server/client display. Then, the administrator press a button on the TAD to generate an authentication blob (digital signature) on digest $H(input)$ with the Srv-/Cli-TAD channel keys. The KISS server/client software verifies this blob to ensure the authenticity of launched commands.

Secure Remote Management. Administrators not physically present at the KISS server or client leverage the KISS managers and the TADs to perform maintenance tasks. The KISS manager software is isolated from the untrusted regime, and connects with the

Category	Operations	Local or remote?	Quorum or any?	Manual or automatic?
1	server bootstrap, adding administrators	local	quorum	manual
2	server software and config update, removing administrators	either	quorum	manual
3	client bootstrap	local	either	manual
4	client registration, software and config update (e.g., change key usage control policy)	either	either	manual
5	server/client key life-cycle operations (e.g., key generation, distribution, usage)	either	either	either

Table 2. KISS System Operation Categorization.

user-interface devices via hypervisor-established trusted paths. The administrators not only use their TADs to authenticate the command input (the same as in local management), but also to verify the authenticity of the system output returning from the KISS server or client software. The KISS server/client generate similar authentication blobs for each of their responses, using the Srv-/Cli-TAD channel keys. The KISS manager software recomputes the digest $H(response)$, and displays it to the administrators via the trusted paths. It also forwards the digest and the authentication blob to the TADs. The TADs verify the authenticity of the blobs, and display the digest on the screen. If the two digests are identical, the administrators trust that the response indeed originated from the KISS server/clients. Note that our remote management mechanism can be extended to protect the secrecy of the command input and system output, and avoid the hash computation overhead and comparison efforts (Section 10).

Administrative Policies. KISS fully considers the balance between security and usability when making administrative policies. We categorize different system operations according to their administrative requirements, as is shown in Table 2.

In KISS, only three operations require the physical presence of administrators at the KISS server/client; the majority of operations can be performed remotely. In Category 1, server bootstrap and adding new administrators require the physical presence of a quorum of administrators. These two operations bootstrap cryptographic channels between TADs and the KISS server software and require our server bootstrap protocol (Section 6.1). Client bootstrap also mandates the physical presence of administrators, because administrators scan the TPM public key to their TADs.

In KISS, only a few operations mandate a quorum of administrators. We require all server-side administrative operations in Category 1 and 2 to be performed by an administrator quorum in an attempt to *prevent* malicious insider attacks on the KISS server. However, once the server audit log is bootstrapped, all subsequent client-side administrative operations in Categories 3 and 4 and server/client key life-cycle operations in Category 5 could possibly relax the quorum requirement, because we can always *detect* insider attacks by analyzing the audit log.

In addition, for efficiency and usability, all Category 5 operations can be automatically performed by the KISS server/client software, without the involvement of administrators. For example, once an authorized corporate application requests a new key, the

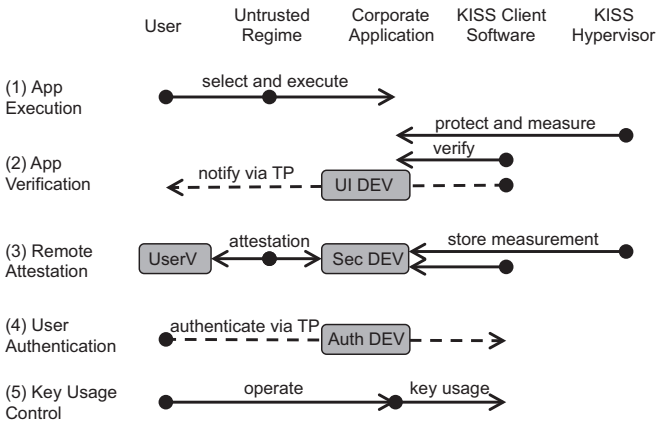


Fig. 5. Work flow of key usage control on KISS client. Dashed lines are interactions via trusted paths. UI, Sec, and Auth Dev are identical to those in Fig. 2. UserV denotes the users' dedicated verifier that can remotely attest to the KISS client.

KISS client software can immediately contact the server for the new key. These automatic operations are controlled by the administrator-configured key usage policies (see Section 8), and can be recorded in the server audit log (or similar audit logs on clients).

8 Fine-grained Key Usage Control

This section explains how the KISS client software and hypervisor performs fine-grained control of key usage. Figure 5 presents a typical workflow where a user executes a KISS-capable application that uses the cryptographic key generated by KISS.

Application Verification. The user selects the corporate application he/she intends to run via the untrusted regime (e.g., via a pop-up dialog by the OS). The OS loads and executes the selected corporation application and notifies the KISS hypervisor of the application execution. The hypervisor creates a corporate regime and protects the executed application in this regime. The hypervisor then measures that application and sends the measurement as the application identifier to the KISS client software. The software compares the received measurement with the known-good value in its application database and notifies the result to the user via trusted paths. Recall that the authorized application database in the KISS client software was configured during the client bootstrap and can be updated by the administrators via remote management.

The KISS-capable corporate applications are not legacy applications. They are developed to execute in corporate regimes, communicating with the hypervisor instead of the OS [12, 13]. Note that recent research [6] eases this development effort by allowing protected applications to securely use OS services. The corporate application should also be modified to communicate with the KISS client software for key usage. While allowing key usage control, this introduces context switch overhead between the application and the KISS client software. A corporate application can be a stand-alone

application (e.g., a KISS-capable document editor) or the security-sensitive modules of a legacy application that uses cryptographic keys (e.g., the ServerKeyExchange authentication module in an HTTPS server software). This is an application-specific design choice that depends on the application complexity (e.g., how the application is modularized and privilege-separated) and the strictness of the key usage control policy (application-wise or module-wise).

Remote Attestation. To trust the application verification results displayed in last step, and to defend against subtle user-oriented credential stealing attacks (e.g., tricking the user to input passwords), the users should leverage a small, dedicated device, called UserV, to attest that they are interacting with the correct KISS software and corporate applications. The UserV is similar to, but much simpler than TAD. The only task of the UserV is to perform standard remote attestation to the KISS hypervisor and client software. It does not generate or store any secrets (e.g., shared secrets or private keys). It merely needs one button to start the attestation, and a LED to display attestation results [25]. Upon successful remote attestation, the user verifies that the application displayed is the one that he/she intends to run. Otherwise, the user should stop interacting with the corporate applications to prevent any sensitive information leakage.

User Authentication. In order to use the corporate application, the user needs to authenticate to the KISS client software. If the authentication fails, the KISS hypervisor immediately teminates the corporate application. KISS can support all types of common authentication methods (knowledge, inherence, and ownership-based) and multi-factors authentication. For knowledge-based authentication (e.g., password, PIN) or inherence-based methods (e.g., fingerprint scanning, voice pattern recognition), the users should leverage the trusted paths between the authentication-relevant devices (e.g., keyboard, fingerprint reader) and the KISS client software. With the trusted paths, malware in the untrusted regime cannot intercept the users' credentials [1]. For ownership-based authentication, users usually carry certain authenticators (e.g., smart cards, security tokens) and rely on the embedded secrets to respond to the challenges of the KISS client software. No trusted path is needed between the authentication devices (e.g., smart card reader) and KISS client software. For all the authentication methods above, the KISS client software should be configured with necessary authentication information (e.g., password hash, fingerprint database, and keys to verify smart cards' responses) by the administrators during client bootstrap or remote management.

Key Usage Control. During execution, the corporate applications trigger key usage requests to the KISS client software via KISS hypervisor. The key usage requests can be driven by the users (e.g., the user wants to encrypt a confidential document) or by the application itself (e.g., the HTTPS web server software digitally signs its ServerKeyExchange messages). Upon receiving the key usage requests, the KISS client software knows the identifiers of the requesting application and the user. The KISS client software leverages the pre-configured access control policies to decide whether to approve or deny the requests. KISS supports flexible access-control policies with different granularity. It can perform simple ON/OFF key usage control. For example, KISS allows

[1] Even if the attackers have the users' credentials, they still need to physically be present at the KISS client to input the credentials. The KISS client software takes inputs directly from the hardware devices via trusted-paths, not from any software.

user Alice to use the authorized document editor to decrypt her own documents, but restricts other users who are using the same editor or Alice using different software (e.g., an email client, or a compromised document editor) from accessing the documents. It can also support more complicated policies, such as rate limiting, access time restriction, and role-based access control. The administrators decide the access control policies, configure them in the client software during bootstrap, and update the policies via remote management.

9 Security Analysis

This section analyzes potential attacks on KISS and our defense mechanisms.

System Bootstrap. During the system bootstrap, malicious administrators or malware on KISS server/clients may tamper with the code or configurations of the hypervisor and the KISS software. The benign administrators can detect this attack via TAD remote attestation. Malicious administrators may also launch Sybil attacks by creating bogus administrator accounts during the bootstrap process. As described in Section 6, the administrators visually check that all TADs display success messages. This confirms that the server/client software receives only the public keys of the participating TADs, not any bogus key.

Key Life-cycle Operations. Malware in the server/client untrusted regime may try to modify the KISS software code, interfere with its execution, or access the cryptographic keys generated or stored by the software. The KISS hypervisor prevents these attacks by protecting the code and data memory of the KISS software from the untrusted regime. When the KISS software is at rest, the cryptographic keys are protected by the TPM sealed storage. Only the same KISS software can unseal the keys; the malware or the compromised KISS software cannot. Malware attacks that compromise the client software to trigger unintended KISS server operations also fail, because the client private key for authenticating operation requests is sealed by the TPM.

System Administration. Any manual administrative operation requires at least one authorized TAD. The malware cannot steal the private keys in TADs, nor can it intercept other administrator credentials, such as bio-metric information or passwords, which are transferred to the KISS software via trusted paths, and/or Srv-/Cli-TAD authentic channels (Section 7). Similarly, the administration commands and system output are also transferred via trusted paths or Srv-/Cli-TAD channels. The attackers cannot modify any command or forge any system output. Though malicious administrators may use their TADs to execute operations that do not require the quorum, those operations are recorded in the server/client audit log and held accountable.

Key Usage Control. As described in Section 8, unauthorized applications and users cannot bypass the KISS hypervisor and the client software to use any cryptographic key. A malicious administrator may intentionally update the application and user database in the KISS client software to allow key mis-uses. However, this administrative operation is recorded in the client audit log and held accountable. The malware cannot steal users' authentication credentials, because those credentials are delivered to the KISS client software via trusted paths. The users also verify that they are communicating with the authentic KISS client software before inputting their authentication credentials.

10 Discussion

This section discusses the KISS system extensions that provide higher security guarantees and address some real-world application issues (e.g., cloud computing).

Administrative Operation Secrecy. Section 7 describes how KISS protects the authenticity of administrative inputs and system outputs. We could extend KISS to protect input/output secrecy by establishing encryption keys for Srv-/Cli-TAD channels, and an extra trusted path on KISS manager between the manager software and the USB controllers that connects the TAD. Note that this trusted path also avoids the hash computation overhead and comparison efforts described in Section 7, because it protects the authenticity of data between the TAD and the manager software.

TPM 2.0 Enhanced Authorization. The TPM 2.0 library specification [22] is currently under public review. It supports enhanced authorization by allowing the construction of complex authorization policies using multiple authorization qualifiers (e.g., password, command HMAC, PCR values, NVRAM index values, and TPM time information). KISS can reduce its TCB by offloading some authorization checking to TPM 2.0, given that it can securely collect the authorization information, deliver it to the TPM, and protect it from the untrusted OS. However, it is not clear how the performance of TPM authorization checking compares to that of the KISS software.

Compatibility to Cloud Computing. The KISS hypervisor is a small, dedicated hypervisor that runs on bare metal. If the KISS servers and clients are deployed on an enterprise private cloud, we could consider (1) integrating KISS hypervisor with the full-functioning the hypervisor/VMM or (2) adding nested virtualization support [24] to KISS hypervisor and running the full-functioning hypervisor/VMM upon it. Option (1) has much larger TCB, but has better compatibility and performance than option (2).

11 Related Work

We review the state-of-the-art key management systems and related technologies. The first category of KMS solutions are **software-based solutions**, such as OpenSolaris Crypto KMS Agent Toolkit [14], IBM Tivoli Key Manager [8], and StrongKey open-source KMS software [19]. These rely on process isolation, user privilege control, and file permissions provided by the OS to protect cryptographic keys and control the applications' access to them. Their implementation of trusted paths for administrators is based on the OS services (e.g., Ctrl+Alt+Del command or trusted window manager). Compared with KISS, the software-only approaches are more cost-effective and easier to deploy on commodity computers (e.g., no hypervisor, work with legacy corporate applications, no security hardware requirement). However, they rely heavily on the large OS and thus fail to provide the same level of security assurance as KISS.

An alternative is leveraging high profile **HSMs** [20, 18, 15, 7, 17]. An HSM provides hardware-level tamper-resistant protection to cryptographic keys and algorithms for both run-time and at rest, while KISS provides hypervisor-based software isolation for keys and algorithm during run-time, and TPM level hardware protection for keys at rest. For performance, an HSM may employ customized hardware engine to accelerate cryptographic algorithms. It is more efficient than KISS and the software-only

solutions. The downside of the HSM is that it fails to provide the same secure level of key usage control as in KISS, as we have explained in Section 1. Indeed, the KISS system can be extended to serve as the key usage control front end of the HSM, which may achieve the benefits of both systems. For system administration, some high-end HSMs [20, 18] achieve the same level of security guarantees as KISS (e.g., quorum control, trusted paths using on-HSM I/O devices, remote management using administrator devices). However, their administrator devices introduce larger TCB than KISS (e.g., complicated key management software stack for interpreting commands and operation results). The HSM administrators usually blindly trust the devices, and have no intuitive way to verify their software status.

There are research proposals that seek to offer similar protections for user credentials in the key management systems. Wallet-based web authentication systems (e.g., [5]) isolate user credentials in an isolated domain (e.g., a L4 process upon L4 Micro-kernel) during run-time and protect the credentials at rest by TPM-based sealed storage. They only allow authenticated websites to access their own credentials. These systems have a reasonable TCB size, but do not provide fine-grained and flexible key usage control as in KISS (e.g., user-based control). Bugiel and Ekberg [2] propose a system that only allows the application to access its own credentials (protected in mobile trusted module). The On-board Credentials (ObC) [10] approach enables an isolation environment (like KISS) for both third-party credential algorithms/applications and credentials, on smartphones and conventional computers. However, one faces multiple challenges extending these systems for corporate key management. For example, ObC approach lacks protection mechanisms against malicious administrators and do not support trusted paths for administrator management. PinUP [4] binds files to the applications that are authorized to use them by leveraging the SELinux capability mechanisms. This suggests that PinUP introduces a larger TCB to provide security assurance on par with KISS.

12 Conclusion

In this paper, we leverage widely-deployed trusted computing techniques to design a trustworthy key management system architecture. KISS aims to reduce cost by relying solely on commodity computer hardware, and minimize the system TCB by the thin-hypervisor-based design and lightweight administrator devices. KISS is the first key management system to support fine-grained control of key usage. KISS is bootstrapped and operated in the face of software attacks from malware in the OS and insider attacks from malicious administrators. KISS provides user-verifiable trusted paths and simple dedicated external devices for secure system administration. KISS showcases the benefits of applying trusted computing techniques to designing trustworthy systems. KISS offers trustworthy key management systems at a price point that enables wide-spread adoption beyond the security-sensitive financial or governmental institutions.

Acknowledgments. We are grateful to the reviewers and Aziz Mohaisen for their insightful suggestions. We also want to thank Geoffrey Hasker, Yueqiang Cheng, and Miao Yu for stimulating conversations and valuable feedback.

Bibliography

[1] AMD. AMD64 architecture programmer's manual. No. 24594 rev. 3.19, 2012.

[2] S. Bugiel and J. Ekberg. Implementing an application-specific credential platform using late-launched mobile trusted module. In *Proc. ACM STC*, 2010.

[3] Y. Cheng, X. Ding, and R. H. Deng. DriverGuard: A fine-grained protection on I/O flows. In *Proc. ESORICS*, 2011.

[4] W. Enck, P. McDaniel, and T. Jaeger. Pinup: Pinning user files to known applications. In *Proc. ACSAC*, 2008.

[5] S. Gajek, H. Löhr, A. Sadeghi, and M. Winandy. Truwallet: trustworthy and migratable wallet-based web authentication. In *Proc. ACM STC*, 2009.

[6] O. S. Hofmann, S. Kim, A. M. Dunn, M. Z. Lee, and E. Witchel. Inktag: secure applications on an untrusted operating system. In *Proc. ASPLOS*, 2013.

[7] HP. Enterprise Secure Key Manager. `http://h18006.www1.hp.com/products/quickspecs/13978_div/13978_div.PDF`.

[8] IBM. Tivoli Key Lifecycle Manager. `http://www-01.ibm.com/software/tivoli/products/key-lifecycle-mgr`.

[9] Intel. Intel trusted execution techonology. No. 315168-008, 2011.

[10] K. Kostiainen. *On-board Credentials: An Open Credential Platform for Mobile Devices*. PhD thesis, Aalto University, 2012.

[11] A. Matrosov, E. Rodionov, D. Harley, and J. Malch. Stuxnet Under the Microscope. `http://www.eset.com/us/resources/white-papers/Stuxnet_Under_the_Microscope.pdf`.

[12] J. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. TrustVisor: Efficient TCB reduction and attestation. In *Proc. IEEE Symp. on Security and Privacy*, 2010.

[13] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for TCB minimization. In *Proc. EuroSys*, 2008.

[14] Oracle. Opensolaris project: Crypto kms agent toolkit. `http://hub.opensolaris.org/bin/view/Project+kmsagenttoolkit/WebHome`.

[15] Oracle. Oracle Key Manager. `http://www.oracle.com/us/products/servers-storage/storage/tape-storage/034335.pdf`.

[16] B. Parno, J. R. Lorch, J. R. Douceur, J. Mickens, and J. M. McCune. Memoir: Practical state continuity for protected modules. In *Proc. IEEE Symp. on Security and Privacy*, 2011.

[17] RSA. RSA Data Protection Manager. `http://www.emc.com/security/rsa-data-protection-manager.htm`.

[18] SafeNet. SafeNet hardware security modules. `http://www.safenet-inc.com/products/data-protection/hardware-security-modules-hsms/`.

[19] StrongAuth. StrongKey SKMS. `http://www.strongkey.org`.

[20] Thales. Thales hardware security modules. `http://www.thales-esecurity.com/en/Products/Hardware%20Security%20Modules.aspx`.

[21] Trusted Computing Group. TPM specification version 1.2, 2009.

[22] Trusted Computing Group. Trusted platform module library family "2.0", 2011.

[23] VASCO. Diginotar reports security incident. `http://www.vasco.com/company/about_vasco/press_room/news_archive/2011/news_diginotar_reports_security_incident.aspx`, 2011.

[24] F. Zhang, J. Chen, H. Chen, and B. Zang. Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proc. ACM SOSP*, 2011.

[25] Z. Zhou, V. Gligor, J. Newsome, and J. McCune. Building verifiable trusted path on commodity x86 computers. In *Proc. IEEE Symp. on Security and Privacy*, 2012.