Secure Cryptography Infrastructures in the Cloud

Dawei Chu*‡, Kaijie Zhu§, Quanwei Cai*†, Jingqiang Lin*†‡, Fengjun Li♯, Le Guan¶, Lingchen Zhang*†
*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences
†Data Assurance and Communication Security Research Center, Chinese Academy of Sciences
§National Digital Switching System Engineering & Technological R&D Center, CHINA
‡School of Cyber Security, University of Chinese Academy of Sciences
♯Department of Electrical Engineering and Computer Science, The University of Kansas, USA
¶Department of Computer Science, University of Georgia, USA

***Abstract*—Information systems are deployed in clouds as virtual machines (VMs) for better agility, elasticity and reliability. It is necessary to safekeep their cryptographic keys, e.g., the private keys used in TLS and SSH, against various attacks. However, existing virtualization solutions do not improve the cryptography facilities of in-cloud systems. This paper presents *SECRIN*, a secure cryptography infrastructure for VMs in the cloud. SECRIN is composed of *a*) virtual cryptographic devices implemented in VM monitors (VMMs), and *b*) a device management tool integrated in the virtualization management system. A virtual device receives requests from VMs, computes with cryptographic keys within the VMM and returns results. The keys appear only in the VMM's memory space, so that they are kept secret even if the VMs were compromised. With the management tool, the operator of virtualization management systems assigns virtual cryptographic devices to a VM as well as other resources, while the tenant (or owner) of a VM still holds proper controls on the keys. The virtual devices work compatibly with live migration, and the cryptographic computations are not interrupted when the VMs are moving from a host to another. We develop the SECRIN prototype with KVM-QEMU and oVirt. Experimental results show that, it works compatibly with existing virtualization solutions, provides reliable cryptographic computing services for applications, and is secure against attacks happening in VMs.**

I. Introduction

Various information systems are deployed as virtual machines (VMs) on Amazon EC2, Microsoft Azure, Alibaba Aliyun and other virtualization platforms. Virtualization platforms maintain computation, network and storage resources for tenants with better agility, elasticity and reliability; however, the cryptography facilities of VMs are not well improved in existing virtualization solutions. Meanwhile, deployed in virtualized (or conventional) environments, information systems still depend on cryptography (in particular, the semantic security of cryptographic algorithms and the confidentiality of cryptographic keys) to provide secure services. While secure algorithms are generally available, it is difficult to ensure the confidentiality of keys. For example, TLS and SSH servers shall protect their private keys against various attacks [1]–[6]; otherwise, the attackers would exploit the compromised keys to decrypt messages or impersonate the owners of the keys.

Dawei Chu and Kaijie Zhu are co-first authors, and Quanwei Cai is the corresponding author (Email: caiquanwei@iie.ac.cn). This work was partially supported by National Natural Science Foundation of China (Award 61772518), National Key R&D Plan of China (Award 2017YFB0802100), NSF DGE #1565570, NSA SoS Initiative #H98230-18-D-0009 and the Ripple University Blockchain Research Initiative.

Virtualization technologies and cloud solutions offer the potentials to mitigate the threats to cryptographic keys and build cryptography infrastructures for all VMs in the cloud. Firstly, if a virtualization platform builds cryptography services for VMs, the keys will be decoupled from the VMs' memory space. All computations with keys will be performed in the VM monitor (VMM), so the attack resilience is significantly improved – if a VM is under attacks [1]–[4] or even completely compromised, the keys are still kept secret. Secondly, compared with conventional solutions that keep cryptographic keys within the VMs, hosting the keys in the VMM does not introduce extra threats, because these sensitive data are always accessible to the VMM when they are in VMs. Finally, cryptographic facilities are managed in an agile and elastic manner (as well as other resources), and reliable cryptography services benefit from live migration of the virtualization platform.

We present *SECRIN*, a secure cryptography infrastructure in the cloud. It provides cryptography services for VMs with enhanced protections on the keys. SECRIN takes advantage of the isolation mechanism from virtualization technologies, and the cryptographic computations are implemented as virtual devices, called *virtio-ct*. The virtual devices are implemented within VMMs, so that the keys appear as plaintext only in the VMM's memory space and never in VMs.

To improve its usability, SECRIN integrates the management of cryptographic devices into the virtualization management system. In a cloud platform, all computation, network and storage resources are coordinated by the virtualization management system [7]. The virtual devices of SECRIN are managed in the same way as other cloud resources, while the tenants hold proper controls on the keys. When a virtio-ct device is not mounted, its keys are encrypted by a password. Once a (remote) tenant is booting his/her VM that is configured with a virtio-ct device, he/she is prompted to enter the password. This password is securely transported to the host, to decrypt the key file and then to activate the virtual device. SECRIN also works compatibly with live migration, to provide reliable services. When it is serving for the applications in a VM and the VM is moving from a host to another, the cryptographic computations of SECRIN (and the applications in this VM) are not interrupted.

We develop the prototype system with open-source VMMs and virtualization management systems, i.e., KVM-QEMU

978-1-7281-0962-6/19/$31.00 ©2019 IEEE

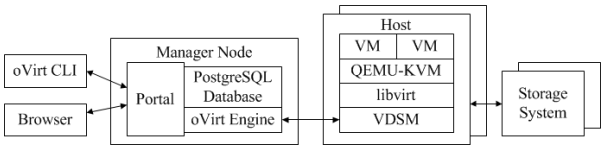

Fig. 1. oVirt Structure

and oVirt [8]. By intensive experiments, we show that SECRIN works compatibly with existing virtualization solutions, provides secure cryptography services for various applications.

II. Background on Virtualization

KVM-QEMU is a popular VMM solution, consisting of a Linux *kernel* module called KVM (kernel-based virtual machine) and a *user-space* program called QEMU. KVM initializes CPU hardware and provides VM management interfaces via the `ioctl` system call, such as mapping the memory of a VM, and assigning virtual CPUs. QEMU invokes the interfaces to run VMs, each of which is a user-space process.

VMMs emulate isolated peripheral devices for each VM. virtio [9] is a framework of device virtualization for KVM-QEMU. It presents an abstraction layer called *virtqueue*, i.e., virtual queues connecting a *front-end driver* that runs in VMs to the corresponding *back-end driver* in the user-mode QEMU. Front-end and back-end drivers cooperate to emulate devices.

A virtualization management system [7] coordinates lots of VMMs and VMs on the physical computation, network and storage resources of the platform. Generally, a virtualization platform consists of the following hardware:

Manager Node. A manager node provides the operators and the tenants with portals for authentications and operations. It communicates with hosts to execute requested operations.

Host. Each host runs a VMM to manage the VMs on it. A platform may consist of hundreds, or even thousands of hosts.

Storage. It stores VM images, snapshots and other data.

oVirt [8] is an open-source virtualization management system, built on top of KVM-QEMU. It is widely used in public and private clouds. The oVirt system consists of the following components, as shown in Figure 1: (1) *oVirt engine* running on the manager node, to manage resources in the platform; (2) *virtual desktop server manager* (VDSM) on each host as the agent of oVirt engine, to execute VM life-cycle operations via libvirt, and configure networks and storage; and (3) *libvirt* [10] on each host to interact with KVM-QEMU to run VMs.

oVirt engine issues VM-operation commands to VDSM. It also handles user authentication, and provides user interfaces (UIs) for the oVirt operators (not the root user of manager nodes or hosts) and tenants. Two UIs are provided, a web-based interface and a command-line one named *oVirt CLI*. It mounts a PostgreSQL database to store the configurations.

VDSM accepts the XML-RPC invocations from oVirt engine, and then configures hosts, networks and storages, or communicates with libvirt in XML messages to maintain the VMs on its host. libvirt converts the XML messages into QEMU commands to create, start, stop, save and restore VMs.

III. SECRIN: SEcure CRyptography INfrastructures in the Cloud

This section presents the assumptions and the design goals of SECRIN. Then, the system architecture is introduced.

A. Threat Model and Assumption

SECRIN aims to protect cryptographic keys against the attacks happening in VMs. The VM that invokes the SECRIN services, might be completely compromised; i.e., attackers could exploit vulnerabilities [1]–[4], [11] to access sensitive data or even run privileged programs arbitrarily in the VM.

VMMs are assumed to be trustworthy. Isolation is enforced to prevent a VM from accessing the VMM's memory data. We assume that it is free of VM escape (or privilege escalation) vulnerabilities [12], [13]. This assumption can be ensured by the assistance of hardware virtualization features and reducing the size of trusted codes in VMMs [14], [15].

SECRIN provides cryptography facilities, integrated with the virtualization management. The virtualization management system is also assumed to be trustworthy. There is no operation interface in the virtualization management for the operators to disclose sensitive data of virtual cryptographic devices (i.e., the passwords and the keys in plaintext). First of all, SECRIN works well, when the operators do not know the passwords or the keys. Note that, an operator of virtualization management is usually not the root of manager nodes or hosts, and a cloud service does not grant unnecessary privileges to the operators. The integrity of the platform is maintained by root users that have to be trusted, and unauthorized modification is prevented. We assume that the operators cannot break the integrity of trusted binaries of the management system or VMMs.

The threat model assumes the same conditions as most public cloud services. Amazon S3 introduces three options to encrypt data in clouds [16], [17]: *a*) in the client side before being uploaded, *b*) in the server side using the keys managed by the cloud, called SSE, or *c*) in the server side but using tenant-provided keys, called SSE-C. Similar modes are supported by other clouds [18], [19]. We extend these service modes, from cloud storage to cryptography services in the cloud. The first option requires tenants to maintain cryptography facilities by themselves; that is, a tenant runs its own cryptographic device. SSE-C follows the same assumption as SECRIN in the public cloud: Plaintext cryptographic keys are used by trusted cloud binaries but unknown to the operators. Finally, SECRIN in private clouds follows a mode similar to SSE, where the cryptography facilities are fully controlled by the operator.

B. Design Goal

SECRIN aims to provide secure cryptography infrastructures for VMs in the cloud, easy to be used and managed. *Provide cryptographic computation services for VMs, with enhanced protections of the keys.* The cryptographic keys are kept secret even if a VM was completely compromised. Attackers might compromise a VM and invoke the SECRIN services arbitrarily, but not harm the confidentiality of keys.

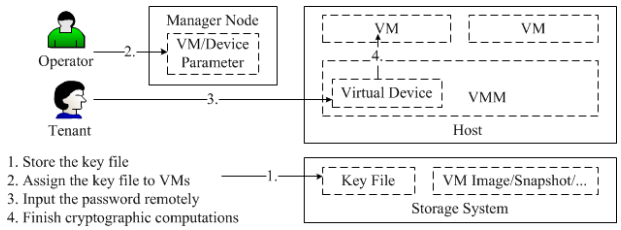

Fig. 2. SECRIN Overview

Integrate the cryptography infrastructure into the virtualization management, following the same service modes of cloud resources. The cryptographic devices are managed in the virtualization management system in a way similar to other cloud resources. The operator assigns virtio-ct devices to VMs. From the tenants' view, the SECRIN services are (implemented as) common peripheral devices mounted in a VM. The cryptography facilities support live migration, and active virtio-ct devices are migrated across hosts.

Tenants and operators cooperatively control the virtual cryptographic devices. Each key file encrypted by (an AES key derived from) the tenant's password, is stored in the storage system. To mount a virtual cryptographic device in a VM, the operator of virtualization management configures the key file for the VM, and the tenant inputs the password to decrypt the key file when the VM is booting. Such control prevents a VM from accessing other tenants' cryptographic devices and avoids incidents due to the operators' misoperations.

C. System Architecture

SECRIN is designed based on the general framework of virtualization platforms. It is composed of *a*) *virtual devices* implemented in VMMs, and *b*) a *management tool* integrated in the virtualization management system. A virtualization management system provides operation interfaces and channels for operators and remote tenants, so we build the SECRIN management tool on top of these interfaces and channels.

Figure 2 shows the life-cycle of a virtio-ct device. Firstly, the tenant generates his/her key file in a secure machine, and sends it to the cloud platform. This file contains cryptographic keys, encrypted by a password strong enough to resist brute-force attacks. Then, the operator of virtualization management assigns this key file to a VM belonging to the tenant. This configuration instructs VMMs to initialize the cryptographic device when booting the VM. Finally, when the tenant is booting a VM, the virtualization management system prompts him/her to input the password. The VMM receives the password to decrypt the key file and initialize the virtio-ct device, which provides services only for this VM.

Cryptographic Key and Password. Cryptographic keys at rest are encrypted as files in the storage system. When the VMM initializes a virtio-ct device, it reads the key file and decrypts it with the password. The virtualization management system transports file paths and passwords to VMMs, along with other configurations. The typical steps as follows. The operator configures a VM by setting the file path of keys, based on the ownership of VMs and cryptographic keys. When a tenant is starting a VM configured with a virtio-ct device, he/she is prompted to input the password, via a secure channel connected to the VMM but unaccessible to VMs.

In addition to the typical mode in public clouds as above, SECRIN supports another mode to initialize the devices: Both key file paths and passwords are input by the operator. It is for private clouds, where the operator is responsible for all VMs' cryptography facilities.

Virtual Cryptographic Device. Virtual devices compute with the keys in the back-end drivers, i.e., in the VMM's memory space that is unaccessible to any VM. The isolation among multiple devices is enforced by the virtual device solutions [9]. When the back-end driver of virtio-ct devices is launching an instance (i.e., the device is being activated for a VM), the corresponding key file is decrypted and the keys are assigned to this instance only. The messages from the front-end driver of the VM are delivered only to this instance of the back-end driver, and vice versa. So a key file only serves the VM that it belongs to, as configured in the system.

In the case of live migration, the running state of back-end drivers is migrated between hosts, including the keys and intermediate states of cryptographic computations. So the cryptography services are not interrupted by live migration.

IV. Implementation

We build the prototype system with oVirt and KVM-QEMU.

A. Drivers of virtio-ct

The virtio-ct devices are implemented on virtio [9]. Each virtio-ct device is a virtual PCI device, logically attached with a number of tokens. Each token corresponds to a key. We borrow the source code of OpenSSL to finish the cryptographic computations in the back-end driver. The front-end driver in VMs and the back-end in QEMU, consist of about 2,000 lines of C code (LoC), excluding the blocks from OpenSSL.

On discovering a virtio-ct device, the front-end driver instantiates necessary data structures, and communicates with the VMM to initialize the device. virtio assigns a default management channel to each virtio-ct device. This management channel is used to initialize the virtio-ct device by adding tokens. The back-end driver sends necessary information for the VM to add a token, including the algorithm (RSA or AES), a user-friendly string of the token, and its public key (only for an RSA token). The prototype supports only AES and RSA, and we will support more algorithms in the future.

When the VM is notified of a new token, another channel is allocated exclusively for this token. All applications in the VM invoke the SECRIN services via the front-end driver, and the front-end and back-end drivers communicate over this dedicated channel for cryptographic computations. Each request to a token, indicates the operation (encryption, decryption, signing or verification), the cipher mode (ECB, CBC, OFB or CFB) for AES, and the padding (PKCS1 or OAEP) for RSA. On receiving a request, the back-end driver computes with the key. If random bits are needed, the OpenSSL pseudorandom

number generator is used. The response always returns a status code. If there is no error, the result bytes are written into the buffer provided by the front-end driver.

B. Management and Initialization of virtio-ct

We integrate the management of virtio-ct into oVirt, with the following functions: (1) configure and assign virtio-ct devices to VMs, and (2) transport virtio-ct parameters to hosts, to initialize the devices. We introduce about 200 LoC in oVirt CLI for tenants to input passwords interactively, and about 300 lines of scripts in VDSM to handle the virtio-ct parameters.

virtio-ct Configuration. It is implemented in oVirt engine. We define a custom property named `virtio-ct-key` for VMs. When a virtio-ct device is assigned to a VM, the operator sets the VM's `virtio-ct-key` property and the property value is the path of the key file. In the public-cloud mode, file paths and passwords are input by operators and tenants, respectively. In the private-cloud mode, both the password and the file path are entered by the operator.

The virtio-ct parameters (i.e., file paths, passwords and key files) are transported among oVirt UIs, oVirt engine, VDSM, and libvirt. The UI channels of oVirt engine, either web pages or oVirt CLI, are protected by TLS. The operator uses these channels to configure and assign virtio-ct devices. We extend oVirt CLI as follows for tenants to input passwords interactively. When a tenant requests to start a VM, CLI reads the VM's configuration and prompts the tenant to input the password after checking that a virtio-ct device is assigned. Then, the password is collected by oVirt engine as a *run-once* temporary parameter, not saved in the PostgreSQL database.

Then, oVirt engine sends the file path and the password to VDSM in a VM-start XML-RPC request, as well as other configurations. This is also protected by TLS.

virtio-ct Initialization. VDSM and libvirt work cooperatively to prepare parameters, for QEMU to activate the device.

On receiving the VM-start XML-RPC invocation from oVirt engine, VDSM generates the corresponding XML message for libvirt. VDSM allows customized scripts by hooking a certain point in the VM life-cycle. We hook the `before_vm_start` point, to handle the `virtio-ct-key` property. This hook happens, after VDSM finishes the XML message for libvirt and before it calls libvirt to start the VM. The scripts handle the XML message with the file path and password, so that it is ready for libvirt to turn it into correct QEMU commands. We also hook `after_vm_start`, to clean the passwords.

If an error appears when a virtio-ct device is initialized (e.g., incorrect password or no key file), the failure is fed back to VDSM, and then to oVirt engine.

C. Cryptographic Computation Service API

Two categories of APIs are supported by virtio-ct devices. The **sysfs** attributes are used to identify each device. Using **sysfs**, the key type, the name and the public key are exported on the VM. Figure 3 shows two tokens, identified as `ct0token0` and `ct0token1`. Meanwhile, cryptographic computations are exported to the user space via the `ioctl` system call, which sends requests to a device and receives responses.

```
[root@localhost /]# ls -l /dev/virtio-ct*
crw------- 1 root root 249, 0 Dec  2 13:26 /dev/virtio-ct0token0
crw------- 1 root root 249, 1 Dec  2 13:26 /dev/virtio-ct0token1
```

Fig. 3. `virtio-ct` Device

This `ioctl`-based interface is further encapsulated into an OpenSSL ENGINE, to make it easier to integrate virtio-ct devices into existing applications. The OpenSSL ENGINE mechanism dynamically loads an external cryptographic implementation that complies with its interface specification, to finish the computations. With this ENGINE, an application still calls the OpenSSL interface, while OpenSSL sources out its cryptographic computations to virtio-ct devices.

D. Application

We rebuild two prevailing applications on the prototype, i.e., Apache HTTPS server and dm-crypt full disk encryption. Apache invokes asymmetric cryptographic computations (i.e., RSA decryption in TLS handshakes) from the user space, and dm-crypt invokes symmetric computations (i.e., AES encryption/decryption) in OS kernel. Less than 10 LoC are modified in these applications. These applications evaluate SECRIN in different typical scenarios (i.e., asymmetric and symmetric cryptography, in the user space and the kernel space, for a small amount and a large mass of data).

Apache adopts the virtio-ct device as an OpenSSL ENGINE. RSA computations in the TLS handshake are directed to virtio-ct devices, while the symmetric cryptographic computations are still finished in the VM. dm-crypt uses virtio-ct devices as its AES module for disk data encryption/decryption. dm-crypt originally calls the generic cryptographic functions in Linux kernel, so we modify Linux kernel to export the AES functions of virtio-ct to dm-crypt.

E. Live Migration

Live migration works well with SECRIN. When the VM with active virtio-ct devices is migrated between hosts, the in-progress SECRIN services operate continuously. The virtio-ct back-end drivers are installed on every host, and the driver state is sent automatically. Then, the states of front-end and back-end drivers are consistently transported to the destination.

V. Security Analysis

This section discusses the security guarantees of SECRIN. We assume that SECRIN is correctly implemented and the system integrity is ensured. First of all, the cryptographic keys and the tenants' passwords to encrypt/decrypt the keys, are in the memory space of VMMs or virtualization management systems that is unaccessible from VMs. Thus, even if the VM was completely compromised, the keys are still kept secret. It depends on the isolation by VMMs, because these sensitive data are processed within the memory space of VMMs.

SECRIN does not modify any VMM binary with ring -1, so it does not introduce extra risks to the virtualization platform.

The virtio-ct device consists of the back-end driver in the user-mode QEMU and the front-end driver in VMs; the passwords are handled by oVirt CLI, oVirt engine, VDSM hook scripts and libvirt, which are executed in the user space of hosts.

In the next, we examine the threats from different aspects and also the remaining attack surfaces.

Threats from Hosts. The VMM and any processes with root privileges in the host can access the cryptographic keys and the passwords. Due to the isolation enforced by the host OS, unprivileged processes except QEMU cannot access these sensitive data. So we have to assume trustworthy VMMs and host OSes; that is, these binaries do not leak sensitive data.

Patches are enforced to protect the passwords in the host as follows. The password is included in the VM-start QEMU command as an argument, and the command of each process exists in `procfs`, the `proc` file system. We modify QEMU with about 20 LoC: The password is replaced with '*' as the argument. So `procfs` or unprivileged commands based on `procfs` (e.g., `ps` and `top`), cannot disclose the passwords. This approach is widely adopted in open-source software such as MySQL. Besides, the QEMU commands sent by libvirt, are executed by `fork` and `execve`, and not logged in the shell history accessible to unprivileged users.

Threats from Virtualization Management Systems. In Section III-A, we assume that *a*) root privileges are held only by trustworthy persons, and *b*) the (curious) operators are non-root unprivileged users on the manager node and hosts. Such operators cannot break the integrity of VMMs, host OSes, or the virtualization management system (including the VDSM hook scripts); otherwise, they will insert malicious binaries to extract keys or passwords. The operators operate oVirt well, without accessing these sensitive data of SECRIN via regular operation interfaces.

The passwords are processed on manager nodes and hosts, and recorded in the log modules of several components. By setting the log level, we ensure no passwords in the logs of oVirt engine and VDSM. This setting is changed only by root privileges on the manager node or hosts. The log of libvirt also contains the passwords and it cannot be disabled by tuning the log level, so we modify libvirt to ensure no password is logged.

Threats from Networks. Attackers might attempt to obtain the password and the key file over networks. In SECRIN, the password is transported over TLS, from oVirt CLI to oVirt engine, then to VDSM. Meanwhile, the keys are transported during live migration, protected by TLS or SSH [10]. So the network attacks are defeated.

Remaining Attack Surfaces. Physical attacks [5], [6] and side-channel attacks [20]–[22] might compromise the keys. These attacks are out of the scope of this work, and countermeasures are available on hosts [6], [22]–[24].

VI. Evaluation

This section presents the experiments on security and efficiency. The prototype consists of: *a*) a manager node, *b*) two Dell PowerEdge R820 servers as the hosts, each of which runs with 4 Intel Xeon E5-4607 v2 CPUs and 32GB RAM, and *c*) an NFS server as the storage system. These machines are connected in 1Gbps Ethernet. We install oVirt v3.5, libvirt v0.10.2, KVM v0.12.1 and QEMU v1.7.1. One VM is created with 4 vCPUs and 4GB RAM, and its OS is CentOS v6.6.

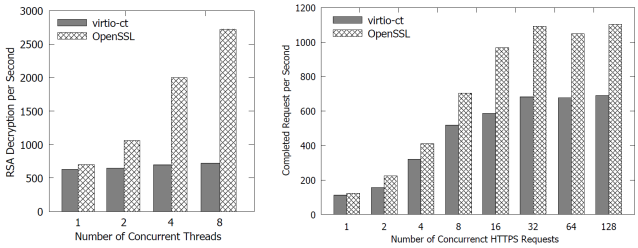

Fig. 4. RSA Performance of virtio-ct

Apache and dm-crypt run in the VM. Apache uses OpenSSL v1.0.2d and the virtio-ct device alternatively as its RSA module, and dm-crypt alternatively invokes generic AES functions in Linux kernel and virtio-ct as its AES engine. virtio-ct uses OpenSSL v1.0.2d in its back-end driver. The keys are 2048-bit RSA or 128-bit AES. During the HTTPS experiments, another machine runs Apache Benchmark to issue HTTPS requests.

A. Security

Experiments were conducted to ensure no copy of keys in VMs. When the virtio-ct device is being used (i.e., an infinite loop requests RSA and AES decryption), we executed **dump-guest-memory** to dump the VM's memory, and **info registers** to obtain the register contents. Then, the KeyFinder tools [25] searched for the patterns of AES round keys and DER-encoded RSA keys on dumped images. We also employed **bgrep** to match the keys in the images. We dumped the memory and registers for 60 times, but no occurrence of keys was found. Contrarily, when we ran the tools on the memory dump of the QEMU process, all keys were recovered each time.

B. Efficiency

RSA. We compare the RSA computations by virtio-ct with OpenSSL in the user space. Figure 4 shows the speed of native RSA computations, when a user-space program requests RSA decryption at different levels of concurrency. When the concurrency level is 1 (i.e., single thread), the speed of virtio-ct is comparable. However, in the multi-threaded case, the speed of OpenSSL increases while virtio-ct almost keeps unchanged. In the current QEMU design, there is a global mutex that synchronizes core codes across the threads that handle I/O events; therefore, for each VM only one thread is reserved to handle I/O events accessing virtio-ct or other virtual devices. This limitation is for each VM, not for multiple VMs on one host. One of our future works is to allocate multiple threads that are independent from the QEMU context as SPICE [26], so that the cryptographic computations are executed in parallel.

We measure the performance, when virtio-ct and OpenSSL are alternatively served as the RSA engine of Apache HTTPS servers. The client sends 10,000 HTTPS requests for a 5KB web page, at different concurrency levels. As shown in Figure

4, the number of HTTPS requests served does not reach the maximum speed of native RSA computations, either for virtio-ct or OpenSSL, due to the network protocol overheads. As the concurrency level increases, the speeds of both virtio-ct and OpenSSL increase; when there are 128 concurrent clients, the speed of virtio-ct is about 62% of OpenSSL.

AES. We first compare virtio-ct with two AES modules: The first one is a common OpenSSL-based program in the user mode, the second runs AES encryption in Linux kernel that is invoked via the `ioctl` system call, and virtio-ct is invoked via both `ioctl` and virtio. A single-threaded program calls AES encryption in an infinite loop, and each call includes k blocks of encryption. All of them are based on the same library of OpenSSL. In Table I, the native AES speed of virtio-ct is much less than OpenSSL. It is explained that, compared with AES encryption, the call-path overhead of virtio-ct is not negligible. This overhead also explains that, as the number of blocks per call increases, the speed of virtio-ct increases more remarkably than other implementations in VMs. These results show almost the most serious impact of efficiency by replacing the conventional cryptography implementations with virtio-ct: *a*) AES is one of the fastest cryptographic primitives, so the call-path impact of virtio-ct will become more negligible when heavier operations are invoked; and *b*) the program in this case consists of only cryptographic computations, so when a practical application is running on top of SECRIN, other processing will mask the impact. The evaluation results of RSA in Figure 4, are consistent with this inference.

We then measure the efficiency of virtio-ct, integrated in dm-crypt for disk encryption. Three disks are mounted: The first is plaintext, the second is encrypted with the default option of dm-crypt, and the last one is encrypted by virtio-ct. IOzone measures the file throughput, and Table II shows the results. The throughput with virtio-ct is about 450 kB/s.

C. Live Migration

We migrate the VM, when HTTPS requests are being sent to Apache in the VM. 10 times of live migration are finished in one hour, and no HTTPS request fails. The average time to finish live migration is nearly equal: 120.43s (virtio-ct) vs. 120.38s (OpenSSL). Besides, the speed of HTTPS requests completed decreases to about 70%, compared with the results in Figure 4 due to live migration, for either virtio-ct or OpenSSL.

VII. Related Work

vTPM [27] is built in VMMs with a trusted platform module (TPM), to present TPM functions to multiple VMs. CaaS [28] builds a special VM on Xen, to provides TPM-based functions for other VMs. SECRIN, vTPM and CaaS shares the same spirit that builds isolated cryptography services to protect keys, but we provide more flexible interfaces. Besides, vTPM and SECRIN support live migration [29], while CaaS does not.

Virtualization-based approaches are proposed to implement cryptosystems. **qemu-img** encrypts the VM images, and a user needs to input the password in hosts before the VM is started. TreVisor [30] integrates an AES engine in VMMs to implement transparent full disk encryption. These solutions are designed only for disk encryption, and not integrated in the virtualization management as common cryptography facilities in the cloud.

Intel SGX is a CPU extension to build user-space enclaves for sensitive data. Intel SGX prevents unauthorized access from OSes or VMMs to the user space, while SECRIN employs the isolation of VMMs to build security services for VMs. Therefore, Intel SGX can be utilized to build applications against compromised VMMs, while SECRIN assumes the trustworthy VMMs.

Different designs [3], [23], [24], [31] are enforced to protect cryptographic keys in computer systems. These approaches work compatibly with SECRIN to protect the keys in hosts.

TABLE I
Speed of Native AES Encryption (Block/s)

Blocks per Call	1	2	4	8
in User Space	1302305.8	1307762.7	1339678.9	1334046.0
in Kernel Space	1102686.5	1118331.1	1121683.5	1132547.2
in virtio-ct	31717.6	37242.2	38862.8	38917.3

TABLE II
File Read/Write Throughput (kByte/s)

	File Size (MB)	2	4	8	16	32
Read	w/o encryption	12216	10272	13733	13394	11276
	AES-generic	12144	10160	11719	11718	12194
	virtio-ct	479	499	471	474	487
Write	w/o encryption	10635	10050	11567	12566	13241
	AES-generic	9910	9579	10797	10671	10404
	virtio-ct	408	456	481	467	470

VIII. Conclusion

We propose SECRIN, a secure cryptography infrastructure in the cloud. It utilizes virtualization to enhance the security of cryptographic systems, and extends the capacities of virtualization platforms by introducing virtual cryptographic devices as common resources of cloud services. The virtual devices are *a*) protected in the memory space of VMMs, against the attacks happening in VMs, and *b*) managed in the virtualization management system as well as other resources, but with special controls by the tenants.

References

[1] National Vulnerability Database, "CVE-2014-0160," http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160.

[2] ——, "CVE-2014-0069," http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0069.

[3] K. Harrison and S. Xu, "Protecting cryptographic keys from memory disclosure attacks," in *37th IEEE/IFIP DSN*, 2007, pp. 137–143.

[4] G. Guninski, "Linux kernel 2.6 fun, Windoze is a joke," 2005, http://www.guninski.com/where_do_you_want_billg_to_go_today_3.html.

[5] J. Halderman, S. Schoen *et al.*, "Lest we remember: Cold boot attacks on encryption keys," *17th USENIX Security*, pp. 45–60, 2008.

[6] P. Stewin and I. Bystrov, "Understanding DMA malware," in *9th DIMVA*, 2013, pp. 21–41.

[7] B. Sotomayor, R. Montero *et al.*, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[8] "oVirt," http://www.ovirt.org/.
[9] R. Russel, "virtio: Towards a de-facto standard for virtual I/O devices," *Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.
[10] "libvirt - the virtualization API," https://www.libvirt.org/.
[11] National Vulnerability Database, "CVE-2014-4653," http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-4653.
[12] ——, "CVE-2008-0923," https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-0923.
[13] R. Wojtczuk, J. Rutkowska, and A. Tereshkin, "Xen 0wning trilogy," in *Black Hat*, 2009.
[14] A. Seshadri, M. Luk *et al.*, "SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes," in *21st ACM SOSP*, 2007, pp. 335–350.
[15] U. Steinberg and B. Kauer, "NOVA: A microhypervisor-based secure virtualization architecture," in *5th EuroSys*, 2010, pp. 209–222.
[16] Amazon Web Services, "Protecting data using server-side encryption with customer-provided encryption keys (SSE-C)," https://docs.aws.amazon.com/AmazonS3/latest/dev/ServerSideEncryptionCustomerKeys.html.
[17] ——, "S3 service level agreement," https://aws.amazon.com/cn/s3/sla/.
[18] Google, "Cloud storage: Concept and techniques," https://cloud.google.com/storage/docs/concepts-techniques.
[19] Microsoft, "Protecting data in Microsoft Azure," 2014.
[20] J. Bonneau and I. Mironov, "Cache-collision timing attacks against AES," in *8th CHES*, 2006, pp. 201–215.
[21] C. Arnaud and P. Fouque, "Timing attack against protected RSA-CRT implementation used in PolarSSL," in *CT-RSA*, 2013, pp. 18–33.
[22] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.
[23] L. Guan, J. Lin *et al.*, "Protecting private keys against memory disclosure attacks using hardware transactional memory," in *36th IEEE S&P*, 2015, pp. 3–19.
[24] ——, "Copker: Computing with private keys without RAM," in *21st (NDSS)*, 2014.
[25] Princeton University, "AESKeyFinder and RSAKeyFinder," https://citp.princeton.edu/research/memory/code/.
[26] "SPICE," http://www.spice-space.org/.
[27] S. Berger, R. Caceres *et al.*, "vTPM: Virtualizing the trusted platform module," in *15th USENIX Security*, 2006, pp. 305–320.
[28] S. Bleikertz, S. Bugiel *et al.*, "Client-controlled cryptography as a service in the cloud," in *11th ACNS*, 2013, pp. 19–36.
[29] B. Danev and R. Masti, "Enabling secure VM-vTPM migration in private clouds," in *27th ACSAC*, 2011, pp. 187–196.
[30] T. Müller, B. Taubmann, and F. Freiling, "TreVisor: OS-independent software-based full disk encryption secure against main memory attacks," in *10th ACNS*, 2012, pp. 66–83.
[31] N. Mavrogiannopoulos, M. Trmac, and B. Preneel, "A Linux kernel cryptographic framework: Decoupling cryptographic keys from applications," in *27th ACM SAC*, 2012, pp. 1435–1442.