Secure Out-of-band Remote Management Using Encrypted Virtual Serial Consoles in IaaS Clouds

Kenichi Kourai
Department of Creative Informatics
Kyushu Institute of Technology
Fukuoka, Japan
kourai@ci.kyutech.ac.jp

Tatsuya Kajiwara
Department of Creative Informatics
Kyushu Institute of Technology
Fukuoka, Japan
kajita@ksl.ci.kyutech.ac.jp

***Abstract*—In Infrastructure-as-a-Service (IaaS) clouds, users manage the systems in virtual machines (VMs) through remote management systems such as Secure Shell (SSH). IaaS often provides out-of-band remote management using virtual serial consoles (VSCs). Using VSCs, even on failures inside their VMs, users can directly access their systems through a virtual serial device running in the *management VM*. However, the management VM is not always trustworthy in IaaS. Attackers in the management VM can easily eavesdrop on inputs and outputs in remote management. To prevent such information leakage, this paper proposes *SCCrypt*, which provides encrypted VSCs to the management VM. Encrypted VSCs are securely achieved by the trusted virtual machine monitor (VMM). They decrypt console inputs encrypted by SSH clients, while they encrypt console outputs, which are decrypted by SSH clients. For this purpose, the VMM correctly identifies the inputs and outputs by tracking device state without the cooperation of the management VM or user VMs. At reconnection to encrypted VSCs, the VMM re-encrypts pending console outputs by reversely applying encryption process of a stream cipher. We have implemented SCCrypt in Xen and the OpenSSH client and confirmed that the overhead was small enough.**

***Keywords*-Virtual machines, remote management, information leakage, IaaS clouds, insider attacks**

I. Introduction

Infrastructure as a Service (IaaS) provides users with virtual machines (VMs) hosted in data centers. Its users can set up their systems in the provided VMs called *user VMs* and use them as necessary. They usually manage their systems through remote management systems such as Secure Shell (SSH). To allow users to access their systems even on failures inside their VMs, IaaS often provides *out-of-band remote management* using virtual serial consoles (VSCs). Unlike traditional remote management, an SSH server is run in a special VM called the *management VM*, not in a user VM, and directly interact with a virtual serial device for a user VM. Even if the network of a user VM is disconnected due to configuration errors, a user can continue to manage the VM.

However, this out-of-band remote management increases security risks because the management VM is not always trustworthy in IaaS [1]–[4]. The management VM may be compromised by outside attackers if it is not well maintained. If some of the administrators are malicious, they may mount insider attacks [5]. Such attackers can easily eavesdrop on inputs and outputs in remote management by replacing an SSH server with a malicious one. For example, they can extract passwords from console inputs sent from an SSH client and steal sensitive or private information from console outputs. In addition, they may execute arbitrary commands inside user VMs by sending console inputs.

To solve this security problem, we propose *SCCrypt*, which protects sensitive information in out-of-band remote management against attackers in the management VM. SCCrypt provides *encrypted VSCs* to the management VM. Encrypted VSCs are securely achieved using the virtual machine monitor (VMM). The VMM is a software layer underlying VMs and can be trusted by several techniques [6]–[9]. In an encrypted VSC, console inputs encrypted by an SSH client are securely decrypted by the trusted VMM. In contrast, console outputs are securely encrypted by the VMM and they are decrypted by an SSH client. At that time, the VMM correctly identifies the inputs and outputs by tracking device state without cooperation of the management VM or user VMs. When an SSH client reconnects to an encrypted VSC, the VMM decrypts pending console outputs in a virtual serial device by reversely applying encryption process of a stream cipher and re-encrypts them.

We have implemented SCCrypt for both fully virtualized (HVM) and para-virtualized (PV) guest operating systems in Xen [10]. For HVM guests, the VMM traps I/O instructions to virtual serial devices and extracts input and output data using I/O port addresses and tracked device state. For PV guests, the VMM identifies buffers called console rings allocated in user VMs and mediates the accesses by virtual serial devices. Our experimental results show (1) that attackers in the management VM could not steal console inputs and outputs and (2) that the overhead of SCCrypt was small enough.

The organization of this paper is as follows. Section II describes issues in out-of-band remote management via the management VM. Section III proposes SCCrypt for protect-

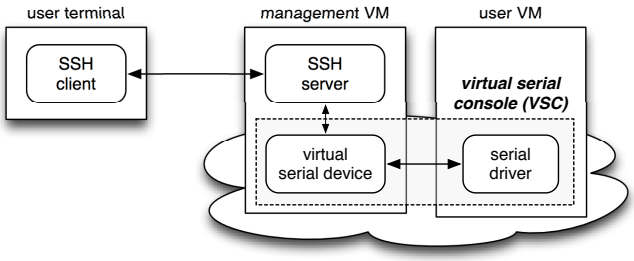

Figure 1. Out-of-band remote management using a virtual serial console.

ing sensitive information in out-of-band remote management and Section IV explains the implementation details in Xen. Section V shows our experimental results. Section VI describes related work and Section VII concludes this paper.

II. Motivation

A. Out-of-band Remote Management in Clouds

To manage user VMs in IaaS clouds, a user usually connects an SSH client at his host to an SSH server running in a user VM. This is called *in-band* remote management because a user accesses a VM using functionalities provided inside the VM. However, this in-band remote management is not powerful enough to manage a user VM at any times. If a user has just failed the configurations of the network or firewall inside a VM, he cannot manage the VM at all. At that time, he would have to abandon that VM and recreate a new VM from scratch. As another example, when an SSH server in a user VM is not running normally, an SSH client cannot access the VM. For example, an SSH server is not started until the operating system in a VM has been booted normally. It may crash by bugs.

To enable users to manage their VMs in such cases, IaaS often provides *out-of-band* remote management. As illustrated in Fig. 1, an SSH server is run in a privileged VM called the *management VM*. The management VM is often provided in type-I VMMs such as Xen and Hyper-V and has privileges for accessing all user VMs. The SSH server accesses a *virtual serial console (VSC)* provided for each user VM. A VSC consists of a virtual serial device in the management VM and a serial driver in a user VM. Out-of-band remote management does not rely on the network or an SSH server in a user VM. A user can access his VM as if he locally logged into the VM even on network failures in the VM. For example, even if a user fails network configuration in a user VM, he could fix the problem by modifying the configuration using a VSC.

This out-of-band remote management relies on the management VM, but the management VM is not always trustworthy in clouds [1]–[4]. Since user VMs can be migrated between data centers, it is not guaranteed that they are run in data centers where all the administrators are trusted. If the management VM is managed by *lazy* administrators, it may have vulnerabilities in software or configurations. In this case, vulnerable management VMs may be penetrated by outside attackers. Worse, if administrators themselves are *malicious*, they can act as inside attackers [5].

If such attackers abuse the privileges of the management VM, they can eavesdrop on or tamper with inputs and outputs in remote management. Even if the network is encrypted between a client host and the management VM, the data processed by an SSH server in the management VM is not encrypted. For console inputs, attackers can modify the SSH server and the virtual serial device and then easily obtain input data. For example, attackers can extract passwords from the inputs to a login prompt. In addition, they can send input data to a user VM and make the VM execute arbitrary commands. For console outputs, attackers can steal sensitive information output to the console. For example, when a user edits configuration files, attackers can obtain displayed passwords in them.

B. Threat Model and Assumptions

We assume that IaaS providers themselves are trusted. This assumption is widely accepted [1]–[4]. To guarantee this, it is natural that a small number of trusted administrators are responsible for the maintenance of the infrastructure in IaaS. The other average administrators may be lazy or malicious. We do not consider physical attacks because server rooms should be strictly protected in data centers.

We assume that the management VM can be malused by outside attackers or untrusted administrators in IaaS. Such attackers could take the root privilege in the management VM and even modify the operating system kernel. In this paper, we focus on the protection of sensitive information sent between an SSH client and a user VM in out-of-band remote management. We do not consider the integrity and availability of out-of-band remote management. Also, the confidentiality of user VMs is out of scope. It can be protected by several techniques [2], [4].

III. SCCrypt

In this paper, we propose *SCCrypt*, which enables secure out-of-band remote management to prevent information leakage to the management VM. For this purpose, SCCrypt provides *encrypted VSCs* to the management VM. An encrypted VSC receives encrypted console inputs from the management VM, decrypts them, and sends them to a user VM. Conversely, it receives unencrypted console outputs from a user VM, encrypts them, and sends them to the management VM. Console inputs and outputs are encrypted and decrypted by an SSH client, respectively. Even if attackers in the management VM eavesdrop on encrypted VSCs, information leakage is prevented because all the console inputs and outputs are encrypted.

A research question is where such encryption and decryption can be done securely. It is straightforward to

encrypt and decrypt data in the management VM, but the management VM is not trusted in clouds. If an SSH server or a virtual serial device decrypts and encrypts console inputs and outputs, respectively, attackers can tamper with such a component in the management VM and easily steal sensitive information. Another possible location is a serial driver in a user VM. However, it is not desirable to modify the existing device driver because users cannot use favorite operating systems. This is critical in IaaS clouds.

Therefore, SCCrypt encrypts and decrypts data for an encrypted VSC in the VMM, as illustrated in Fig. 2. The VMM is a software layer underlying VMs and manages the interaction between VMs. The integrity of the VMM can be guaranteed by several techniques. Remote attestation with TPM [6] enables the trusted authority to check the integrity at boot time. HyperGuard [7], HyperCheck [8], and HyperSentry [9] can securely monitor the VMM at runtime using System Management Mode (SMM) in the commodity x86 processor families. We assume that such infrastructure for trusting the VMM is securely maintained by trusted administrators in IaaS.

Using the trusted VMM, SCCrypt encrypts and decrypts console inputs and outputs as follows. For console inputs, an SSH client encrypts input data and sends it to an SSH server in the management VM. Note that this encryption is different from the usual encryption done by SSH. After the SSH server writes the encrypted data to a virtual serial device, the device transmits the data to a user VM via the VMM. At that time, the VMM decrypts the data. For console outputs, the VMM intercepts output data when the serial driver in a user VM writes it to a virtual serial device. Then it encrypts the data and, instead of the serial driver, writes the encrypted data to the virtual serial device. The data is sent to an SSH client via an SSH server and is decrypted by the client.

A challenge is how the VMM identifies console inputs to be decrypted and outputs to be encrypted. The traditional VMM does not recognize the details of a virtual serial device in the management VM and a serial driver in a user VM. However, the VMM cannot rely on information from an untrusted virtual serial device. In addition, it cannot obtain extra information from an unmodified serial driver. In SCCrypt, therefore, the VMM identifies necessary data without the cooperation of the device and the driver. It tracks the state of a virtual serial device and a serial driver from only the interactions between the management VM and a user VM. Based on the tracked state and the knowledge of the standard of serial devices, the VMM can extract only console inputs and outputs. Further details are explained in the next section.

In SCCrypt, an SSH client securely shares a session key with the VMM whenever it connects to an encrypted VSC. First, it obtains the public key of the VMM from the trusted key server outside a cloud. We assume that the public key is

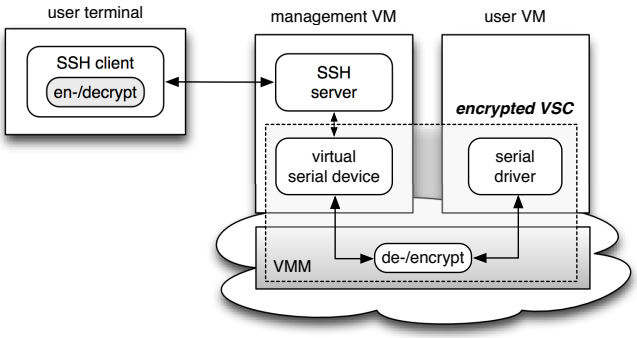

Figure 2. The architecture of SCCrypt.

registered by trusted cloud administrators in advance. Then the SSH client generates a session key, encrypts it with the public key, and sends it to the VMM. The VMM decrypts the encrypted session key with its private key and uses it for decrypting console inputs and encrypting console outputs. It should be noted that the SSH client has to send the session key via the management VM due to the lack of the communication ability in the VMM. Attackers can obtain the encrypted key but cannot decrypt it because the private key is stored in the VMM.

At reconnection, it is challenging to process pending console outputs in a virtual serial device. While an SSH client is not connected to an encrypted VSC, console outputs from a user VM are stored in the buffer of a virtual serial device. When an SSH client is reconnected to the encrypted VSC, output data in the buffer is sent to the client. However, the SSH client cannot decrypt it because SCCrypt has encrypted the output data with an old session key. Therefore, the VMM decrypts output data in the buffer with an old session key and re-encrypts it with a new session key before the data is sent to an SSH client. Since SCCrypt uses a stream cipher, output data encrypted by the VMM can be usually decrypted only by an SSH client. To enable the VMM to decrypt such data, the VMM reversely applies the encryption process to encrypted data.

IV. Implementation

We have implemented SCCrypt in Xen 4.1.3 [10] and the OpenSSH 6.0p1 client. In Xen, the management VM and a user VM are called Dom0 and DomU, respectively. A virtual serial device is a part of QEMU running in Dom0. As guest operating systems, SCCrypt supports fully virtualized and para-virtualized Linux operating systems. The former is called *HVM guests*, which can be run in VMs without any modification. The latter is called *PV guests*, which are modified for running in VMs. In the current implementation, SCCrypt targets the x86-64 architecture.

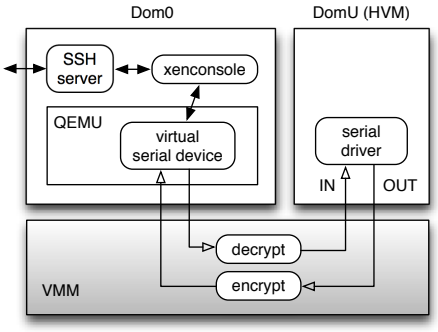

Figure 3. The encrypted delivery of console inputs and outputs for HVM guests.

A. Output Delivery from HVM Guests

In the out-of-band remote management using a traditional VSC, the delivery of console outputs to a HVM guest are performed as follows. When the operating system in DomU outputs data to a serial console, a serial driver writes the data to a serial port using the OUT instruction. At that time, the VMM traps the instruction using the function of Intel VT-x, as illustrated in Fig. 3. It emulates the instruction and sends the output data to QEMU in Dom0. The virtual serial device in QEMU receives the data and stores the data in a FIFO buffer. Then it sends the data to xenconsole via a pseudo-terminal (PTY). Xenconsole is executed by an SSH server to connect the server to the virtual serial device. Then it sends the data to the SSH server and finally the server sends it to an SSH client.

SCCrypt encrypts the output data when the VMM emulates the OUT instruction. The OUT instruction specifies an I/O port address and written data. The VMM encrypts the data with RC4 when the address is `0x3F8`, to which the transmitter holding buffer of COM1 is mapped. Since a virtual serial device has multiple contexts, the data is encrypted only if the following three conditions are satisfied. First, the FIFO buffers must be enabled. They are enabled at the boot time of an HVM guest and any data is not transmitted until that. Second, the virtual serial device must not be in the divisor latch access mode. The mode is used to set the baud rate of the transmission. Third, the device must not be in the loopback mode. The mode is used to test the device at boot time and written data are not transmitted. However, the traditional VMM does not recognize the context of the device in QEMU because QEMU runs in untrusted Dom0.

To recognize the context in the VMM, the VMM watches all the writes to a virtual serial device. When the FIFO enabling bit is set to the FIFO control register, which is mapped to I/O port address `0x3FA`, the VMM can know that the FIFO buffers are enabled and starts to encrypt written data. Similarly, while the divisor latch access bit is set to the line control register, which is mapped to `0x3FB`, the VMM considers that the device is in the divisor latch access mode and ignores written data. While the loopback mode bit is set to the modem control register, which is mapped to `0x3FC`, the VMM considers the device in the loopback mode and does not encrypt written data.

B. Input Delivery to HVM Guests

In the traditional systems, console inputs are delivered to an HVM guest as follows. When an SSH server in Dom0 receives input data sent from an SSH client, it sends the data to QEMU via xenconsole, as in Fig. 3. The virtual serial device in QEMU stores the data in a FIFO buffer. When the serial driver in DomU executes the IN instruction to read input data, the VMM traps and emulates the instruction. It communicates with the virtual serial device in Dom0 and receives data in the FIFO buffer. Finally it stores the received data in the EAX register as a return value.

SCCrypt decrypts input data when the VMM stores it in the EAX register. Since the IN instruction specifies an I/O port address like the OUT instruction, the VMM decrypts the data received from the virtual serial device with RC4 when the address is `0x3F8`, to which the receiver buffer is mapped. As in the output delivery, the input data is decrypted only if the FIFO buffer is enabled and if the virtual serial device is in neither the divisor latch access mode nor the loopback mode.

C. Output Delivery from PV Guests

The delivery of console outputs from a PV guest is largely different from that from an HVM guest in the interaction between Dom0 and DomU. In Xen, the split-driver model is used for PV guests. As shown in Fig. 4, a virtual serial device is implemented as the console backend driver in QEMU, whereas a serial driver is implemented as the console frontend driver in DomU. When the operating system in DomU outputs data to a serial console, the frontend driver writes the data to a ring buffer called a *console ring* for outputs. Traditionally, the backend driver in Dom0 directly reads the data in the console ring by sharing it with DomU. This means that the VMM cannot recognize console outputs at all and cannot encrypt them.

For SCCrypt, we have modified the backend driver so that the driver issues a hypercall for reading data in the console ring to the VMM. A hypercall is similar to a system call to the operating system. When the hypercall is issued, the VMM reads data in the console ring, encrypts it with RC4, and returns it to the backend driver. The backend driver cannot directly access the console ring any longer. The VMM can prohibit the backend driver from directly sharing it or encrypt DomU's memory including the console ring using SRE [2] or VMCrypt [4]. Note that we do not need the modification of the frontend driver in DomU.

Since the traditional VMM is not aware of a console ring in DomU, our VMM identifies a console ring from information stored in the memory page called start_info.

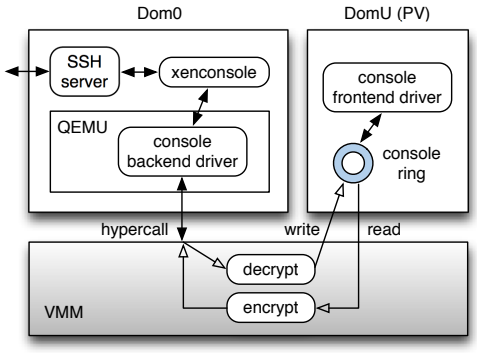

Figure 4. The encrypted delivery of console inputs and outputs for PV guests.

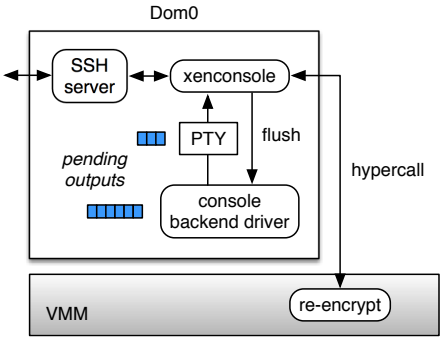

Figure 5. The re-encryption of pending console outputs.

This page is set up by Dom0 for each DomU when Dom0 boots DomU. The VMM first obtains the virtual address of the start_info page from the RSI register of a virtual CPU for DomU. The address is set by Dom0 at the boot time of DomU. Next, the VMM translates the virtual address into a physical page frame number by examining the page list of DomU. Then it maps the start_info page and obtains information on a console ring.

D. Input Delivery to PV Guests

As in the delivery of output data, we have modified the console backend driver so that it writes input data to a console ring for inputs in DomU using a hypercall added for SCCrypt. Traditionally, when the backend driver receives input data from an SSH server via xenconsole, it directly writes the data to the console ring in DomU. Then the frontend driver in DomU reads the data from the console ring. In SCCrypt, the VMM can decrypt input data passed from the backend driver with RC4 and write the decrypted data to the console ring, as illustrated in Fig. 4.

E. Re-encryption at Reconnection

When an SSH client reconnects to an encrypted VSC, the VMM re-encrypts pending console outputs. Such data is stored in a virtual serial device for HVM guests or the console backend driver for PV guests while an SSH client is not connected. Some data is also left in PTY if it has not been read by xenconsole. To re-encrypt such data, xenconsole first pauses a user VM to prevent new console outputs from being sent. For PV guests, it sends the flush command and makes the backend driver write output data in the buffer to PTY, as shown in Fig. 5. The buffer can store pending data up to 1 MB by default. For HVM guests, output data in the transmit shift register and the FIFO buffer is automatically resent unless a retry count exceeds the maximum. After that, xenconsole reads all the pending output data from PTY.

Since only the VMM can re-encrypt output data, xenconsole issues a hypercall with the received pending data. The VMM first restores unencrypted data from data encrypted with an old session key and then encrypts obtained data with a new session key. Since it cannot simply decrypt the data that is encrypted with RC4, it reverses the encryption process, inspecting the internal state of RC4 associated with the old session key. The internal state consists of an array S with 256 elements and two indexes i and j. The encryption process of RC4 increases i by one, adds j to $S[i]$, and swaps $S[i]$ and $S[j]$. Then it calculates XOR between data and $S[k]$, where k is $S[i] + S[j]$. Note that the arithmetic operations are done with modulo 256. Its reverse process is to calculate XOR between encrypted data and $S[k]$. For the successive restoration, it also swaps $S[i]$ and $S[j]$, subtracts j from $S[i]$, and decreases i by one. Such a reverse operation is applicable to other ciphers such as AES-CTR. This process is done from the newest data in a reverse order. Finally, xenconsole resumes a user VM and sends re-encrypted output data to an SSH client.

F. Connection to encrypted VSC

In SCCrypt, all of the input data are encrypted by an SSH client. This causes one issue when a user connects to an encrypted VSC. For a new connection, a user needs to log in Dom0 using an SSH client and to execute the xenconsole command. Since the SSH client cannot distinguish that command input to a shell from inputs to an encrypted VSC, the command input is also encrypted. Consequently, a user cannot execute that command correctly. To prevent the input of the xenconsole command from being encrypted, SCCrypt uses the remote command execution in SSH. When an SSH client is executed with a remote command, the command is sent to an SSH server in a manner different from normal inputs. Therefore the SSH client can keep that command unencrypted.

V. Experiments

We conducted experiments to confirm the prevention of information leakage by SCCrypt and examine its overhead. For a server machine, we used a PC with one Intel Core i7 870 2.93 GHz processor, 4 GB of memory, and a Gigabit

Ethernet NIC. We ran a modified version of Xen 4.1.3 for the x86-64 architecture. For comparison, we also used unmodified vanilla Xen. We assigned eight virtual CPUs and 1 GB of memory to DomU and eight virtual CPUs and 3 GB of memory to Dom0. In DomU, we ran Linux 3.2.0 as HVM and PV guests. In Dom0, we ran the OpenSSH 5.9p1 server on Linux 3.2.0. For a client machine, we used a PC with one Intel Xeon E5630 2.53 GHz processor, 6 GB of memory, and a Gigabit Ethernet NIC. We ran a modified version of the OpenSSH 6.0p1 client on Linux 3.2.0.

A. Attempts at Eavesdropping

To confirm that SCCrypt prevents Dom0 from eavesdropping on inputs to a VSC, we embedded a custom keylogger into a virtual serial device in Dom0. This keylogger recorded console inputs sent from an SSH client. Using an SSH client, we logged into DomU by typing a user name and a password. Without SCCrypt, the plain-text password was recorded. When SCCrypt was enabled, the password was encrypted by the SSH client and the keylogger recorded encrypted one. For console outputs, we embedded custom screen capture into the virtual serial device. This screen capture recorded console outputs sent by DomU. When we used vanilla Xen, the plain-text outputs were recorded. With SCCrypt enabled, encrypted ones were recorded. Nevertheless, we could log into DomU and manage DomU correctly.

B. Response Time of Console Inputs

To examine the overhead of SCCrypt, we measured the response time of console inputs. The response time was the time from when an SSH client sent a console input until it received a console output caused by its remote echo. We measured the response time 100 times and obtained the average and standard deviation. Then we compared the result in SCCrypt with that in vanilla Xen for HVM and PV guests.

Fig. 6 shows the results. For a HVM guest, the response time in SCCrypt was 2.9% shorter than that in vanilla Xen. The reason is unclear because we have just added extra code for SCCrypt to the VMM. However, this means that the overhead of encryption and decryption by RC4 was negligible. For a PV guest, on the other hand, the response time in SCCrypt was longer than that in vanilla Xen. The overhead of SCCrypt was 11%. This is because the virtual serial device in Dom0 issued hypercalls to the VMM for accessing I/O rings. Note that the response time for a PV guest is still shorter than that for an HVM guest in SCCrypt.

C. Throughput of Console Outputs

To examine the performance of SCCrypt under a heavy workload, we measured the throughput of console outputs. In this experiment, we logged into DomU using a VSC and wrote a text file of 10 million characters to a serial console by the cat command. We compared the throughput in SCCrypt with that in vanilla Xen for HVM and PV guests.

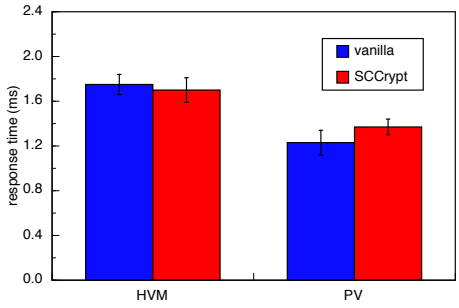

Figure 6. The response time of console inputs.

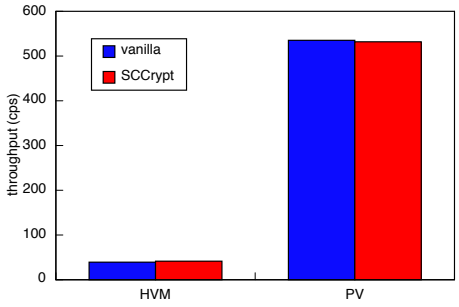

Figure 7. The throughput of console outputs.

Fig. 7 shows the results in characters per second (cps). For an HVM guest, the throughput in SCCrypt was 5.6% higher than that in vanilla Xen. The reason is probably the same as that why the response time was shorter. For a PV guest, on the other hand, the throughput in SCCrypt was almost the same as that in vanilla Xen. Unlike the experiment for the response time, the virtual serial device in Dom0 could obtain many outputs from a console ring at once. Therefore the number of issued hypercalls was reduced. It should be noted that the throughput for a PV guest was much higher than that for an HVM guest.

D. CPU Overhead for Console Inputs

We measured the CPU utilization when we sent inputs to an encrypted VSC periodically. For periodic console inputs, we used the keyboard auto-repeat in the X Window System at a client machine. We configured the repeat rate to 2, 10.9, and 30 cps. The system default was 10.9 cps. We measured the CPU utilization of Dom0 and DomU for 10 seconds without inputs and successive 60 seconds with inputs. When Dom0 issued hypercalls, the CPU time consumed by the VMM was accounted for Dom0. When an HVM guest executed I/O instructions, which were trapped by the VMM, the CPU time was accounted for DomU. Then we compared the result in SCCrypt with that in vanilla Xen for HVM and PV guests.

Fig. 8 shows the changes of the CPU utilization of Dom0 and DomU for HVM and PV guests in SCCrypt when the repeat rate was 10.9 cps. When we started the keyboard

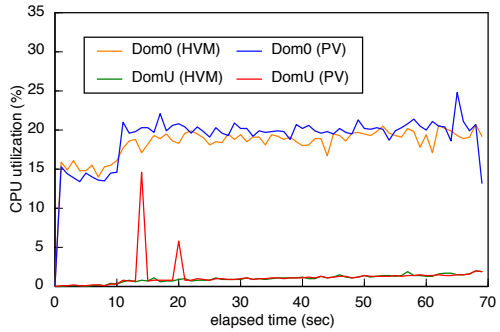

Figure 8. The change of the CPU utilization for console inputs in SCCrypt.

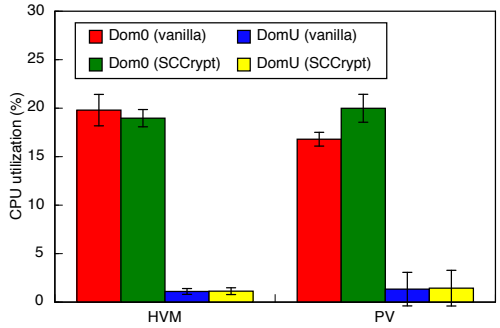

Figure 9. The CPU utilization for console inputs.

auto-repeat at time 10 seconds, the CPU utilization of Dom0 suddenly increased, while that of DomU almost did not increase. The CPU utilization of DomU gradually increased over time. It was almost the same between HVM and PV guests except for two spikes in DomU for a PV guest.

Fig. 9 shows the average CPU utilization and the standard deviation while we sent inputs. This figure compares the CPU utilization of Dom0 and DomU for HVM and PV guests between vanilla Xen and SCCrypt. The repeat rate was 10.9 cps. For an HVM guest, the CPU utilization of Dom0 in SCCrypt was 4.2% lower than that in vanilla Xen. For a PV guest, in contrast, that in SCCrypt was 19% higher than that in vanilla Xen. These match the results of the response time. When we compare the CPU utilization of Dom0 between PV and HVM guests that for a PV guest was lower than that for an HVM guest in vanilla Xen. In SCCrypt, however, that for a PV guest became 5.4% higher than that for an HVM guest. Note that the CPU utilization was almost not affected by the difference of the repeat rate.

E. CPU Overhead for Console Outputs

We measured the CPU utilization when we executed a benchmark that periodically wrote characters to a serial console in DomU. We configured the output rate to 0, 10, 100, or 1000 cps. We conducted this experiment as in the previous section. Fig. 10 is the CPU utilization of Dom0 and DomU for a PV guest in vanilla Xen and SCCrypt. We

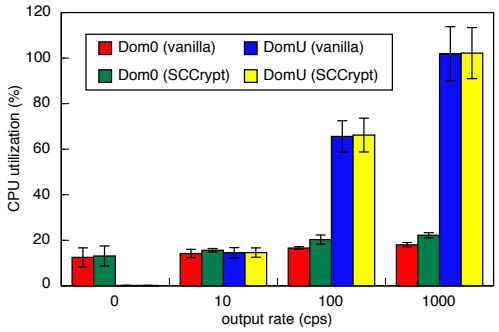

Figure 10. The CPU utilization for console outputs for a PV guest.

omitted the results for an HVM guest because they were almost the same as those for a PV guest.

As the output rate became large, the CPU utilization of DomU increased. This is due to the CPU consumption by the benchmark itself. The CPU utilization of Dom0 increased as well, but the increase was 69% even at 1000 cps in SCCrypt. Only for a PV guest, the difference of the CPU utilization of Dom0 between vanilla Xen and SCCrypt increased. At 1000 cps, the CPU utilization in SCCrypt was 23% higher than that in vanilla Xen.

VI. Related Work

FBCrypt [11] prevents information leakage in out-of-band remote management for VNC. It encrypts keyboard and mouse inputs and video outputs using the VMM and a VNC client. FBCrypt is similar to SCCrypt, but there are two major differences. One is that FBCrypt strongly depends on VNC. SCCrypt can use other remote management systems using VSCs, e.g., web-based Ajaxterm [12], because it decouples encrypted VSCs from SSH. The other difference is that FBCrypt does not need to process pending outputs in a virtual video card at reconnection. A VNC client requests full-screen data again at that time. In SCCrypt, the correct processing of pending console outputs is crucial. In addition, out-of-band remote management using VSCs is often more desirable because VNC is heavyweight.

So far, out-of-band remote management without the management VM has been proposed. Xoar [13] runs a virtual serial device and probably an SSH server in a dedicated VM called Console VM. Since Console VM provides a single service, it is more difficult for outside attackers to intrude into it. However, if the SSH server is compromised, attackers can eavesdrop on sensitive information on out-of-band remote management. Untrusted administrators in IaaS can normally log in Console VM and easily obtain such information. Similarly, stub domains [14] are VMs used for running QEMU in Xen. Out-of-band remote management using a VNC server in QEMU is possible. Since QEMU is run as the only application on top of Mini-OS, it is difficult to run an SSH server in such a VM and perform remote

management using VSCs.

VMware vSphere Hypervisor runs a VNC server and virtual devices in the VMM. The VNC server can directly access virtual devices for user VMs. In vSphere, therefore, information leakage via the management VM does not occur. However, attackers can steal sensitive information in remote management if they can compromise the VNC server in the VMM. They can take over even the control of the VMM itself. SCCrypt preserves the confidentiality in remote management by the VMM even in the case that an SSH server in the management VM is compromised.

Several researchers have proposed systems that encrypt data for user VMs in underlying layers. The secure runtime environment (SRE) [2], [15] and VMCrypt [4] prevent information leakage from the memory of user VMs to the management VM. When the management VM maps memory pages of a user VM, the VMM encrypts their contents. This architecture is complementary to SCCrypt in that it prevents the management VM from stealing information inside user VMs.

BitVisor [16] can prevent information leakage from storage and network of a user VM. It is similar to SCCrypt in that the VMM transparently encrypts I/O of a user VM without the help of the management VM. However, BitVisor does not provide a means of remote management.

CloudVisor [3] runs the security monitor underneath the VMM and encrypts the memory and storage of the user VMs in the security monitor. Since it distrusts not only the management VM but also the VMM, it can prevent information leakage even from the VMM. However, the security monitor does not encrypt inputs and outputs in remote management.

VII. Conclusion

In this paper, we proposed SCCrypt for enabling secure out-of-band remote management in IaaS clouds. To prevent information leakage via the management VM, SCCrypt provides encrypted VSCs to the management VM. Console inputs and outputs are securely decrypted and encrypted in the trusted VMM, respectively. The VMM correctly identifies the inputs and outputs by tracking device state without the cooperation of the management VM or user VMs. To support the change of a session key at the reconnection to an encrypted VSC, the VMM re-encrypts pending console outputs by reversely applying encryption process. We have implemented SCCrypt for fully virtualized and paravirtualized guest operating systems in Xen. We confirmed that the security in out-of-band remote management was enhanced and that the overhead of SCCrypt was small enough.

One of our future work is to apply SCCrypt to other remote management systems using VSCs, e.g., Ajaxterm [12]. This would be easy because SCCrypt provides encrypted VSCs independently from SSH. Another direction is to support stronger stream ciphers such as AES-CTR, as supported in FBCrypt [11].

Acknowledgment

This work was supported in part by JSPS KAKENHI Grant Number 25330086.

References

[1] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards Trusted Cloud Computing," in *Proc. Workshop Hot Topics in Cloud Computing*, 2009.

[2] C. Li, A. Raghunathan, and N. K. Jha, "Secure Virtual Machine Execution under an Untrusted Management OS," in *Proc. Intl. Conf. Cloud Computing*, 2010, pp. 172–179.

[3] F. Zhang, J. Chen, H. Chen, and B. Zang, "CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization," in *Proc. Symp. Operating Systems Principles*, 2011, pp. 203–216.

[4] H. Tadokoro, K. Kourai, and S. Chiba, "Preventing Information Leakage from Virtual Machines' Memory in IaaS Clouds," *IPSJ Online Transactions*, vol. 5, pp. 156–166, 2012.

[5] TechSpot News, "Google Fired Employees for Breaching User Privacy," http://www.techspot.com/news/40280-google-fired-employees-for-breaching-user-privacy.html, 2010.

[6] Trusted Computing Group, "TPM Main Specification," http://www.trustedcomputinggroup.org/, 2011.

[7] J. Rutkowska, R. Wojtczuk, and A. Tereshkin, "Xen 0wning Trilogy," Black Hat USA, 2008.

[8] J. Wang, A. Stavrou, and A. Ghosh, "HyperCheck: A Hardware-Assisted Integrity Monitor," in *Proc. Int. Symp. Recent Advances in Intrusion Detection*, 2010, pp. 158–177.

[9] A. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. Skalsky, "HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity," in *Proc. Conf. Computer and Communications Security*, 2010, pp. 38–49.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Proc. Symp. Operating Systems Principles*, 2003, pp. 164–177.

[11] T. Egawa, N. Nishimura, and K. Kourai, "Dependable and Secure Remote Management in IaaS Clouds," in *Proc. Intl. Conf. Cloud Computing Technology and Science*, 2012, pp. 411–418.

[12] A. Lesuisse, "Ajaxterm," https://github.com/antonylesuisse/qweb.

[13] P. Colp, M. Nanavati, J. Zhu, W. Aiello, G. Coker, T. Deegan, P. Loscocco, and A. Warfield, "Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor," in *Proc. Symp. Operating Systems Principles*, 2011, pp. 189–202.

[14] S. Thibault, "Stub Domains," in *Xen Summit Boston 2008*, 2008.

[15] C. Li, A. Raghunathan, and N. K. Jha, "A Trusted Virtual Machine in an Untrusted Management Environment," *IEEE Trans. Services Computing*, vol. 5, no. 4, pp. 472–483, 2012.

[16] T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai, K. Kono, S. Chiba, Y. Shinjo, and K. Kato, "BitVisor: A Thin Hypervisor for Enforcing I/O Device Security," in *Proc. Intl. Conf. Virtual Execution Environments*, 2009, pp. 121–130.