IPSJ Transactions on Advanced Computing Systems Vol.6 No.3 108–117 (Sep. 2013)

Regular Paper

Security Enhancement of Out-of-band Remote Management in IaaS Clouds

Tomohisa Egawa[1,a)] Naoki Nishimura[1,b)] Kenichi Kourai[1,2,c)]

Received: December 21, 2012, Accepted: March 3, 2013

Abstract: In Infrastructure-as-a-Service (IaaS) clouds, the users manage the systems in the provided virtual machines (VMs) called *user VMs* through remote management software such as Virtual Network Computing (VNC). For dependability, they often perform out-of-band remote management via the *management VM*. Even in the case of system failures inside their VMs, the users could directly access their systems. However, the management VM is not always trustworthy in IaaS. Once outside or inside attackers intrude into the management VM, they could easily eavesdrop on all the inputs and outputs in remote management. To solve this security issue, this paper proposes *FBCrypt* for preventing information leakage via the management VM in out-of-band remote management. FBCrypt encrypts the inputs and outputs between a VNC client and a user VM using the *virtual machine monitor* (*VMM*). Sensitive information is protected against the management VM between them. The VMM intercepts the reads of virtual devices by a user VM and decrypts the inputs, whereas it intercepts the updates of a framebuffer by a user VM and encrypts the pixel data. We have implemented FBCrypt for para-virtualized and fully-virtualized guest operating systems in Xen and TightVNC. Then we confirmed that any keystrokes or pixel data did not leak.

Keywords: virtual machine, remote management, information leakage

1. 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 software such as Virtual Network Computing (VNC). To allow users to access their systems even on failures inside their VMs, IaaS often provides *out-of-band remote management* via a special VM called the *management VM*. Unlike traditional remote management, management servers are run in the management VM, not in user VMs, and directly interact with virtual devices for user VMs, such as virtual keyboard and video devices. Even if the networks of user VMs are disconnected due to configuration errors or if the systems crash in user VMs, the users can continue to manage their VMs.

However, this out-of-band remote management increases security risks because the management VM is not always trustworthy in IaaS [1], [2], [3], [4], [5]. 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 [6]. Such attackers can easily eavesdrop on the inputs and outputs in remote management by replacing the management servers with malicious ones. For example, they can extract passwords from keystrokes sent from the clients and take screenshots of user VMs to steal sensitive or private information. In addition, they may execute arbitrary commands inside user VMs by sending keyboard events.

To solve this security issue, we propose *FBCrypt*, which protects sensitive information in out-of-band remote management against the attackers in the management VM. FBCrypt encrypts the inputs and outputs in remote management between a VNC client and a user VM using the *virtual machine monitor* (*VMM*). It can prevent information leakage via the management VM between them in a manner transparent to a user VM. The inputs to a user VM are encrypted by a VNC client and decrypted by the VMM when a user VM reads them from virtual devices. When a user VM updates a framebuffer in a virtual video device, the pixel data are encrypted by the VMM and decrypted by a VNC client. As such, only encrypted data are passed to the management VM. In addition to the confidentiality, the VMM checks the integrity of the inputs. It verifies the message authentication code generated from inputs and detects tampering before passing the inputs to a user VM.

To guarantee the integrity of the VMM itself inside IaaS, FBCrypt performs remote attestation of the VMM with a trusted server outside IaaS. Remote attestation certifies the authenticity of the VMM by tamper-resistant hardware such as the trusted platform module (TPM) [7]. Although the management VM often has high privileges, the code and data of the VMM are still protected against the management VM. Thanks to the memory protection by the VMM, the attackers in the management VM cannot steal secret keys for encryption in the VMM or modify the code in the VMM to invalidate the proposed security mechanisms.

[1] Kyushu Institute of Technology, Iizuka, Fukuoka 820–8502, Japan
[2] JST, CREST, Kawaguchi, Saitama 332–0012, Japan
[a)] egawan@ksl.ci.kyutech.ac.jp
[b)] naonishi@ksl.ci.kyutech.ac.jp
[c)] kourai@ci.kyutech.ac.jp

We have implemented FBCrypt for para-virtualized and fully-virtualized guest operating systems in the Xen VMM [8] and TightVNC [9]. To securely pass keyboard inputs to a para-virtualized guest, the VMM identifies the keyboard queue in the user VM and directly writes decrypted inputs into the queue. To encrypt pixel data on a screen for the management VM, the VMM replicates a framebuffer that holds pixel data for a user VM and provides the original and encrypted ones to the user VM and the management VM, respectively. It synchronizes these two if the user VM updates its framebuffer. Our experimental results show that the attackers in the management VM cannot steal keystrokes and screenshots and that the overhead of FBCrypt is not so large.

This paper is an extended version of our previous paper [10]. In this paper, we add support for fully-virtualized guest operating systems to FBCrypt. Then we report the overhead of out-of-band remote management in full virtualization and compare it with that in para-virtualization.

The organization of this paper is as follows. Section 2 describes issues in out-of-band remote management using the management VM. Section 3 proposes FBCrypt for protecting sensitive information in out-of-band remote management and Section 4 explains the implementation details in Xen. Section 5 shows our experimental results. Section 6 describes related work and Section 7 concludes this paper.

2. Motivation

2.1 Out-of-band Remote Management

To manage a user VM in a cloud, the user usually connects a management client to a management server running in the user VM. This is called *in-band* remote management because the user directly accesses the user VM. Let us consider remote management through VNC. Whenever the user presses a key or pointer button or moves a pointing device, an input event is generated and sent from a VNC client to the server. When the user VM draws graphic objects in a screen, a framebuffer update is sent from the VNC server to the client in response to a request from the client. A framebuffer is an area of memory used to hold pixel data. Since the communication between the VNC client and server can be encrypted with a virtual private network or SSH tunneling, sensitive information in the inputs and outputs is protected. However, this in-band remote management is not dependable. If the user just fails the configurations of the network or firewall in the user VM, the VNC client cannot access the VM at all. If the system in the VM crashes, the user cannot obtain any information via VNC.

To increase dependability in remote management of the user VM, *out-of-band* remote management is desired. In this style of remote management, a VNC server is run in the *management VM*, as illustrated in **Fig. 1**. The management VM is provided in the type-I VMM, which runs directly on hardware, such as Xen and Hyper-V and has privileges for accessing all user VMs. It also emulates virtual devices for each user VM. The VNC server in the management VM can directly access the virtual devices to interact with a user VM. This out-of-band remote management does not rely on the network or the VNC server in the user VM. The user can access the user VM as if he locally logged in to the VM even on network failures of the VM. For example, if the user

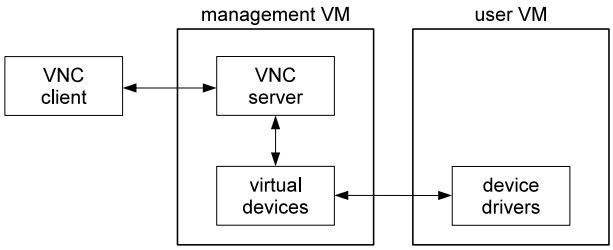

Fig. 1 Out-of-band remote management of a user VM.

fails in network configuration in the user VM, he could fix the problems by modifying the configuration through the virtual keyboard. Even when the system in the user VM crashes, the user may check kernel messages through the virtual video device.

This out-of-band remote management relies on the management VM, but the management VM is not always trustworthy in clouds [1], [2], [3], [4], [5]. Since the 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 [6].

If such attackers abuse the privileges of the management VM, they can eavesdrop on or tamper with the inputs and outputs in remote management. Even if the network is encrypted between a client host and the management VM, the data processed by the VNC server in the management VM is not encrypted. For the inputs to a user VM, the attackers can easily obtain keyboard and pointer inputs by modifying the VNC server. The VNC server has to receive the input events from the client and write them to the virtual devices for the user VM. For example, the attackers can extract passwords and credit card numbers from keystrokes. In addition, they can send keyboard events to a user VM and make the VM execute arbitrary commands.

For the outputs from a user VM, the attackers can take screenshots of the user VM through the framebuffer in a virtual video device. The attackers may steal sensitive information displayed on the screen. For example, when passwords have to be written in configuration files, the attackers can obtain displayed passwords even if they cannot eavesdrop on keyboard events to the user VM. If the user uses a software keyboard to avoid keyloggers, the attackers can steal information on pressed keys displayed on the screen.

2.2 Threat Model and Assumptions

We assume that the management VM can be compromised by outside attackers or abused by IaaS administrators. 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 attempts to steal and modify sensitive information sent between a VNC client and a user VM in out-of-band remote management.

We assume that IaaS providers themselves are trusted. This assumption is widely accepted [1], [2], [3], [4], [5]. To guarantee the trustworthiness, a small number of trusted administrators are

responsible for the maintenance of the VMM and the hardware. If average administrators that may be lazy or malicious maintain the VMM or the hardware, trusted administrators should verify it. Consequently, the VMM is well-maintained and has no vulnerabilities. Also, we do not consider physical attacks because server rooms should be strictly protected in data centers.

3. Secure Out-of-band Remote Management

To solve the security issue caused by using the untrusted management VM in IaaS, we propose *FBCrypt* for enabling secure out-of-band remote management.

3.1 FBCrypt

FBCrypt encrypts the inputs and outputs between a VNC client and a user VM using the VMM in a manner transparent to the user VM. The attackers in the management VM between them cannot steal sensitive information included in the interaction. **Figure 2** shows the architecture of FBCrypt. When the user generates an input event, the VNC client encrypts the input with a stream cipher and sends it to the VNC server in the management VM. The VNC server writes the encrypted input to the corresponding virtual device such as keyboard and pointing devices. When the user VM reads the encrypted input from the virtual device, the VMM intercepts this read and decrypts the input. It also checks the integrity of the input, that is, whether the input is not tampered with after being sent from the VNC client. The existing operating system in the user VM can read the decrypted input from the virtual device via the traditional interface.

In FBCrypt, the attackers in the management VM and in the network cannot eavesdrop on keyboard or pointer inputs sent from a VNC client. They cannot decrypt any inputs because only the VNC client and the VMM share a session key for encryption. In addition, they cannot send arbitrary keystrokes to user VMs to execute malicious commands. The encryption and integrity check of inputs by FBCrypt prevent the attackers from generating their own input events. Also, the attackers cannot reuse encrypted inputs to perform replay attacks. Thanks to the stream cipher used by FBCrypt, encrypted inputs captured by the attackers cannot be decrypted correctly when they are sent to the VNC server later. The stream cipher encrypts even the same message in a different way. FBCrypt can detect such inputs that are not correctly decrypted by the integrity check.

For the outputs from a user VM, on the other hand, FBCrypt encrypts the framebuffer of a virtual video device in the management VM. When an application such as an X server in a user VM draws graphic objects, the user VM updates the framebuffer by accessing the device. The VMM intercepts this update and encrypts the updated pixel data. In response to a request from the client, the VNC server reads the framebuffer and sends encrypted pixel data to the client. Then the VNC client decrypts the received pixel data and draws it in its window. Encrypting the framebuffer does not cause any problems because the VNC server is not aware of the contents of the framebuffer. It simply deals with encrypted pixel data as if they were not encrypted. Since the virtual video device provides the same interface to the user VM, no modification to the operating system is needed.

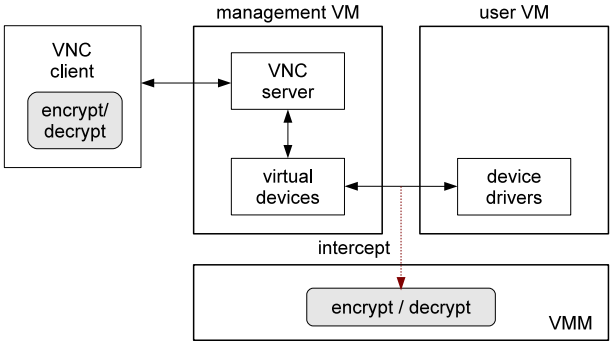

Fig. 2 The architecture of FBCrypt.

The attackers in the management VM and in the network cannot eavesdrop on framebuffer updates sent to a VNC client. The sent updates are a part of the encrypted framebuffer, which can be decrypted only by either the VMM or the VNC client. The attackers in the management VM can directly access the entire framebuffer but cannot decrypt it. In addition, they cannot modify the encrypted framebuffer arbitrarily. Even if they copy some area in the encrypted framebuffer to other areas, the copied areas cannot be decrypted correctly because FBCrypt encrypts pixel data with the information on their positions. Furthermore, malicious framebuffer updates generated by the attackers can be easily detected. Since such updates cannot be decrypted correctly, meaningless objects are just drawn in its window. Consequently, the user could notice such attacks soon.

Instead of the VMM, device drivers in user VMs could decrypt inputs and encrypt outputs, but this is not realistic in clouds. Since the systems in user VMs are managed by end users, it is difficult for cloud providers to force the users to install special device drivers. Moreover, device drivers cannot always be modified, particularly, for commercial operating systems such as Windows.

3.2 Protecting FBCrypt in IaaS

To guarantee the integrity of the VMM in IaaS, FBCrypt performs remote attestation of the VMM with a trusted server outside IaaS. Remote attestation certifies the authenticity of the VMM by tamper-resistant hardware such as the trusted platform module (TPM) [7]. It measures the VMM by calculating its hash value, sends the signed measurement to the trusted server, and verifies its integrity. The VNC clients can check the integrity of the VMM when connecting to the VNC server in the management VM. According to our assumption in Section 2.2, only a small number of trusted administrators are allowed to register the hash value of a legitimate VMM to the trusted server for remote attestation.

The VMM is protected even against malicious management VMs by using the protection mechanisms of the VMM itself. The management VM usually has high privileges and can access most of the hardware without limitations. However, the management VM is still a sort of VM. Similar to the other VMs, it cannot access the state of the CPUs or the memory used by the VMM. Therefore, the attackers in the management VM cannot tamper with the code in the VMM to invalidate the proposed security mechanism. They cannot steal data in the VMM, such as secret keys for encryption.

In addition, the memory and the CPU state of a user VM can be protected against the management VM by using the secure runtime environment (SRE) [2], [5] or VMCrypt [4]. The management VM can usually access all the resources of a user VM to enable VM management such as migration. The SRE and VMCrypt encrypt the memory of a user VM only for the management VM. The SRE encrypts the CPU state as well. With them, the management VM cannot steal keyboard and pointer inputs read by a user VM via the memory or its CPU registers. It cannot read unencrypted pixel data to be written in the framebuffer by a user VM. Also, it cannot tamper with the code in a user VM so that the user VM itself sends such sensitive data to the attackers, for example.

3.3 Key Management

A VNC client securely shares a session key with the VMM whenever it establishes a connection to a VNC server. When it connects to a user VM, it first generates a new session key. Then the VNC client communicates with the trusted server for remote attestation and checks that the VMM on which the user VM runs is legitimate. If so, the VNC client can obtain the public key of the VMM from the server. We assume that the public key is securely registered to the server in advance. Next, the VNC client encrypts the session key with the public key and transfers it to the VNC server in the management VM. The VNC server passes it to the VMM and the VMM decrypts it with its private key. The attackers in the management VM cannot decrypt the session key because they cannot obtain the private key of the VMM. The private key is sealed by TPM and can be unsealed only when a legitimate VMM is booted.

4. Implementation

We have implemented FBCrypt in Xen 4.1.1 [8] and TightVNC Java Viewer 2.0.95 [9]. We added only 5,497 lines of code to the VMM. In Xen, the management VM and a user VM are called domain 0 and domain U, respectively. A VNC server and virtual devices are a part of QEMU running in domain 0. As guest operating systems, FBCrypt supports para-virtualized Linux and fully-virtualized operating systems. The former is called a PV guest and is modified for running in VMs. The latter is called an HVM guest and can be run in VMs without any modification. We have confirmed that Linux 2.6.32 and 3.2.0, Windows 7 and FreeBSD 9.1 run with FBCrypt.

4.1 Keyboard Inputs

FBCrypt securely delivers user's inputs from a VNC client to a keyboard driver in domain U.

4.1.1 Delivery to PV Guests

For PV guests, user's inputs are delivered to the para-virtualized keyboard driver named kbdfront in domain U, as illustrated in **Fig. 3**. Currently, FBCrypt supports only keyboard inputs, but it can support pointer inputs in the same way. When the user presses a key, the VNC client sends an encrypted keyboard input to the VNC server and the VNC server writes it to a virtual keyboard device. Then the device passes it to the VMM using a new hypercall. In the hypercall, the VMM decrypts the

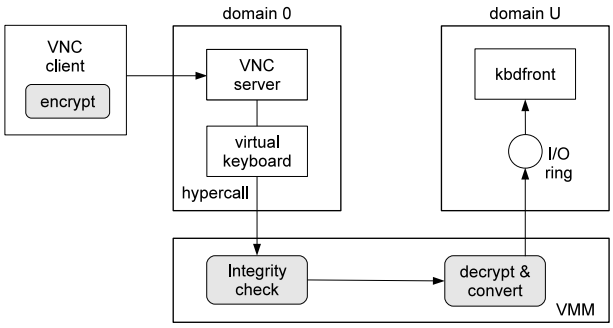

Fig. 3 The secure delivery of keyboard inputs to a PV guest.

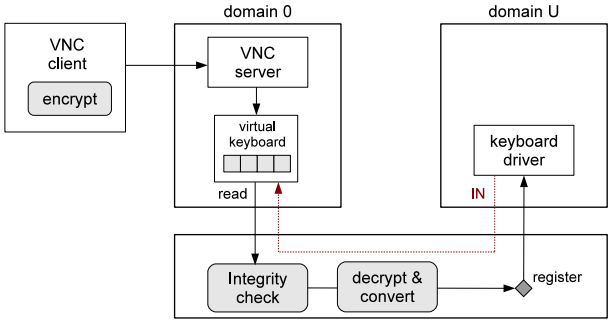

Fig. 4 The secure delivery of keyboard inputs to an HVM guest.

encrypted input and stores the decrypted one into the *I/O ring* for kbdfront. The I/O ring is a queue used for passing data between domains. From the I/O ring, kbdfront obtains the decrypted input. To prevent domain 0 from reading the contents of the I/O ring, the memory for the I/O ring can be encrypted by SRE [2], [5] or VMCrypt [4]. In the original Xen, virtual devices directly write unencrypted inputs to the I/O ring.

The VMM identifies this I/O ring when domain U is booted. Originally, the VMM does not recognize the I/O ring because only domain U and domain 0 share it. On initialization, kbdfront allocates and sets up the xenkbd page, which is a memory page containing the I/O ring. Then it registers the frame number of the page to XenStore in domain 0. XenStore is a filesystem-like database containing information shared between domains. The VMM monitors this registration from domain U to domain 0 and obtains necessary information. Domain 0 cannot interfere with this identification mechanism. We describe how to monitor the interaction in Section 4.3.

4.1.2 Delivery to HVM Guests

For HVM guests, user's inputs are delivered from a VNC client to a native keyboard driver in domain U, as in **Fig. 4**. When the VNC server receives an encrypted keyboard input, it writes the input to a virtual keyboard device. Unlike that for a PV guest, the device contains a keyboard queue in it and stores the input into the queue without decryption, so that domain 0 cannot eavesdrop on the contents of the input. Then the device generates a keyboard interrupt to domain U. When the keyboard driver attempts to read the I/O port for the device, the IN instruction is intercepted by the VMM. To emulate that instruction, the VMM invokes the device and obtains the encrypted input stored in the keyboard queue. Then it decrypts the input and stores the value into a virtual register of domain U as a result of the emulation of the IN instruction. The register can be encrypted by SRE [2], [5] and read only by

domain U and the VMM.

4.1.3 Conversion of Encoding

For keyboard inputs, the VMM converts the encoding of keys when it stores decrypted inputs into the I/O ring in PV guests or the virtual register in HVM guests. The I/O ring and the IN instruction for reading a keyboard device are designed so that they receive keycodes as keyboard inputs. A keycode (scancode) is data generated when a key is pressed and released. However, decrypted inputs are keysyms, which are used in VNC. A keysym is defined by the X Window System and the same as the corresponding ASCII value for ordinary keys. To fill this gap, the VMM converts keysyms to keycodes using the mapping table. This conversion was originally done by the VNC server. In FBCrypt, the VNC server cannot perform this conversion because keyboard inputs received from the VNC client are encrypted.

4.1.4 Encryption

For encryption and decryption of inputs, FBCrypt uses AES in CTR mode (AES-CTR) as a stream cipher. We used a modified version of the CyaSSL library [11] in the VMM and the Java standard API in TightVNC Java Viewer. AES-CTR generates a key stream by encrypting counter block values with AES and encrypts data by combining them and the next key in the key stream via XOR. When AES-CTR uses up all keys in the key stream, it increments the counter values and generates a new key stream from them. When the VNC client terminates, the internal state such as a key stream is lost. However, the VMM cannot recognize that termination and it continues to preserve the internal state. To synchronize the internal state when the VNC client reconnects to the server, the VNC server issues a new hypercall for resetting the internal state in the VMM.

4.1.5 Integrity Checking

To check the integrity of inputs, FBCrypt uses a message authentication code (MAC). When the VNC client sends an encrypted input to the server, it calculates a SHA-1 hash value from the input before encryption, a sequence number, and the session key for encryption. The sequence number is incremented whenever one input is handled. Thanks to a secret session key and a sequence number, the attackers cannot calculate the hash value correctly or reuse a pair of captured encrypted input and MAC value.

To send the MAC value with an encrypted input, we have extended the RFB protocol [12] used in VNC. For PV guests, the MAC value is passed from the virtual keyboard device to the VMM with the corresponding encrypted input. For HVM guests, it is stored in the VMM until the corresponding encrypted input is read to emulate the IN instruction for reading a keyboard device. Before the VMM decrypts the encrypted input, it compares the MAC value with the hash value calculated from the input. If the two values are different, the VMM discards the input.

4.2 Video Outputs

FBCrypt securely delivers updated pixel data from a video driver in domain U to a VNC client.

4.2.1 Delivery from PV Guests

For PV guests, updated pixel data are delivered from the paravirtualized video driver named fbfront in domain U, as illustrated in **Fig. 5**. In the original Xen, fbfront allocates a virtual framebuffer (VFB) in domain U and shares it with the virtual video device in domain 0. In FBCrypt, the VMM replicates the VFB for the device and encrypts the replicated one. When an application in domain U draws graphic objects, fbfront updates its own VFB and sends an update event to the device. At this time, the VMM synchronizes the original VFB with the replicated one. The VNC server periodically monitors updates in the replicated VFB and sends framebuffer updates to the client if pixel data are changed. The client decrypts the received data and re-draws its window. Since the original VFB in domain U can be encrypted only for domain 0 by SRE [2], [5] or VMCrypt [4], domain 0 cannot read unencrypted pixel data in the VFB.

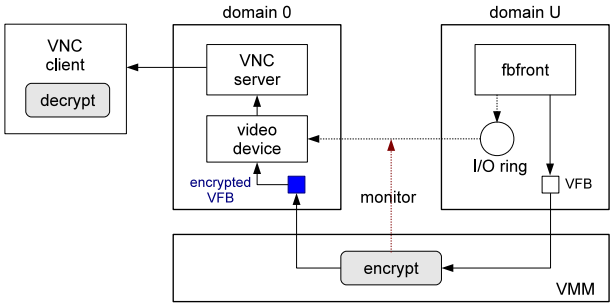

Fig. 5 The secure delivery of pixel data from PV guests.

To replicate the VFB for the virtual video device, the VMM first identifies the VFB in domain U. When domain U is booted, fbfront sets up the xenfb page, which contains the pointer to the VFB. When fbfront registers this page to XenStore in domain 0, the VMM intercepts it and allocates a new VFB as an encrypted replica. At the same time, it rewrites the xenfb page so that the page points to the replicated VFB. Since the xenfb page stores a VFB in a structure like page tables, the VMM constructs a replicated VFB with the same structure. Then it sends the rewritten xenfb page to XenStore in domain 0 and the virtual video device uses the replicated VFB. As such, the device is given an illusion as if it shared the same VFB with domain U.

The VMM synchronizes the original VFB with the replicated one when fbfront sends update events for the VFB to the virtual video device. An update event consists of an updated area in the VFB and is passed via the I/O ring shared between fbfront and the device. Since the xenfb page also contains the I/O ring, the VMM can easily identify it. When an update event is sent using the I/O ring, the VMM intercepts it, encrypts the pixel data in the specified area of the original VFB, and writes it to the replicated VFB. After this synchronization, the VMM sends the update event to the virtual video device.

4.2.2 Delivery from HVM Guests

For HVM guests, updated pixel data are delivered from the native video driver in domain U to a VNC client, as in **Fig. 6**. Originally, a virtual video device in domain 0 allocates video RAM (VRAM) in domain U and shares it with the video driver. This emulates memory-mapped VRAM for domain U. VRAM is the same as a framebuffer, but we use this term for HVM guests. In FBCrypt, the VMM replicates the VRAM for the virtual video device and encrypts the replicated one. This is similar to the replication of VFBs for PV guests. When an application in domain U

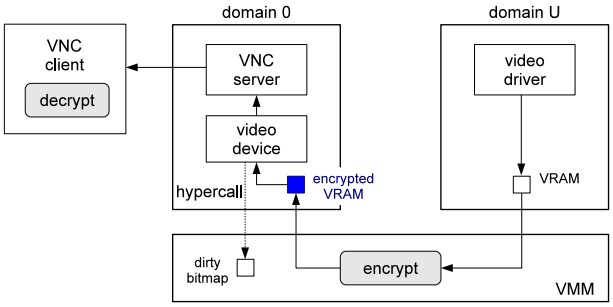

Fig. 6 The secure delivery of pixel data from HVM guests.

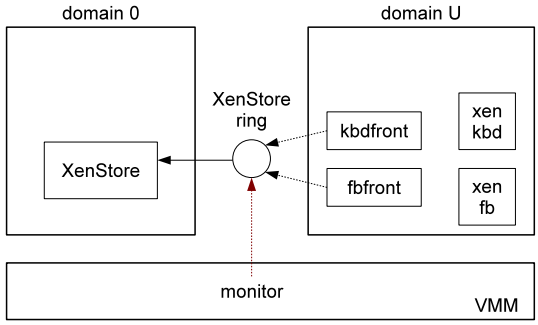

Fig. 7 Intercepting the registration to XenStore.

draws graphic objects, the video driver updates its own VRAM. Unlike VFBs, the VMM synchronizes the original VRAM with the replicated one when the device checks updated pixel data. This is because the native video driver does not send any update events to the device explicitly.

When the virtual video device attempts to map the VRAM of domain U for sharing it, the VMM replicates the pages used for the original VRAM and maps the replicated pages. To distinguish the mapping of VRAM and other memory regions in domain U, the device notifies the VMM of the physical address and the size of VRAM by using a new hypercall. Unlike VFBs, it is difficult that the VMM identifies the VRAM without such involvement of the device because the device determines the physical address of the VRAM. If a malicious device notifies the VMM of an address different from the VRAM, it could directly map the original VRAM without replication. However, memory regions that have not been registered as VRAM can be encrypted by the VMM [2], [4], [5].

The VMM synchronizes the original VRAM with the replicated one when the virtual video device obtains a dirty bitmap from the VMM. A dirty bitmap is used for tracking updates to VRAM in the VMM. Each bit of the bitmap corresponds to each page in VRAM. It is set when the corresponding page is updated. This mechanism is also used for tracking dirty pages and sending only modified pages to the destination in live migration [13]. When the device issues the hypercall for obtaining the dirty bitmap, the VMM copies the contents of only dirty pages in the original VRAM to the replicated one with encryption.

4.2.3 Encryption

For encryption and decryption of pixel data, FBCrypt uses a modified version of RC5 [14], which uses the non-standard block size of 48 bits to accommodate two pixels. It is desirable to encrypt a VFB or VRAM by one pixel because arbitrary pixels can be updated. However, the block size of 24 bits is less than the standard minimum size of 32 bits and therefore we chose 48 bits as the block size. In addition, FBCrypt considers the position of each pixel to be encrypted. If all the pixels were encrypted by the same key without considering their positions, the attackers in domain 0 could obtain approximate screen images.

Since FBCrypt encrypts two pixels together, the VMM expands synchronized regions so that the regions are aligned by the boundary of two pixels. For example, when the X-position or the width of an updated area is odd, the VMM has to decrease the X-position by one or increase the width by one. Also, the VNC server aligns updated areas in a similar way because it calculates the areas by itself. If the VNC server transferred only one of two pixels, the VNC clients could not decrypt the received pixel data correctly.

4.3 Monitoring a XenStore Ring

To intercept the registration to XenStore in domain 0 from a PV guest in domain U, the VMM monitors a XenStore ring, which is shared between domain U and domain 0, as shown in **Fig. 7**. When domain U registers data to XenStore, it writes a pair of path and value to the XenStore ring. For example, the path for a virtual keyboard device is `device/vkbd/0/page-ref` and the value is the frame number of the xenkbd page. The VMM inspects the XenStore ring when domain U sends an inter-domain event to domain 0 using a hypercall after it writes data to the ring. If the written path is the registered one, the VMM obtains the information on a shared page.

Since the original VMM in Xen is not aware of a XenStore ring, our VMM identifies the ring from the start_info page, which points to the page containing the XenStore ring. This page is passed from domain 0 to domain U when domain U is booted. The VMM first obtains the virtual address of the start_info page from the RSI register of a virtual CPU for domain U. The address is set by domain 0 at the build time of domain U. Then the VMM translates the virtual address into the frame number of the start_info page.

5. Experiments

We conducted experiments for confirming the prevention of information leakage by FBCrypt and for examining its overhead. For server and client machines, we used two PCs with one Intel Core 2 Quad processor Q9550 2.83 GHz and a Gigabit Ethernet NIC. In the server machine, we ran a modified version of Xen 4.1.1 for the x86-64 architecture. For comparison, we also used the original Xen. We assigned one CPU core and 1 GB of memory to domain U and four cores and 3 GB of memory to domain 0. In domain U, we ran Linux 2.6.32.21 as PV and HVM guests and configured the screen resolution to 800×600. In domain 0, we ran Linux 3.1.1. In the client machine, we ran a modified version of TightVNC Java Viewer 2.0.95 on Linux 2.6.38.8. The client machine had 8 GB of memory.

We used AES-CTR with a 128-bit key for encrypting inputs and SHA-1 for calculating MAC. For RC5 used for encrypting pixel data, we used a 48-bit block, a 192-bit key, and 16 rounds.

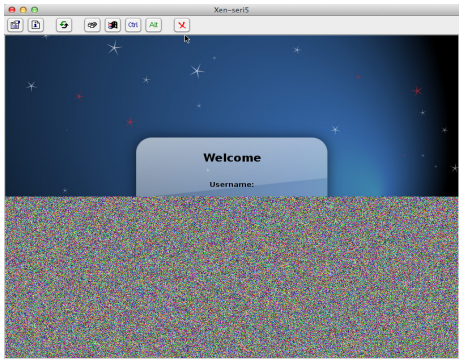

Fig. 8 The screen whose only lower half is encrypted.

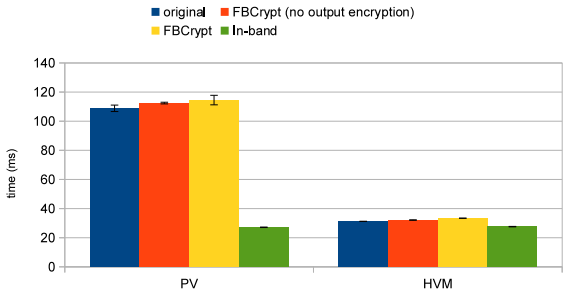

Fig. 9 The response time of a keyboard input.

5.1 Attempts at Eavesdropping

To confirm that FBCrypt prevents domain 0 from eavesdropping on keystrokes, we embedded a custom keylogger into the VNC server in domain 0. This keylogger simply recorded keystrokes sent from the VNC client. Using the VNC client, we logged in to domain U by typing a user name and a password. Without FBCrypt, the plain-text password was recorded. When FBCrypt was enabled, the password was encrypted by the VNC client and the keylogger recorded encrypted one. Nevertheless, the user could log in to domain U as usual.

Next, we embedded a program for screen capture into the VNC server to confirm that FBCrypt prevents domain 0 from stealing the pixel data of domain U. This screen capture periodically saved the pixel data in the VFB to a file. **Figure 8** shows the screen whose lower half is encrypted for demonstration. With FBCrypt disabled, the pixel data was recorded as in the upper half and the attackers could read displayed texts. FBCrypt could randomize the pixel data as in the lower half, so that the attackers cannot recognize the contents.

5.2 Keyboard Inputs

5.2.1 Overhead

First, we examined the overhead on the client and server sides when we pressed one key in the VNC client. On the client side, we measured the time from when the VNC client received a keyboard input until it sent the input event to the VNC server. On the server side, for the PV guest, we measured the time from when the VNC server received the event until the input was stored into the I/O ring. For the HVM guest, we measured the time until the input was stored into a virtual register by the emulation of the IN instruction. We measured these 10 times and obtained the average. The overhead on the client side was 794 μs. Most of the overhead comes from sending extra data for the MAC. On the server side, the overheads were 32 μs and 16 μs for the PV and HVM guests, respectively.

Next, we measured the CPU utilization on the client and server sides when the VNC client generated keyboard inputs at various rates. To change the rate of keyboard inputs, we used the keyboard auto-repeat function. Even when we increased the auto-repeat rate to 12 characters per second, the CPU utilization did not increase.

5.2.2 Response Time

To examine the response time of a keyboard input, we measured the time from when the VNC client received a keyboard input from the user until it drew an input character in its window. In this experiment, domain U read the keyboard input, displayed the character corresponding to it on the terminal, and updated the framebuffer. We measured the response time 10 times and obtained the average and standard deviation. We performed this measurement with and without the encryption of video outputs to reveal the breakdown of the overhead. For comparison, we also measured the response time using in-band remote management, which ran a VNC server in domain U, not in domain 0.

Figure 9 shows the response times in two types of guests. For the PV guest, the response time in FBCrypt was 5.6 ms longer than that in the original Xen. The overhead of encrypting video outputs was 2.0 ms. For the HVM guest, the response time increased only by 2.2 ms in FBCrypt. The overhead of the encryption of pixel data was 1.3 ms. These results means that the increase of the response time by FBCrypt is not large. For in-band remote management, on the other hand, the response time was 27 ms and the fastest. One cause of the difference is the overhead of out-of-band remote management itself.

However, the cause of the difference between the response times in the PV guest and the others is the timer interval used by the VNC server implemented in Xen. The VNC server checks updates in the framebuffer at some interval and sends the updates to the client if any. The interval is first set to 30 ms when the VNC server receives a keyboard event. Since the framebuffer update caused by the keyboard input does not occur in 30 ms for the PV guest, the next interval is set to 80 ms. This means that the VNC server can recognize a framebuffer update in 110 ms after a keyboard event is received. This is the reason why the response times in the PV guest are approximately 110 ms. When we changed the timer interval to 10 ms, the response time was improved to 86 ms in the PV guest with FBCrypt.

5.3 Screen Updates

5.3.1 Overhead

First, we examined the overhead on the server and client sides when we updated screen areas of various sizes in a user VM. On the server side, for the PV guest, we measured the time from when the VMM intercepted the update event for the VFB until it completed to synchronize VFBs. This time included the encryption time of pixel data. For the HVM guest, we measured the time for synchronizing VRAM in the hypercall that a virtual video device issued to obtain a dirty bitmap. On the client side, we measured the time from when the VNC client received encrypted pixel data

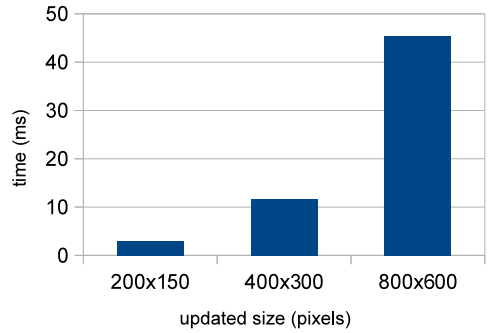

Fig. 10 The overhead in screen updates (client).

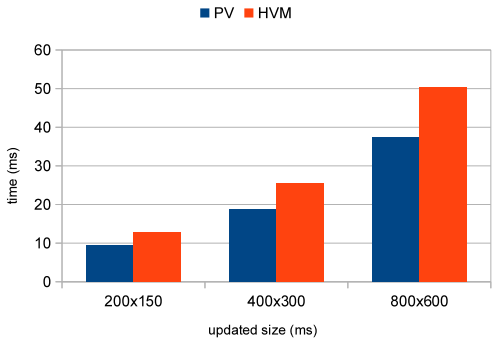

Fig. 11 The overhead in screen updates (server).

until it completed to draw the data in its window. We measured these 10 times and obtained the average.

Figures 10 and **11** show the overhead in screen updates of various sizes. On the client side, the overhead was proportional to the number of updated pixels. On the server side, on the other hand, the overhead was proportional to half of the number of updated pixels in both PV and HVM guests. For the PV guest, the fbfront driver sends an event with an updated area to the virtual video device after it updates the framebuffer, as described in Section 4.2.1. At this time, it expands the width of the updated area to the screen width (800 pixels in this experiment) due to its implementation issue. Since FBCrypt synchronizes that expanded area in VFBs, the overhead is proportional only to the height of an updated area.

For the HVM guest, on the other hand, FBCrypt synchronizes VRAM at the page level on the basis of a dirty bitmap, as described in Section 4.2.2. Each page includes many pixel data (1365.3 pixels in this experiment) in the direction of the X axis. As a result, an updated area is expanded to fit the page size and its width is usually the same as the screen width, as in the case of the PV guest. Therefore, the overhead is approximately proportional to the height of an updated area. In any cases, the VNC server calculates a truly-updated area by comparing with the old pixel data and sends it to the client.

Next, we measured the CPU utilization when we updated screen areas of various sizes in domain U. For this experiment, we have developed a benchmark program that repeated to re-draw the specified area with random colors at 30 frames per second. **Figure 12** shows the results on the client side and **Figs. 13** and **14** show those on the server side for PV and HVM guests, respectively. The CPU utilization increased as the updated area became large. On the client side, the CPU utilization increased by 6.6% at maximum.

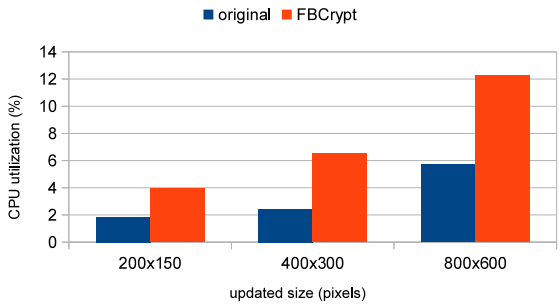

Fig. 12 The CPU utilization in screen updates (client).

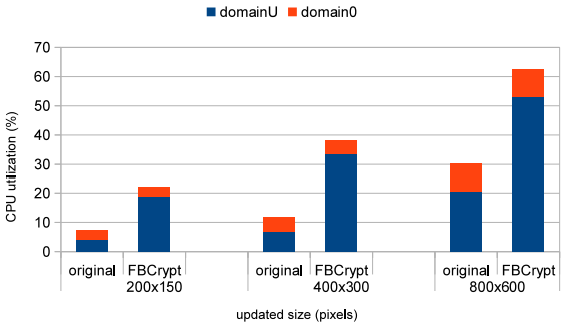

Fig. 13 The CPU utilization in screen updates (server for the PV guest).

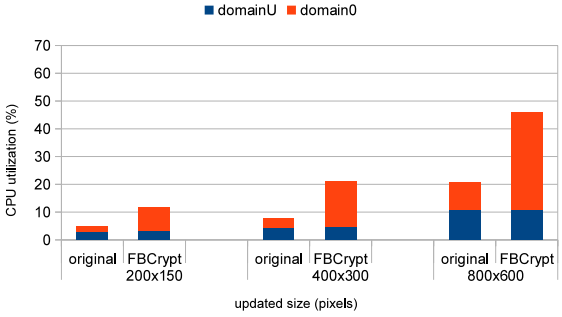

Fig. 14 The CPU utilization in screen updates (server for the HVM guest).

On the server side, for the PV guest, the CPU utilization largely increased in domain U because the synchronization of VFBs is done when the fbfront driver in domain U issues a hypercall to send update events. For the HVM guest, on the other hand, the CPU utilization largely increased in both domain 0 and domain U when we used FBCrypt. The synchronization of VRAM is done when the virtual video device in domain 0 checks a dirty bitmap using a hypercall. In a full-screen update (800×600) using FBCrypt, the CPU utilization increased by 32% and 72% in PV and HVM guests, respectively.

5.3.2 Response Time

To examine the response time of screen updates of various sizes, we measured the time from when the VNC client sent a request until it received updated pixel data and re-drew it in the window. We ran the above benchmark program in domain U so that the framebuffer was always being updated. We measured the response time 10 times and obtained the average and standard deviation. For comparison, we also measured that in in-band remote management.

Figures 15 and **16** show the response times. For the HVM guest, the increase of the response time by FBCrypt was approx-

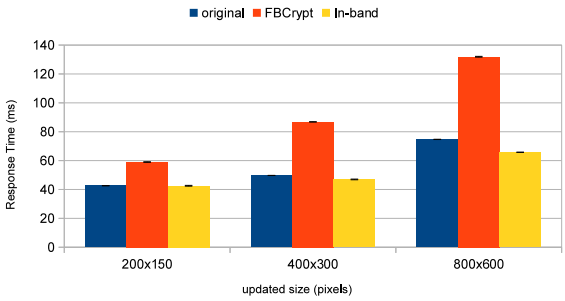

Fig. 15 The response time of a screen update (HVM guest).

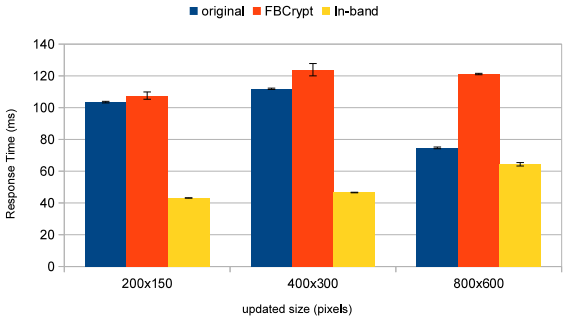

Fig. 16 The response time of a screen update (PV guest).

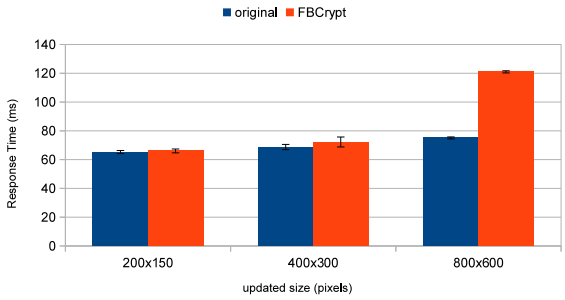

Fig. 17 The response time of a screen update (for the PV guest with a short timer).

imately the sum of the overhead on the server and client sides. For the PV guest, on the other hand, the increase was only the overhead on the client side. This is because the synchronization of VFBs is not done while the VNC server handles requests from the client. That is done when the fbfront driver updates the VFB. Therefore, the increase of the response time is smaller for the PV guest.

Compared with the results for the updates of three different areas, for the HVM guest, the response times increased as the updated area became larger. For the PV guest, on the other hand, they increased when the updated area was small but decreased for large updated areas. The cause is also the timer interval used by the VNC server in Xen, as in Section 5.2.2. The fbfront driver is designed to send an update event to the virtual video device in 50 ms after the VFB is updated. Due to this delay, the device may receive no update events when the VNC server receives a request from the client. In this case, the VNC server has to wait for 80 ms at least. The probability in which this situation happens becomes higher as an updated area gets smaller. Since the VNC client can process a smaller amount of pixel data in a shorter time, it sends requests to the server more frequently.

Figure 17 shows the response times for the PV guest when we changed the timer interval in the VNC server to 10 ms. The response time was improved for the updates of smaller areas and proportional to the size of an updated area.

6. Related Work

Xoar [15] runs QEMU including a VNC server in a separated VM called QemuVM. In Xen, QEMU can be run in a special VM called a stub domain. Since a small operating system named mini-os is run in these special VMs, it is more difficult for the outside attackers to compromise them. However, the VNC server can be compromised because it is open to the Internet. If it is compromised, the attackers can eavesdrop on sensitive information in remote management. In addition, this architecture does not improve the security against insider attacks by IaaS administrators.

VMware vSphere Hypervisor [16] runs a VNC server in the VMM and enables out-of-band remote management without the management VM. The VNC server in the VMM can directly access virtual devices for user VMs. In vSphere, information leakage via the management VM does not occur. However, the attackers can steal sensitive information in remote management if they compromise the VNC server in the VMM. They can take over even the control of the VMM itself. FBCrypt preserves the confidentiality in remote management by the VMM even in the case that the VNC server in the management VM is compromised.

The secure runtime environment (SRE) [2], [5] and VMCrypt [4] prevent information leakage from the memory of the 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 FBCrypt in that it prevents the management VM from stealing information inside the user VMs. Note that the management VM cannot use the VFB encrypted by these systems, instead of the one replicated and encrypted by FBCrypt. These systems synchronize the unencrypted and encrypted VFBs only on memory mapping.

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 the inputs and outputs in remote management.

BitVisor [17] can prevent information leakage from storage and network of the user VM. It is similar to FBCrypt in that the VMM transparently encrypts I/O of the user VM without the help of the management VM. However, BitVisor does not provide a means of remote management.

7. Conclusion

In this paper, we proposed FBCrypt for enabling secure out-of-band remote management in IaaS clouds. To prevent information leakage via the management VM in out-of-band remote management, FBCrypt encrypts the inputs and outputs between a VNC client and a user VM using the VMM. The VMM decrypts the inputs encrypted by a VNC client when a user VM reads them. When a user VM updates a framebuffer, the VMM encrypts the updated pixel data, which are decrypted by a VNC client. As such, sensitive information is protected against the

management VM, which is located in the middle. We have implemented FBCrypt for para-virtualized and fully-virtualized guest operating systems in Xen and TightVNC Java Viewer. We confirmed that the security in out-of-band remote management was enhanced and that the overhead of FBCrypt was not so large. Our future work is to apply FBCrypt to other remote management software such as SSH.

Acknowledgments This research was supported in part by JST, CREST.

References

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

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

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

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

[5] Li, C., Raghunathan, A. and Jha, N.K.: A Trusted Virtual Machine in an Untrusted Management Environment, *IEEE Trans. Services Computing*, Vol.5, No.4, pp.472–483 (2012).

[6] TechSpot News: Google Fired Employees for Breaching User Privacy (2010), available from ⟨http://www.techspot.com/news/40280-google-fired-employees-for-breaching-user-privacy.html⟩.

[7] Trusted Computing Group: TPM Main Specification (2011), available from ⟨http://www.trustedcomputinggroup.org/⟩.

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

[9] TightVNC Group: TightVNC, available from ⟨http://www.tightvnc.com/⟩.

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

[11] yaSSL: CyaSSL Embedded SSL Library, available from ⟨http://www.yassl.com/⟩.

[12] Richardson, T.: The RFB Protocol Version 3.8, available from ⟨http://www.realvnc.com⟩.

[13] Clark, C., Franser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I. and Warfield, A.: Live Migration of Virtual Machines, *Proc. Symp. Networked Systems Design and Implementation*, pp.273–286 (2005).

[14] Rivest, R.L.: The RC5 Encryption Algorithm, *Proc. Workshop Fast Software Encryption* (1994).

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

[16] VMware Inc.: VMware vSphere Hypervisor, available from ⟨http://www.vmware.com/⟩.

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

Tomohisa Egawa received his B.Sc. and M.Sc. degrees from Kyushu Institute of Technology in 2011 and 2013, respectively. His current research interest is in cloud computing and virtual machines.

Naoki Nishimura received his B.Sc. degree from Kyushu Institute of Technology in 2012. His current research interest is in cloud computing and virtual machines.

Kenichi Kourai is an associate professor in Department of Creative Informatics at Kyushu Institute of Technology. He received his Ph.D. degree from the University of Tokyo in 2002. Before 2008, he was an assistant professor at Tokyo Institute of Technology. He has been working on operating systems. His current research interest is in dependable and secure systems using virtual machines.