# CryptVMI: A Flexible and Encrypted Virtual Machine Introspection System in the Cloud

Fangzhou Yao, Read Sprabery and Roy H. Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL, USA
{yao6, spraber2, rhc}@illinois.edu

## ABSTRACT

Virtualization has demonstrated its importance in both public and private cloud computing solutions. In such environments, multiple virtual instances run on the same physical machine concurrently. Thus, the isolation in the system is not guaranteed by the physical infrastructure anymore. Reliance on logical isolation makes a system vulnerable to attacks. Thus, Virtual Machine Introspection techniques become essential, since they simplify the process to acquire evidence for further analysis in this complex system. However, Virtual Machine Introspection tools for the cloud are usually written specifically for a single system and do not provide a standard interface to work with other security monitoring systems. Moreover, this technique breaks down the borders of the segregation between multiple tenants, which should be avoided in a public cloud computing environment. In this paper, we focus on building a flexible and encrypted Virtual Machine Introspection system, CryptVMI, to address the above concerns. Our approach maintains a client application on the user end to send queries to the cloud, as well as parse the results returned in a standard form. We also have a handler that cooperates with an introspection application in the cloud infrastructure to process queries and return encrypted results. This work shows our design and implementation of this system, and the benchmark results prove that it does not incur much performance overhead.

## Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection; E.3 [**Data Encryption**]: Miscellaneous

## General Terms

Security, Design, Management, Performance

## Keywords

Cloud Computing; Virtualization; Virtual Machine Introspection; Confidentiality;


*SCC'14,* June 3, 2014, Kyoto, Japan.

http://dx.doi.org/10.1145/2600075.2600078.

## 1. INTRODUCTION

As the size of data being stored and processed in industry is increasing drastically, the cloud computing shows its power. Building and maintaining storage, as well as computation infrastructures become trivial in a cloud environment. Some companies build their own private cloud services, and other companies including Netflix [4] and Dropbox [11], tend to use public cloud services, such as AWS from Amazon [3]. Virtualization has turned out to be necessary in both private and public cloud computing solutions, because it provides better utilization of resources and reduces the cost by allowing multiple instances of operating systems (OS) owned by multiple tenants to run concurrently on the same physical machine. Though multiple OSes are running in their own virtual machines (VM) and sharing the same hardware resources, the Virtual Machine Monitor (VMM) can still ensure high availability for users [8].

However, sharing the same physical resources brings up security issues. Even though a VM should only be allowed to access its own resources by design, the traffic between VMs essentially breaks down the physical isolation. The logical isolation in cloud environments is built in the software layer and, consequently, the security guarantees are weaker [13]. A compromised VM can easily and quickly spread malware and make the entire system vulnerable to more attacks. Moreover, it takes time for cloud service users to detect those compromises.

While traditional security systems, such as Intrusion Detection Systems (IDS), are able to enforce policies on every node in a network system and detect violations in real-time, the virtualization framework makes cloud environments complex, and hence it is difficult to collect evidence in such systems. These facts lead to the development of Virtual Machine Introspection (VMI) techniques. VMI tools inspect a VM from a trustworthy outside environment, so they are able to access the entire system and acquire the genuine state of a VM [21]. IDSes can be built with this technique over the cloud for greater attack resistance, while providing an excellent view of the states in VMs [15].

There are many private cloud infrastructures that have embedded VMI into their systems to enhance security [19], but those VMI frameworks are customized to specific environments, making it difficult to use them in cooperation with existing security monitoring systems, such as Bro IDS [29]. Concerns are also raised for public cloud systems, since the VMI technique might break down the borders of the segregation between multiple tenants [7]. Companies running their services on public cloud infrastructure would not want to

expose their application states to the cloud service provider. Moreover, the misuse of administrative authorizations might lead to even more attacks [9], and hence any opportunities that could be exploited to examine VM states, even by the cloud administrators, should be minimized.

VMI is a simple and convenient way for cloud users to acquire trustworthy information from their running VM instances, but the inflexibility to integrate with existing security monitoring systems and the possibility of exposing confidential information from VMs makes it difficult to apply this technique in the public cloud.

Therefore, we propose CryptVMI, an encrypted Virtual Machine Introspection system, to provide users complete status of their virtual instances, while keeping confidentiality from possible attackers, including administrators of cloud services. This system also provides users semistructured outputs as results, which could be extended to work with existing security monitoring systems on the user end to provide better security for their systems, especially when their cloud computing frameworks are running in the public cloud.

Our contributions are:

1. We design and implement a Virtual Machine Introspection system to provide a simple interface with semistructured data for users to extend this system with other security monitoring systems.

2. We show that CryptVMI keeps confidentiality of users' information from their VMs in the whole encrypted VMI process.

3. We demonstrate that our encryption scheme introduces minimum overhead to system performance by comparing CryptVMI with VMI tools without encryption.

The rest of this paper is organized as follows. We first explain our motivation of proposing such an encrypted VMI system in Section 2. Next, we show our design in Section 3 and implementation details in Section 4. We then evaluate our system through experiments and analyze the results in Section 5. In Section 6, we conclude our work and discuss future approaches.

# 2. MOTIVATION

Security management and policy compliance are critical issues in modern systems, however checking the state of a system as a whole is hard. It requires the security monitoring system to be located on the host, but this approach makes the security system susceptible to attacks. Alternatively, if we deploy the monitoring systems in the network, the view will be limited [15]. Livewire takes advantage of VMI techniques to retain the full view of a host-based IDS, but pulls the IDS outside of the host, namely the node in a network, for greater attack resistance [15]. In addition to visibility, an input framework built on top of Bro IDS was presented for flexibility and scalability. It provides a simple yet flexible user interface and support for asynchronous operation, which enables an IDS to analyze high-volume packet streams under soft real-time constraints [2]. Odessa also proposed a solution to build a resilient policy compliance system in the network. It is designed to distribute the evaluation of rules in a scalable fashion, which employs a set of monitoring agents on the nodes in the network and validates global rules in their corresponding verifiers [20]. Thus, we want to collect the evidence from the OS directly, while keeping the security system in a safe state. Furthermore, the evidence obtained should be standardized in a simple form for users or other security monitoring systems, like an IDS. The evidence should be also available for multiple analysis frameworks for stronger resilience.

Nowadays, many users start using cloud services due to the benefits of a flexible and elastic infrastructure. Virtualization techniques and bridged networks are widely used in such environments, which make the system even more complicated. It also leads to more security concerns, such as the weakened isolation and possible misuse of administrative permissions. We can use VMI tools to simplify this complex system, and obtain the entire state of a VM from the safer outside, namely the Dom0 or host for virtual systems running on Xen or KVM, respectively. However, many companies pay for public cloud services to store their data and run computational applications, because of the ease of configuration and maintenance. Thus, the concern for confidentiality becomes inevitable. For instance, if Amazon started providing VMI features for its cloud users, then the administrators in Amazon would be able to access the entire state of every customer's VM running in their public cloud. This is definitely not what Amazon's customers want to happen with respect to their privacy. Thus, we want to collect data with simple VMI techniques from VMs bypassing the complex cloud framework, but we also want to keep the confidentiality of any sensitive information obtained during this process.

Therefore, a VMI system in the cloud, which is able to respond to queries from multiple users or security monitoring systems with simple and standardized results would be beneficial. The concern for credibility of data gathered is minimized and the data can be as much as the whole picture of a VM, since VMI work is processed in a trustworthy outside environment. In addition, the whole process should be encrypted to address the concern of the possible leakage of users' data.

# 3. DESIGN

In this section, we present concepts that we used to design CryptVMI, as well as the architecture for this VMI system. First, we present our threat model. Then, we discuss the interface that we used for users' queries and results. Next, we describe our encryption and decryption scheme. Last, we show how we apply these concepts into the architecture of this system.

## 3.1 Threat Model

In the real world, the compute node should not be easily compromised, and hence we make the assumption that systems running VMs are trustworthy environments, which guarantee that the information gathered using VMI is genuine. If the compute node, which is essentially the host system in a virtualized environment, is compromised, it may result in misleading data. However, this should rarely happen in a mature commercial public cloud service system, as they utilize state-of-the-art electronic surveillance and multi-factor access control systems with a 24/7 monitoring service [3].

We also assume that the VM running in a compute node might be totally malicious. Even with this assumption, the impact of a malicious VM is still trivial, since we can obtain

the entire genuine state of the VM and detect the problem in time. For instance, inside of the malicious VM, a malware might be able to hide itself from a user's inspection, such as the `ps` command, but it cannot remove itself from the processes list of the OS running in the VM, otherwise it will not be effective. Even if attackers can exploit a compromised VM and manage to attack the VMM, VMMs like Xen [5] and KVM [18] are able to defend against such attacks with their own security mechanisms. In the worst case, the security of the cloud system will still detect the attack, though it might result in the loss of their data and suspending of their services for cloud users.

Users can access their VMs through a Secure Shell (SSH) connection, which should be secure enough to avoid most man-in-the-middle attacks. However, SSH to a compromised VM might lead to the infection of users' machines. We expect that users have anti-malware programs on their own OSes.

Furthermore, we presume that co-location attacks [27] might happen. Thus, we do not want to expose the structure of our cloud system to users, such as on which compute node a VM is located, or the IP address of the node. This approach should minimize the chances for such attacks, and it also keeps transparency to users.

Our objective is to prevent potential attackers, including cloud administrators, from knowing the entire state of a user's VM. We run our VMI applications with `root` authorization. We assume that the public cloud system has been has been configured in such a way that normal administrators do not have `root` authorization on compute nodes. This assumption is reasonable in reality for a public cloud, as they are in a group with only limited permissions interacting with the cloud service application programming interfaces (API) and VMs, otherwise the administrator can install their own VMI tools, even bypassing the logs in a cloud system. Therefore, configuration files and keys stored with `root` permission for CryptVMI also stay safe.

## 3.2 Flexible and Simplified Interface

We want our interface for users or other security monitoring systems to achieve good performance and scalability. Also, the query should be simple and easy to write, while the result is standardized and flexible. Additionally, considering that data transmission through the network is a major part of our system, we want to reduce the network throughput.

### *3.2.1 RESTful API*

Representational State Transfer (REST) is a data style designed for modern web and distributed applications. It ignores the details of component implementation and protocol syntax, in order to provide better performance and scalability of component interactions [14].

Many applications nowadays are using this type of API style, including IDSes [28]. This fact makes our VMI system easily extendable to other systems. We simply accept HTTP POST requests in an asynchronous fashion.

### *3.2.2 Query and Result in JSON*

We use Javascript Object Notation (JSON) for queries and results. Its semistructured schema makes JSON flexible for various types of data, since it does not have a regular structure [10]. Attributes in a JSON document are self-described, and hence it is clear for users to understand and easy to write. It also simplifies the structure of the document, compared to XML. Thus, using JSON reduces the throughput when transmitting data over the network.

A simple example of query in JSON from users or other security applications would be like below.

```
{
  "user": "[token]",
  "vm": "[vm_name]",
  "command": { ... }
}
```

The result is similar, but binary data, such as the memory dump, is encoded with Base64 [17].

## 3.3 Encryption and Decryption

Since the query itself might disclose the information [26], such as the `pid`, we should encrypt both the query and result. We use the symmetric key to encrypt and decrypt parts of the query and the result. Moreover, each user has a unique pair of public and private keys. We use the public key on the user end to encrypt the symmetric key, while decrypting it with the private key stored in the VMI application end. Each query is associated with a unique random symmetric key.

We use this scheme for two reasons. One reason is that public key encryption, namely the asymmetric key encryption, does not support encryption for large data by itself [1]. Thus, this scheme allows us to encrypt large data, such as a 1GB memory dump, and achieve the nearly equivalent security level as the asymmetric key encryption. CryptDB uses similar approach to allow for query execution on an encrypted datastore [26]. The other reason is for performance. RSA is not as performant as symmetric encryption schemes [1]. Additionally, modern Intel processors have embedded AES symmetric key encryption hardware accelerators, and applications can benefit greatly from using AES on these processors [16].

## 3.4 Architecture

The design of CryptVMI has three major components. Figure 1 shows the overview of this system. The first component is the client application on the remote user end. It accepts queries from users or other security applications, and directs queries to the handler. It also eventually decrypts results returned from the cloud. The second component handles queries sent from the client application. Once the query is processed, the encrypted data is transfered back to the client. The third component involves strategies to acquire desired results with encryption.

Though all three components are written in Ruby, the client application does not have to be run in `root`, since the users usually take control of their own machines and there are only the public keys stored at the their end. Each of these components is described in following subsections. Additionally, there is a series of VMI tools deployed on compute nodes, which are written in C.

### *3.4.1 User Client*

Once a query is accepted, the user client application will assign a unique ID and a random symmetric key $s$ to this query. The command is encrypted with the key $s$, and the key is encrypted with the user's corresponding public key $c$. The client then initiates the request to the management node in the cloud environment through a Secure Sockets

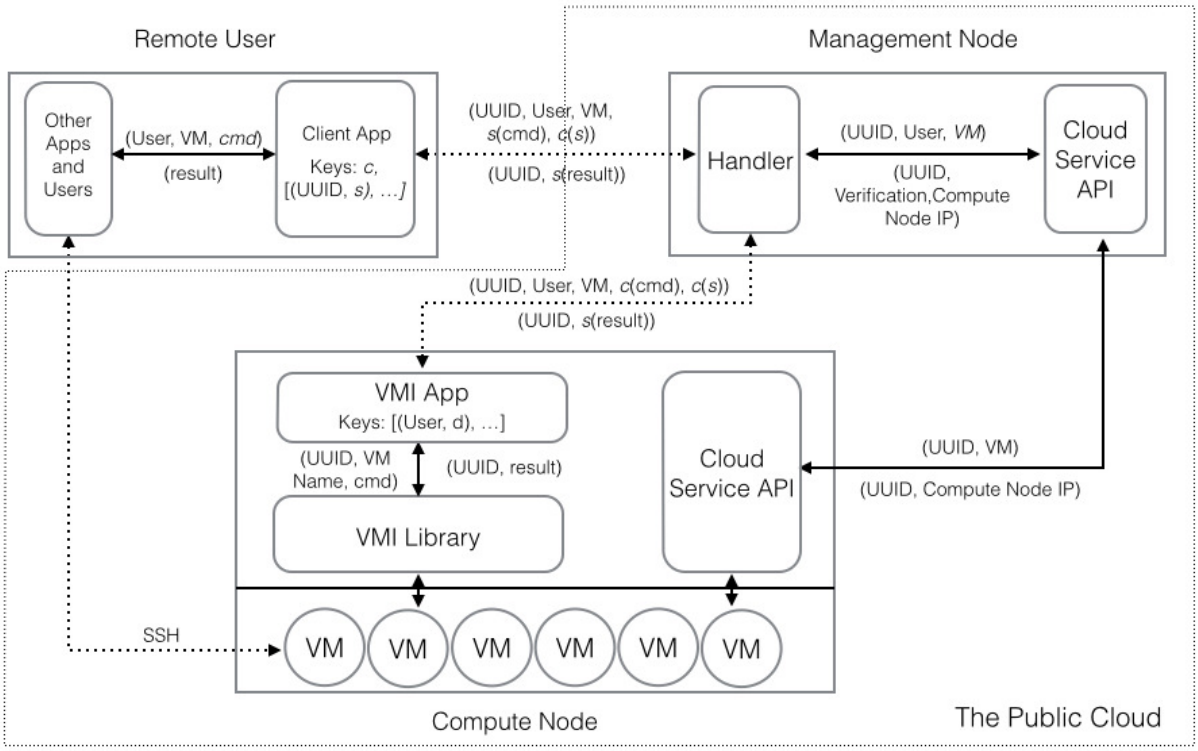


**Figure 1: A High Level View of Our CryptVMI Architecture. The SSH connection is not part of the VMI system, but it is a general approach that users communicate with their VMs. Dotted lines represent secure connections or connections with encrypted components. There is only one remote user and one compute node shown, but multiple clients and compute nodes are supported in our design.**

Layer (SSL) connection, whose address is loaded from the configuration file.

The client application handles the result in its callback function. Once the result is received, the application first decrypts it with key $s$ and then decodes it with Base64, if the query is requesting binary data, such as memory dumps. Eventually, it sends back the data to the query's origination.

Since the application is written in Ruby and might not be run in `root`, users can modify the code to parse the returned data in different formats, and hence make it easy to cooperate with various applications, such as IDSes.

### 3.4.2 Query Handler

The request received from the client has five parts: the unique ID of the query, the user's information including the credential token, VM instance name in the cloud system, symmetric key $s$ encrypted with user's public key $c$ and the command encrypted with key $s$. Once the handler receives this request, it first checks the user's token and the instance name with the cloud service API to verify if this user has access to the current tenant in the cloud and if there is such a VM associated with this tenant. The handler then uses the cloud service API to locate the IP address of the compute node that holds the designated VM, and obtains this VM's name within the hypervisor with the help of the introspection application deployed on the compute node. The name of a VM within the hypervisor is usually different from its instance name provided from the cloud service API to the user.

Communications in this process are not encrypted, because they are in the internal cloud system network, and the cloud administrator should already know the unencrypted information such as the name of the VM. Since the command and the key $s$ are encrypted, the details will not be disclosed. In addition, the command is essentially a JSON document inside of the request, and hence it is flexible with various parameters. Thus, a new request is sent to the introspection application on the corresponding compute node.

Finally, the query handler sends the encrypted result data back to the client.

### 3.4.3 Introspection Application

The introspection application manages users' private keys, and hence it decrypts the key $s$ with the the specific user's private key $d$. Thus, it is able to decrypt the command message. The introspection application translates the message and invokes corresponding VMI tools built on top of the VMI library to acquire desired results.

Results are then encrypted with the symmetric key $s$ and transferred back to the query handler.

## 4. IMPLEMENTATION

The following subsections show details in our implementation for both the simulation to a public cloud and CryptVMI.

## 4.1 Simulation to the Public Cloud

We set up our experiments with OpenStack to simulate a public cloud environment. We want to simulate a public cloud in real life, such as Amazon Web Services (AWS), which uses Xen hypervisor and runs VMs in the para-virtualization mode [3]. However, the VMs in our cloud system were running with full-virtualization enabled with the Xen hypervisor, in order to support unmodified OSes.

### 4.1.1 Cloud API

OpenStack is a popular solution for hosting private clouds developed by Rackspace and NASA [22]. It has various components addressing necessities of a cloud system. We deployed the components needed by the controller node and network node on the same machine, and made it our management node. Even though the compute node can be the same machine as the management node, in order to simulate a public cloud environment, we used a separate machine with the same hardware configuration as our controller node. If there are multiple compute nodes in this cloud system, each of them should have its own running copy of the introspection application.

The OpenStack APIs are also designed in a RESTful manner, and hence our CryptVMI applications can communicate with them easily using industry standard libraries.

### 4.1.2 Virtual Machine Monitor

We used the Xen hypervisor to simulate AWS [3]. but it has been shown that Xen is not as secure as KVM [25]. Moreover, KVM can be less dependent than Xen, because it is essentially a kernel module [18], while Xen needs to modify the OS kernel [5].

However, the VMI library that we utilized requires a patch to KVM [23], and hence it will make the VMM possibly unreliable. Therefore, we still chose to use Xen as our VMM.

## 4.2 CryptVMI

We implemented the three major CryptVMI components in Ruby for three reasons. The first reason is its natural affinity with JSON documents, because the dictionary data structure in Ruby can be easily converted into JSON with the Ruby gem `json`. The second reason is that it provides gems, namely the libraries for Ruby, to help us for faster development. We used Ruby Encrypt in our symmetric key encryption, which simplified the process and hence we could only keep a 32-byte random string as the key for the AES-CBC-256-bit encryption [12]. The third reason is that Ruby is an actively maintained language, therefore security issues can be fixed in time.

The following subsections discuss some other features in CryptVMI.

### 4.2.1 Virtual Machine Introspection Library

We built our VMI application on top of LibVMI, which inherits features from XenAccess. XenAccess is designed based on the concepts of virtual memory introspection and virtual disk monitoring. Consequently, its APIs allow applications built on top of it to access the memory and disk of a specific virtual system [24]. It does not require any modification to the Xen hypervisor, and hence it has small performance and reliability impact. LibVMI also provides functions for accessing CPU registers, pausing a VM and printing binary data [23].

Maitland introduces a lightweight VMI system designed specifically to detect malware through packer detection [6]. The system achieves good performance, but it does not allow user to customize their queries. Furthermore, it is limited to para-virtualization environments.

By building CryptVMI on top of LibVMI, we can have supports for both fully and para-virtualized environments, and we are able to write introspection tools for different payloads.

### 4.2.2 Locating Virtual Machines

OpenStack provides APIs for users' authentications and to locate management nodes, however it does not provide the method to find the name of a VM within a compute node's hypervisor by looking up its instance name revealed in the cloud service [22].

Thus, we implemented this feature in both the query handler on the management node and the introspection application on the compute node. The query handler checks with the OpenStack API to see if there are new instances initiated in a certain time interval threshold $\tau$ seconds. We used 30 seconds in our configuration, because we want to simulate a public cloud environment in the real world, which should process a large number of instance creation requests in a short time period. Since only the query handler makes this check to compute nodes where new instances are created, it does not affect the network throughput much. If there is a new instance, the handler locates the corresponding compute node and asks for a list of new VMs with names on that node. It then creates a mapping and stores it in the configuration file on the compute node. If multiple instances are initiated during $\tau$ seconds, it then checks the instance creation timestamp for each new instance to decide the correct order for the mappings, since part of the VM name within the hypervisor is given with an incremental number.

### 4.2.3 Query Log

The query handler maintains a log for incoming requests, which is a collection of tuples consist of the IP address of request's origination along with the request. This feature helps us report failures and recover the query when the system crashes.

# 5. EVALUATION

In our experiments, the user's client machine, the management node and the compute node have the same hardware configuration with an Intel Core i7 CPU at 3.40 GHz and 8GB Memory installed. The DomU in this experiment, namely the guest system, is allocated with 1 vCPU and 1GB memory.

Since we want to ignore the time consumed during the network transmission of data, we benchmarked the code used in CryptVMI for the encryption and decryption of data. Thus, the total time was the estimated time for CryptVMI to finish the VMI task without considering network transmissions, while the original time was the time consumed in finishing traditional VMI tasks.

Figure 2 shows the results of the average time by running 10 times VMI acquisition for the processes list from a DomU. The overhead introduced by is very little and barely noticed.

In Figure 3, we can see the results of dumping the whole 1GB memory from a DomU. The encryption and decryption roughly took the same time as dumping the memory, which was about 10 seconds. We label the total time for dumping

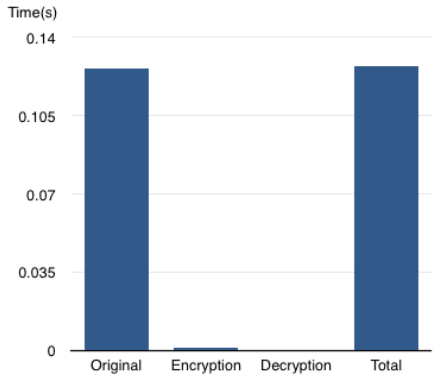


**Figure 2: Time distribution to acquire the processes list from a VM**

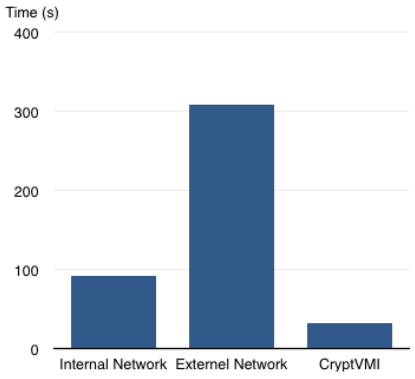


**Figure 4: Time consumed in data transmission in networks and in CryptVMI**

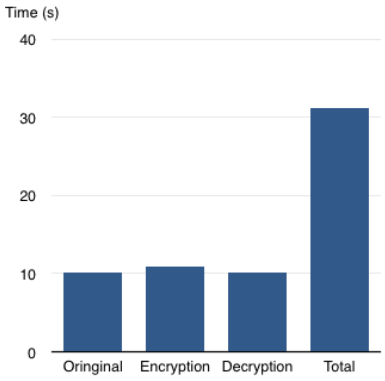


**Figure 3: Time distribution to dump 1GB memory from a VM**

the 1GB memory as the CryptVMI processing time in Figure 4. In this way, we can compare the CryptVMI dumping with encryption time with the time consumed in transferring the dumped data in the cloud system and between the remote client and the management node, respectively. The results shows that the CryptVMI processing time is only a small portion, compared with the time consumed during transmitting data through networks.

Though the overhead is not much in total, the main reason for this overhead can be concluded as the slow I/O operations on the disk. Our introspection application calls the VMI tools to dump the memory. The dump is written into a file by the VMI tools and then we read the file into memory for Base64 encoding and encryption. Once the data is sent to the user client application, the user client decrypts it and saves the encrypted dump into a temporary file. The client then decodes it for users and other applications .

## 6. CONCLUSION AND FUTURE WORK

An earlier version of this VMI system was presented in a work-in-progress paper [30], but we modified the design with new encryption and decryption schemes, and refined our interface for more flexibility. We also finished our implementation and analyzed the results from benchmarks on its performance for different payloads.

In this paper, we show our design and implementation of CryptVMI, an encrypted Virtual Machine Introspection system to provide flexibility in its interface to cooperate with other security monitoring systems while keeping the confidentiality of users' data, especially in the public cloud. Our benchmarks show that CryptVMI only introduces little overhead in performance. We believe that this solution is able to address the concerns of the multi-tenancy public cloud, while also providing an interface for enhanced security.

In the future, we plan to integrate this encryption feature into LibVMI by modifying the library, instead of relying on application level encryption schemes. This approach should also increase the performance of our system as dumps would be then encoded and encrypted in the memory. Eventually, we want to integrate with more security monitoring systems, such as IDSes, through our flexible RESTful APIs.

## 7. ACKNOWLEDGMENTS

This material is based on research sponsored by the Air Force Research Laboratory and the Air Force Office of Scientific Research under agreement number FA8750-11-2-0084, the Department of Energy under Award Number DE-OE0000097, and the National Science Foundation Graduate Research Fellowship Program under Grant Number DGE-1144245.

## 8. REFERENCES

[1] L. M. Adleman, R. L. Rivest, and A. Shamir. Cryptographic Communications System and Method. *U.S. Patent No. 4,405,829*, 1983.

[2] B. Amann, R. Sommer, S. Aashish, and S. Hall. A Lone Wolf No More: Supporting Network Intrusion Detection with Real-Time Intelligence. In *Proceedings of 15th International Conference on Research in Attacks, Intrusions, and Defenses*, 2012.

[3] Amazon Web Services. Amazon Web Services, [Online]. Available: http://aws.amazon.com/.

[4] Amazon Web Services. AWS Case Study: Netflix, [Online]. Available: http://goo.gl/jrVvI0.

[5] 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. *ACM Special Interest Group on Operating Systems Review*, 37(5), 2003.

[6] C. Benninger, S. W. Neville, Y. O. Yazir, C. Matthews, and Y. Coady. Maitland: Lighter-Weight VM Introspection to Support Cyber-Security in the Cloud. In *Proceedings of 5th IEEE International Conference on Cloud Computing*, 2012.

[7] C. Brenton. Introspection: Boon or Bane of Multitenant Security?, [Online]. Available: http://goo.gl/5aIj6W.

[8] T. Burger. The Advantages of Using Virtualization Technology in the Enterprise, [Online]. Available: http://goo.gl/2oqZgo.

[9] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy. Self-Service Cloud Computing. In *Proceedings of 19th ACM Conference on Computer and Communications Security*, 2012.

[10] D. Crockford. The Application/JSON Media Type for JavaScript Object Notation, [Online]. Available: http://tools.ietf.org/html/rfc4627.

[11] Dropbox. Where Does Dropbox Store Everyone's Data?, [Online]. Available: https://www.dropbox.com/help/7/en.

[12] J. Dwyer. Decrypting Ruby AES Encryption, [Online]. Available: http://goo.gl/THA4Sa.

[13] M. Factor, D. Hadas, A. Hamama, N. Har'el, E. K. Kolodner, A. Kurmus, A. Shulman-Peleg, and A. Sorniotti. Secure Logical Isolation for Multi-tenancy in Cloud Storage. In *Proceedings of 30th IEEE International Conference on Massive Storage Systems and Technology*, 2013.

[14] R. T. Fielding and R. N. Taylor. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2(2), 2002.

[15] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of 10th Annual Network and Distributed System Security Symposium*, 2003.

[16] S. Gueron. Intel Advanced Encryption Standard Instructions Set, [Online]. Available: hhttp://goo.gl/2Zp3OW.

[17] S. Josefsson. The Base16, Base32, and Base64 Data Encodings, [Online]. Available: https://tools.ietf.org/html/rfc4648.

[18] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: the Linux Virtual Machine Monitor. *Linux Symposium*, 1, 2007.

[19] J. Konstantas. VM Introspection: Know Your Virtual Environment Inside and Out, [Online]. Available: http://goo.gl/Yd4ioQ.

[20] M. Montanari, E. Chan, K. Larson, W. Yoo, and R. H. Campbell. Secure and Flexible Monitoring of Virtual Machines. In *Proceedings of 26th IFIP International Information Security Conference*, 2011.

[21] K. Nance, B. Hay, and M. Bishop. Virtual Machine Introspection: Observation or Interference? *IEEE Security and Privacy*, 2008.

[22] Open Stack. Open Source Software for Building Private and Public Clouds, [Online]. Available: https://www.openstack.org/.

[23] B. D. Payne. Simplifying Virtual Machine Introspection Using LibVMI. *Sandia Report*, 2012.

[24] B. D. Payne, M. D. P. de A. Carbone, and W. Lee. Secure and Flexible Monitoring of Virtual Machines. In *Proceedings of 23rd Annual Computer Security Applications Conference*, 2007.

[25] D. Perez-Botero, J. Szefer, and R. B. Lee. Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers. In *Proceedings of ACM International Workshop on Security in Cloud Computing*, 2013.

[26] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of 23rd ACM Symposium on Operating Systems Principles*, 2011.

[27] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *Proceedings of 16th ACM Conference on Computer and Communications Security*, 2009.

[28] M. Rouached and H. Sallay. RESTful Web Services for High Speed Intrusion Detection Systems. In *Proceedings of 20th IEEE International Conference on Web Services*, 2013.

[29] The Bro Project. The Bro Network Security Monitor, [Online]. Available: https://www.bro.org/.

[30] F. Yao and R. H. Campbell. CryptVMI: Encrypted Virtual Machine Introspection in the Cloud. Submitted for Review, 2014.