Auditing Cloud Service Level Agreement on VM CPU Speed

Ryan Houlihan, Xiaojiang Du, Chiu C. Tan, Jie Wu
Department of Computer and Information Sciences
Temple University
Philadelphia, PA 19122, USA
Email: {ryan.houlihan, dux, cctan, jiewu}@temple.edu

Mohsen Guizani
Qatar University
Doha, Qatar
mguizani@ieee.org

***Abstract*—In this paper, we present a novel scheme for auditing Service Level Agreement (SLA) in a semi-trusted or untrusted cloud. A SLA is a contract formed between a cloud service provider (CSP)and a user which specifies, in measurable terms, what resources a the CSP will provide the user. CSP's being profit based companies have incentive to cheat on the SLA. By providing a user with less resources than specified in the SLA the CSP can support more users on the same hardware and increase their profits. As the monitoring and verification of the SLA is typically performed on the cloud system itself it is straightforward for the CSP to lie on reports and hide their intentional breach of the SLA. To prevent such cheating we introduce a framework which makes use of a third party auditor (TPA). In this paper we are interested in CPU cheating only. To detect CPU cheating, we develop an algorithm which makes use of a commonly used CPU intensive calculation, transpose matrix multiplication, to randomly detect cheating by a CSP. Using real experiments we show that our algorithm can detect CPU cheating quite effectively even if the extent of the cheating is fairly small.**

Keywords*—*Cloud computing; Service Level Agreement; auditing; CPU

I. Introduction

Over the years, cloud computing has steadily gained popularity in both industrial and academic settings. Cloud computing is a model which allows for ubiquitous and on demand network access to a shared pool of configurable computing resources which are capable of being rapidly provisioned and released [1]. The cloud maximizes its efficiency using shared resources and rapid elasticity to achieve both coherence and economies of scale. Using cloud services, users can find many economic benefits by avoiding upfront infrastructure costs, maintenance costs and operational expenditures. Cloud computing systems also significantly reduce unnecessary overhead by both providing enterprises with faster deployment and improved manageability by a reduction in maintenance demands. Recently, a number of large cloud providers have begun pay-as-you-go services. Some of these are Amazon [2], IBM [2], Google [3] and Mega [4]. With the emergence of such large and effective cloud providers, an ever increasing number of enterprises and individual users have been migrating their data and computational tasks to cloud systems.

By definition, offered cloud services belong to one of three models. These models include infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS). IaaS is any cloud system that provides provision processing, storage, networks and other fundamental computing resource. PaaS is any cloud system that deploys consumer-created or acquired applications. SaaS is any cloud systems that provides applications [1]. Cloud service providers (CSPs) offer their services to clients in a pay-as-you-go fashion. The actual services the CSPs are required to provide are defined in the Service Level Agreement (SLA) which is a contract between the client and the CSP. For this work, we are most concerned with providers of IaaS services such as Amazons Elastic Compute Cloud (EC2) [2] that provides basic components such as memory, disk drives, and CPUs. For IaaS based services, the SLA metrics include CPU speed, storage size, network bandwidth, etc.

In cloud computing, a SLA serves as the basis for the expected level of service the CSP is required to provide. Being that a CSP is a profit driven enterprise, there is a great incentive for the CSP to cheat on the SLA. Due to this incentive to cheat, a CSP can not guarantee to audit the SLA and to verify for sure that it is being met. To handle this problem Amazon EC2, for example, has now moved the burden of auditing the SLA to the user. Unfortunately, the overhead for individual users to audit the cloud by themselves is high, since any auditing process will consume resources which the user has paid for. Thus, the only reasonable choice is to put the burden of auditing the SLA onto a third party whose purpose is to verify that the SLA is being met. This, however, is also problematic being that the CSP has incentives to defeat the SLA monitoring and verification techniques performed by the third party by interfering with the monitoring and measurement process.

In this paper, we present an algorithm for auditing CPU allocation and verify the corresponding SLA is being met via a SLA verification framework which makes use of a third party auditor (TPA). The TPA framework, first introduced by [5], [6], is highly beneficial for three fundamental reasons. First, it is highly flexible and scalable and can easily be extended to cover a variety of metrics from memory allocation to CPU usage. Secondly, it supports testing for multiple users which increases the accuracy of the cloud testing. Third, it removes the auditing and verification burden from the user and instead puts it on the TPA. Using the TPA, we can either prove that the CSP satisfies the SLA or detect and report an SLA violation. In this paper, our contribution can be summarized as follows:

- Develop a novel algorithm for auditing CPU allocation using a TPA framework to verify the SLA is met.
- Use real experiments to demonstrate the effectiveness

of our algorithm for detecting CSP cheating on the SLA metric of CPU speed.

II. Related Work

Brandic et al. [7] proposes a layered cloud architecture to model the bottom-up propagation of failures and uses these to detect SLA violations via mapping of resource metrics to SLA parameters. There have also been a variety of approaches for SLA assessment which focus on measuring or estimating Quality of Service (QoS) parameters. Sommers et al. [8] proposes a passive traffic analysis method for online SLA assessments which reduce the need for measuring QoS metrics. Wang and Eugene [9] present a quantitative study of end-to-end network performance among Amazon EC2 and conclude that virtulization causes significant unstable throughput and abnormal variations in delay. Li et al [10] compares the performance cost of four major cloud providers including Amazon, Microsoft, Google and Rackspace.

Goldburg et al. built off of Li's study by considering the previous work on an untrusted cloud which can interfere with the measurement and monitoring process which is triggered by users. Zhang et al. [5] and Ye et al. [6] propose a flexible and scalable framework which uses a TPA for SLA verification. This framework supports various types of SLA tests. In particular, they also develop an effective testing algorithm that can detect SLA violations on physical memory size of a VM.

III. Assumptions and A Threat Model

A. Assumptions

In our paper we make the following assumptions:

- The TPA can be trusted by the user to properly carry out the auditing functions while auditing the CSP and verifying the SLA.
- The CSP must provide the hypervisor source code to the TPA to ensure that it does not exhibit malicious behavior.
- The TPA must be able to ensure the integrity of the hypervisor. This is provided by Trusted Platform Group (TCG)'s Trusted Platform Module (TPM) and Core Root of Trust for Measurement (CRTM) [13]. The framework for ensuring hypervisor integrity is provided by Hypersentry [11].
- Communication time between the cloud system and the TPA is 200 ms or less.

B. Threat Model

Our threat model consists fundamentally of the fact that the CSP has complete control over all its own resources which include physical machines, VMs, hypervisor, etc. The CSP is able to access any data held on the VM and know about anything executed on the VM. The CSP can also modify any data held on the VM or any output of any execution. For example if a test is run and outputs a variety of timestamps, the CSP could stealthy change the timestamp values. Thus, the output data saved on the cloud system is not to be trusted. Finally, the CSP will only perform cheating if the benefit is greater than the cost. The cost may be too large for the CSP and thus the CSP would have no incentive to cheat.

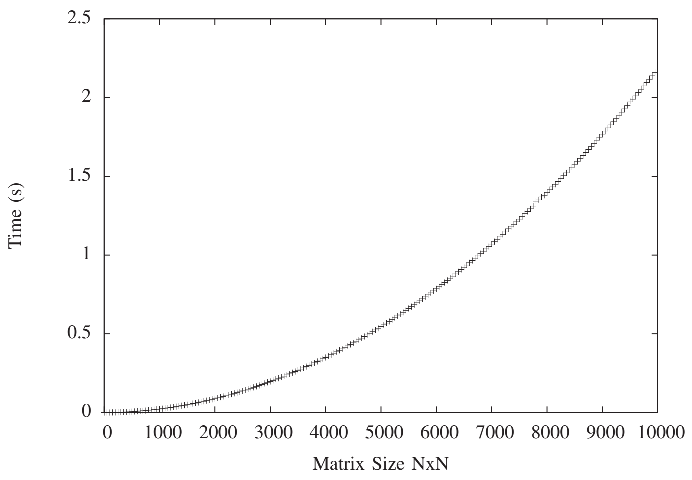

Fig. 1. Time it takes to compute a SHA-1 [15] of a NxN matrix of doubles. As is shown the time to compute the SHA-1 hash [15] is relatively small being only 0.022021013 s for a 1000x1000 matrix. This is 0.3% the time it takes to compute a transpose matrix multiplication of a 1000x1000 matrix.

IV. Implementation and Evaluation

A. Requirements

A variety of requirements must be fulfilled for our auditing test to be effective in preventing SLA cheating. First, our auditing test must run a fairly generic computational task so as to not be easily detected as an audit while also being computationally heavy and not wasting time with memory passing. To fill this first requirement, we choose to use a transpose matrix multiplication. This was chosen over a standard matrix multiplication as transpose multiply minimizes memory passing while maximizing CPU time.

Our auditing test must also be able to detect if the cloud system has modified the input or output of our auditing test. It must then report this malicious behavior. We accomplish this through redundant time recording performed by the TPA which will be covered in depth further on. Next, our auditing test should be sensitive enough to detect a wide variety of cheating behaviors, reporting as low as 2% cheating to be unacceptable.

Next, our auditing test should be developed in such a way that its accuracy does not depend on any of the cloud systems timing functions but instead depends only on the accuracy of the timing functions on the TPA's system. This prevents the CSP from reporting false times and hiding cheating. The communication overhead of 200 ms must also be less than 1% the total execution time of a single cycle of our execution. Finally, we must be able to assure the computational task has actually been run, not just faked. We do this by taking a SHA-1 hash [15] of the resulting matrix.

The time to compute the SHA-1 hash [15] of a NxN matrix is relatively small compared to the computation time of a transpose matrix multiplication. This is shown to be true in (*Fig. III-A*) where it is clear that the time to run the SHA-1 hash on a NxN matrix of doubles does not increase significantly with larger sizes of N compared to the increased computational time of the transpose matrix multiplication of the larger matrices. As an example it takes 0.022021013 s to compute the SHA-1 hash [15] of a 1000x1000 matrix while it takes on average 6.7361414708 s to compute the transpose matrix multiplication of two 1000x1000 matrices on the same system. This means that the SHA-1 hash [15] only takes about 0.3 % of the time it takes to compute the matrix itself.

V. Implementation and Algorithm

Our implementation consists of three distinct parts. Initialization, algorithm execution, and verification.

A. Initilization

The initialization consists of the following parts:

- VM mirroring
- NxN matrix creation and upload

The VM mirroring is done by the TPA. The TPA must first rent a VM on the target cloud system. The TPA must then, on their auditing system, create a VM which mirrors the specifications of the VM they have rented from the cloud provider. This VM's must also have the same or similar background tasks. Next, the TPA must create two NxN matrices where N is a number such that when the two NxN matrices are multiplied with each other the time, τ to perform the multiplication is large enough that the communication overhead is less than 1% of τ. Finally, once the NxN matrices are created they must be loaded onto the target cloud systems VM.

B. Auditing Test Execution

To introduce the next stage in our implementation, we must first introduce the auditing test itself. Only three pieces of information are needed for the execution of our test. Two premade NxN matrices, A and B, and a number X where X is the number of transpose matrix multiplications to be run. The execution of the auditing test on the cloud VM is as follows:

- Output signal to terminal that multiplication will begin and record the time, t_{2-i}.
- Perform a matrix multiplication where $C = A \times B$.
- Record the elapsed time, e_{2-i}, and output to the terminal that the multiplication has ended.
- Compute the SHA-1 hash of the resulting matrix C, represented as SHA-1[C].
- Output the time to compute the matrix multiplication, e_{2-i} and SHA-1[C] to the terminal.
- Shift each element of matrix A and B by one.
- Repeat the previous steps $X - 1$ more times where i is equal to the current transpose matrix multiplication being performed.

The execution of the algorithm on the TPA VM is as follows:

- Record the time, T.
- Initialize and execute the auditing test on the cloud VM.
- Watch the output from the cloud VM terminal. Compute the time elapsed between the signal that the multiplication has started and the signal that the multiplication has ended, $e1_i$. Also record the hash value, SHA-1[C], and the execution time, e_{2-i} as reported by the cloud VM.
- Record the elapsed time, E, for the entire execution of the test.

First, the TPA begins the execution of the auditing test on the cloud VM and records the time, T, the test begins. The auditing test then initializes itself and signals via the terminal that it will compute a transpose matrix multiplication. This signals the TPA to take a recording of the time t_{1-i} where i is initialized to one. The auditing test also takes its own recording of the time t_{2-i}. Two matrices, A and B, are then multiplied as $C = A \times B$. The time is then again recorded by the auditing test and the time elapsed since t_{2-i} is recorded as e_{2-i}. The auditing test then signals the terminal that the multiplication has ended. The TPA receives this signal and records the time elapsed since t_{1-i} as e_{1-i}. Next, the auditing test computes the SHA-1 hash [15] of the resulting matrix, C, as *SHA-1[C]*. The elapsed time e_{2-i} and *SHA-1[C]* are then both output to the terminal. The TPA then records these values.

Next, all the values in A and B are then shifted to the right by one place. The counter i is increased by one. The multiplication process as listed above is then repeated until $i = X$ for a total of $X - 1$ times. Finally, the execution of the auditing test ends and the TPA records the time elapsed since T as E. While this is all going on the TPA runs, on its VM, the same algorithm records the time for each transpose matrix multiplication test as well as the hash of the result for validation purposes.

C. Verification

Now the question remains: Given all this information we have recorded, how do we detect a breach in the SLA by the CSP? The first step we take is to sum up the e_{2-i} value, $\sum_i^X e_{2-i}$ for all tests run on the cloud VM. We compare this to the value of E. Since the communication overhead, SHA-1 hashing, and time to shift the matrix values are very small compared to the execution time of the algorithm we can be assured that $\sum_i^X e_{2-i}$ should be close to but less than the value of E. If the CSP has blatantly cheated then $\sum_i^X e_2$ will be greater than, or significantly less than, the value of E and a violation of the SLA can be detected.

Next, we take $\sum_i^X e_{1-i}$ and compare it to $\sum_i^X e_{2-i}$. Again since the communication time is small compared to the actual transpose matrix multiplication, $\sum_i^X e_1$ should be very close, $<< 1\%$, to $\sum_i^X e_2$. If it is not, then a violation of the SLA is once again is detected..

Furthermore, the hash of the resulting matrix C, written as *SHA-1[C_i]*, from the X tests run on the TPA's VM should match the *SHA-1[C_i]* values produced on the clouds. This prevents the cloud from avoiding doing an actual computation or delaying computations to a later time.. Since it is impossible to compute the correct *SHA-1[C_i]* values without computing C itself, the CSP can not avoid carrying out the computations. Also since the hash is output and recorded via the terminal by the TPA, each transpose matrix multiplication in the cloud has no way of delaying the computation and later modifying

TABLE I. EXECUTIONS OF THE AUDITING TEST.

Average Time 100% CPU:	6.7361414708				
	AVERAGE (s)	**STDEV (s)**	**STDEV % DIFF**	**TTL EXECUTION**	**% DIFF AV**
100% CPU (Run 1):	6.727433531	0.0064725957	0.10%	112 min 49.320 s	0.13%
100% CPU (Run 2):	6.7399728319	0.0179976439	0.27%	113 min 2.049 s	0.06%
100% CPU (Run 3):	6.7398816269	0.0169161707	0.25%	113 min 2.049 s	0.06%
100% CPU (Run 4):	6.7372778932	0.0169161707	0.25%	112 min 59.325 s	0.02%
90% CPU:	7.5026936102	0.0360844519	0.48%	125 min 50.123 s	11.38%
80% CPU:	8.4290672141	0.0306025842	0.36%	141 min 21.955 s	25.13%
70% CPU:	9.628378256	0.0271392312	0.28%	161 min 28.456 s	43.12%
60% CPU:	11.242350495	0.0325625398	0.29%	188 min 31.927 s	67.11%
85% CPU 15% TTL:	6.9142334212	0.645254425	9.33%	115 min 57.541 s	2.78%
85% CPU 30% TTL:	7.0991599864	0.8731635046	12.30%	119 min 3.476 s	5.53%
70% CPU 15% TTL:	7.165942121	1.0357176467	14.45%	119 min 31.927 s	6.52%
70% CPU 30% TTL:	7.6036018606	1.3313916735	17.51%	127 min 31.176 s	13.02%

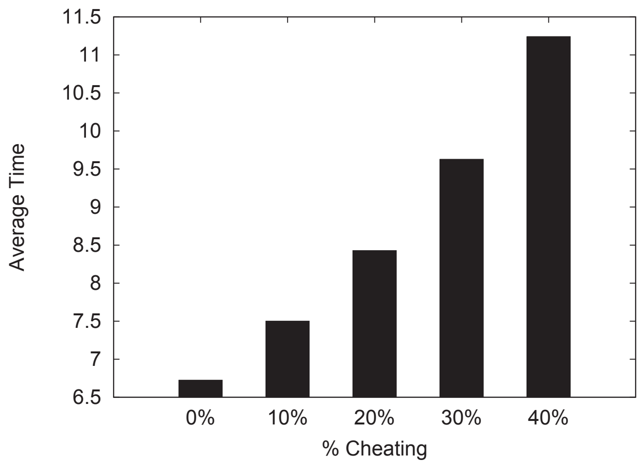

Fig. 2. The average time to run a single transpose matrix multiplication based on the percent cheating (100%-CPU Cap %). As the % cheating increases the average run time increases linerealy, as expected.

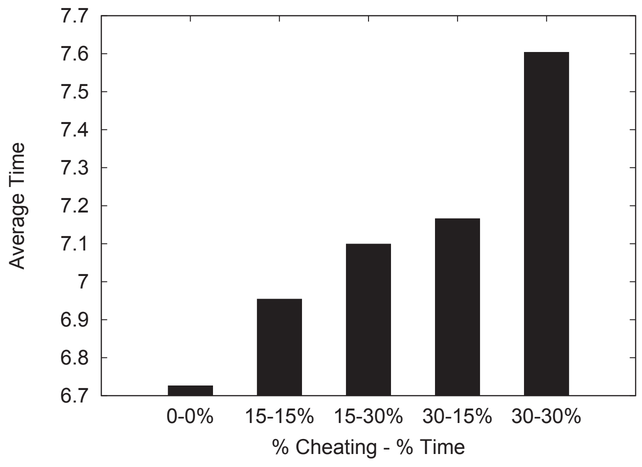

Fig. 3. The average time to run a single transpose matrix multiplication based on the percent cheating (100%-CPU Cap %) and the % time the cheating lasts. As the % cheating or the % time of cheating increases the average run time increases as expected.

the output. If the hashes do not match the TPA VM and the CSP VM, then we have once again detected a violation of the SLA.

VI. TESTING

A. Background

To test our algorithms' ability to detect SLA violations by the CSP, we ran a variety of different tests. Initially, we ran four tests where no cheating has occurred to find a base execution time. We then ran one test each where the CSP limits the CPU percentage to 90%, 80%, 70%, and 60% of the expected CPU, respectively. We also ran tests where the CSP limits the CPU to 85% and 70% of its required value for 15% of the algorithms execution and 30% of the algorithms execution.15% and 30% of the algorithms' execution, respectively. To perform this testing, we used Ubuntu Server 12.04 LST with Xen DOM-0 Hypervisor 4.1 x64 [12]. For our SHA-1 hashing [15] algorithm we used PolarSSL's [14] library. The tests were run on a system with 4 Gigs of ram and a Intel Q6600 Quad Core processor. The VM used was given one processor with a clock of 1.0 Ghz as well as 1 Gigabyte of RAM.

To create a CPU cap on our VM, we used Xen's sched-credit function. The sched-credit function allows us to specify a CPU cap in percentage. This is done by the command

```
xm sched-credit -d <domain> -c <cap>
```

where *<domain>* is the name of the VM in question and *<cap>* is the cap we would like to apply to the CPU in terms of percentage.

B. Results

The results of our test are shown in *Table I* . For all these runs, we used a 1000x1000 matrix of doubles. In *Table I* the first column lists the average, $\frac{\sum_i^X e_{2-i}}{X}$ for each individual auditing test. The second lists the standard deviation, σ of each $\sum_i^X e_{2-i}$. The third lists the percent difference between the average and the standard deviation. The fourth lists the recorded time to execute the entire auditing test. Finally, the last column lists the percent difference of the $\frac{\sum_i^X e_{2-i}}{X}$ with the average of $\frac{\sum_i^X e_{2-i}}{X}$ for each of the four runs where there was no cheating.

For the first four runs we did not set a cap on the CPU and used 100% of the allocated CPU. As we can clearly see that σ for all four of the runs using 100% of the CPU is very small and the corresponding percent difference is also small as expected. The total run times of each of the auditing test were also fairly consistent. Lastly the percent difference from the average of the four 100% CPU runs is also small. The largest percent difference is only 0.13% where the smallest is 0.02%.

Next, if we look at the 90% CPU run we can see that σ and the percent difference are both also small as expected.. We notice though that the total run time of the auditing test has increased significantly. We also notice that the percent difference between the 90% CPU test and the average of the four 100% CPU runs has increased significantly to 11.38%. Thus in our analysis, it is very obvious that the CSP has violated the SLA by putting a cap on the total CPU percentage we can use.

For a CPU cap of 80%, 70% and 60%, we notice similar results. The average, $\frac{\sum_i^X e_2}{X}$, for each run steadily increases. The corresponding percent difference from the average of the four 100% runs also increases significantly. When the CSP has a CPU cap of 60% we see a percent difference of 67.11% from what we would expect. Thus, from these results it is fairly safe to say that if a CSP puts an unchanging cap on the CPU, the proposed auditing test will easily detect this cap and report it as a violation of the SLA.

We also performed an analysis on cheating of a different type. A cloud provider, rather than putting a single unchanging cap on the CPU might instead cheat only a percentage of the time. We replicated such an event by capping the CPU at 85% and 70% for 15% and 30% of the execution time. For the remaining 85% and 70% of the respective executions, the cap was removed.

First, looking at the two 85% CPU cap runs we notice a significant increase in σ and the percent difference. This inconsistency in run time of each individual transpose matrix multiplication is the first obvious sign of a malicious activity. For the run with a cap only used for 15% of the total execution, a percent difference of 9.33% was found. For the run with a cap used for 30% of the total execution a percent difference of 12.30% was found. Furthermore, the percent difference from the average also shows clear cheating by the CSP. Overall, for both 85% CPU cap runs violations of the SLA by the CSP are very obvious.

Similarly for the two 75% CPU cap runs, there were also large inconsistencies between the run times of each individual transpose matrix multiplication. For the first run with a cap for 15% of the execution time, the percent difference between σ and the average has increased noticeably to 14.45%. For the second run with a cap for 30% of the execution time, this percent difference increases even further to 17.51 %. Again this is a sign of malicious activity by the CSP. Finally, the percent differences from the average expected time are 6.52% and 13.02% which are clear signs of a SLA violation by the CSP.

VII. Conclusion

Due to an increased interest in cloud computing, providing accountability to clients has become a critical component of the value proposition offered by cloud providers. Service Level Agreements (SLA) define the agreement between cloud service providers (CSPs) and their users. Being that CSP's are profit based companies, it is in the CSP's best interest to cheat on the SLA. To alleviate this problem we make use of a Third Party Auditor (TPA) to audit the SLA and verify it is being met by the CSP. In this paper, we develop a scheme which makes use of a TPA to audit the SLA metric of CPU speed and verify it is being met by the CSP. Overall, our auditing scheme shows promising results and is able to detect even minor cheating by the CSP on the SLA regardless of the CSPs attempts to hide its cheating.

Acknowledgment

This work was supported in part by the US National Science Foundation under grants CNS-0963578, CNS-1022552, and CNS-1065444.

References

[1] "The NIST Definition of Cloud Computing". National Institute of Standards and Technology. Retrieved 2013-8-5

[2] Amazon EC2, [Online]. Available: http://aws.amazon.com/ec2

[3] Google App Engine, [Online]. Available: http://www.google.com/enterprise/appengine

[4] Mega, [Online]. Available: http://mega.co.nz

[5] H. Zhang, L. Ye, J. Shi, X. Du. "Verifing Cloud Service-Level Agreement By a Third-Party Auditor," *Security and Communication Networks*, 2013.

[6] L. Ye, H. Zhang, J. Shi, X. Du. "Verifying Cloud Service Level Agreement," *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, pp. 777-782, 2012.

[7] I. Brandic, V. C. Emeakaroha, M. Maurer, S. Dustdar, S. Acs, A. Kertes, and G. Kecskemeti, *Proceedings of 34th Annual IEEE Computer Software and Applications Conference* Workshops, pp. 366-370, 2010.

[8] J. Sommers, P. Barford, N. Duffield, and A. Ron, *IEEE/ACM Transactions on Networking*, vol. 18, issue. 2, IEEE Press: NY, USA, pp. 652-665, 2010.

[9] G. Wang and N. T. Eugene, *Proceedings of the 29th IEEE Conference on Computer Communications*, pp. 1163-1171, 2010.

[10] A. Li, X. Yang, S. Kandula, and M. Zang, Proceedings of the 10th Internet Measurement Conference, ACM: New York, NY, USA, pp. 1-14, 2010.

[11] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, N. C. Skalsky. "HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity." *Proc. of the 17th ACM Conference on Computer and Communications Security*, pp. 38-49, 2010.

[12] P. Barham, B. Dargovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield. Xen and the Art of Virtualization. *Proc. 19th ACM Symposium on Operating Systems Principles*, SOSP 2003, Bolton Landing, USA, October 2003.

[13] Trusted Computing Group. TPM specifications version 1.2. https://www.trustedcomputinggroup.org/downloads/specifications/tpm, July 2005.

[14] PolarSSL. Offspark, 2011. Avaliable: http://polarssl.org/source_code

[15] Department of Commerce National Institute of Standards and Technology. *Secure Hash Signature Standard (SHS) (FIPS PUB 180-2)*. February 2004