Detecting VMs Co-residency in the Cloud: Using Cache-based Side Channel Attacks

Yu Si[1,2], Gui Xiaolin[1,2], Lin Jiancai[1,2], Zhang Xuejun[1,3], Wang Junfei[1,2]
[1]*Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, P.R.China*
[2]*The Key Laboratory of Computer Network, Xi'an Jiaotong University, Xi'an 710049, P.R.China*
[3]*School of Electronic and Information Engineering, Lanzhou JiaoTong University, Lanzhou 730070, P.R.China*
yusi.xjtu@gmail.com

***Abstract*—Virtual machine technology enables the cloud to offer large scale and flexible computing ability. However, it also introduces a range of new vulnerabilities. Malicious users can extract sensitive information from other users covertly via side channel attacks, which breaks the isolation between the virtual machines (VMs). In this paper, we investigate such a security threat and propose the VMs Co-residency Detection Scheme via cache-based side channel attacks (VCDS) to get the location of the specified VM. Using load pre-processor based on cubic spline interpolation, VCDS makes the raw measurements more smoothing and relevant. With the load predictor based on linear regression model, VCDS probes cache load changes produced by the victim VM more accurately. Based on the normal cloud model, VCDS computes the co-residency probability to describe VMs co-residency quantitatively. The experimental results show that VCDS improves the true detection rate even with the interference of the co-resident noisy VM compared to the existing schemes.**

***Index Terms*—Cloud computing, virtual machine, side channel attack, co-residency detection.**

I. INTRODUCTION

Cloud computing, an emerging computing and service paradigm, in which the core computing and software capabilities are outsourced on demand, offers the prospect of lower costs, lighter administrative burdens and more convenient user experiences. However, it also introduces new security challenges, such as data privacy [1] and VM (Virtual Machine) security [2], which severely hinder the development of the cloud computing.

Based on the logical isolation between VMs, access control mechanisms are proposed to guarantee security of the cloud. However, information leak persists. It is mainly introduced by the vulnerabilities caused by the sharing of hardware resources, e.g. CPU cache, main memory, network traffic, etc., which lead to side channel attacks [3]. Malicious users can extract private information from other co-resident VMs covertly by analyzing the responses of sharing resources, such as computing time, power consumption, etc.

The focus of this paper is to reveal the threats introduced by cache-based side channel attacks in the cloud. VMs co-residency detection via side channel attacks aims to get the location of the victim VM by analyzing the responses of the shared cache. The attack is harmful to both the service providers and common users of cloud computing. To the service providers, adversary can undermine the location-dependent features of the cloud, due to the leakage of location. For example, how many VMs exist in the cloud and where the VMs are applied, etc. To the common users, disclosure of VMs location means that the isolation between VMs is broken. Therefore, on the one hand, adversary may launch more specific side channel attacks to steal private information. On the other hand, victim VMs are more likely to suffer from the intrusion attacks, such as DDOS.

Side channel attacks in cloud have been investigated recently [4], [5]. However, to the best of our knowledge, they are still limited in the following two aspects. *First*, they pay less attention to the interferences introduced by the VMs which reside in the same host with the malicious VM, which has impact on the success rate of cache-based side channel attacks. *Second*, the noises introduced by the hardware features and software features, such as TLB (Translation Lookaside Buffers) misses, hardware prefetching, VMM (Virtual Machine Monitor) scheduling, etc., are ignored by the existing work as well, which make the raw measurements are extremely variable.

To overcome these limitations, a novel *VMs Co-residency detection scheme via cache-based side channel attacks* (VCDS) is presented. In VCDS, load preprocessor based on cubic spline interpolation and load predictor based on linear regression are proposed to process the raw measurements; co-residency probability computing algorithm is designed to calculate the probability of VMs co-residency; detection rules are put forward to determine the results. The experimental

Manuscript received September 09, 2012; accepted February 23, 2013.
This research was supported in part by NSFC under Grant 60873071, 61172090 and 61103231, National Science and Technology Major Project under Grant 2012ZX03002001-004, Scientific and Technological Project in Shaanxi Province under Grant 2012K06-30, Natural Science Basic Research Plan in Shaanxi Province of China under Grant 2011JM8012, Ph.D. Programs Foundation of Ministry of Education of China under grant 20120201110013.

results show that our proposed scheme is feasible and effective.

II. CACHE-BASED SIDE CHANNEL ATTACKS IN CLOUD

In the multi-core CPU architecture, data cache, used to improve the computing speed, is shared by different cores. So, data cache is shared by the VMs residing in the same physical machine (same with host in the following). Moreover, administrator privilege is not needed to change cache load. Therefore, data cache can be used to construct a side channel.

Before detailing the cache-based side channel attacks in the cloud, we divide VMs into three roles in the attack scenario. They are malicious VM, victim VM and noisy VM, respectively. Malicious VM belongs to the attacker, which probes and analyzes cache activities. Victim VM is the attack target. Noisy VM resides in the same host with the malicious VM, which interferes with the attack.

Commonly, cache-based side channel attacks consist of two major steps: *(1)* measuring the cache load and *(2)* analyzing the cache load to extract the private information.

The key issue of the first step is to exploit the timing differences in access latencies between cache and main memory. The load measurement generation method is the *Prime-Probe* [4] including the following steps:

1) Prime: Malicious VM allocates a contiguous buffer of b bytes and fills the entire cache by reading the buffer at s-byte offsets. Here, b should be larger than cache size; s is the cache line size;
2) Wait: Malicious VM waits for a pre-specified time interval. In this interval, cache is used by other co-resident VMs, hopefully in favor of the victim VM;
3) Probe: Malicious VM reads the buffer at s-bytes offsets again and measures the time.

When reading the buffer, *pointer-chasing* technique [6], [7] is used to mitigate the affections of hiding of access latencies incurred by hardware pre-fetcher.

The time of the final step's read is the load measurement, measured in number of CPU cycles. The load measurement will be strongly correlated with the use of cache during the *Wait* step, since that usage of cache evicts some portion of the buffer and thereby drives up the read time during the *Probe* phase.

If there is much cache activity from victim VM during the probing interval, malicious VM's data is likely to be evicted from the cache and replaced with data accessed by victim VM, which results in a noticeably higher timing measurement. Therefore, high timing measurement implies that cache is heavy load and victim VM is busy; while the low one implies that cache is light load and victim VM is idle.

In this paper, we assume that victim VM provides public services, such that any user can access the services. We believe that such an assumption is feasible, since the malicious users are always interested in the public service providers. Therefore, malicious VM can burden victim VM's computational load by accessing the public services. After the computation produced by accessing the service, cache load will decrease. Malicious user can get three cache load measurements sets which are sampled before, during and after accessing the victim VM's service, respectively. Ultimately, malicious user can get the private information by analyzing differences of the three load sets. Consequently, side channel attacks in cloud computing can be converted to the matter of computing the similarity of cache load sets.

III. VMS CO-RESIDENCY DETECTION SCHEME

In this section, VCDS is detailed, which takes as its input three cache load measurements sets. The first set is obtained before accessing the services of victim VM, marked as *First cache load set*, short for $Fcl=(fcl_1,fcl_2,\ldots,fcl_n)$. The second set is probed when victim VM is doing the computation produced by accessing the service, marked as *Second cache load set*, short for $Scl=(scl_1, scl_2,\ldots,scl_n)$. The third set is probed after the end of the service computation, marked as *Third cache load set*, short for $Tcl=(tcl_1, tcl_2, \ldots, tcl_n)$.

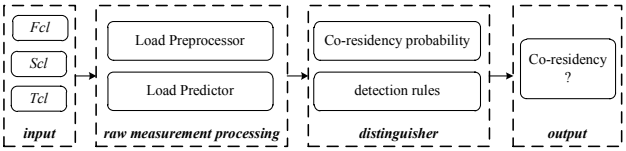

Fig. 1. Illustrated process of co-residency detection scheme.

VCDS consists of two major modules which are *raw measurement processing* and *distinguisher*. The processes are illustrated in Fig. 1. The major steps include:

Step 1. Load preprocessor takes as its inputs *Fcl*, *Scl* and *Tcl*, and returns Fcl^1, Scl^1 and Tcl^1 which respectively represent the load trend of *Fcl*, *Scl* and *Tcl*.

Step 2. Load predictor takes as its inputs Fcl^1 and Scl^1 respectively, and returns Fcl^2 and Scl_1^2, which are the estimated caches load in the near future for Fcl^1 and Scl^1. Meanwhile, choosing specified elements from Scl^1 and Tcl^1 to construct Scl_2^2 and Tcl^2, where the timestamps of Scl_2^2 and Tcl^2 are respectively same with the timestamps of Fcl^2 and Scl_1^2.

Step 3. Obtain Fcl^3, Scl_1^3, Scl_2^3 and Tcl^3 according to Fcl^2, Scl_1^2, Scl_2^2 and Tcl^2, based on the normal cloud model [8]. Fcl^3 includes twofold information of the load value and its membership degree. And the same to Scl_1^3, Scl_2^3 and Tcl^3.

Step 4. Distinguisher takes as its inputs Fcl^3, Scl_1^3, Scl_2^3 and Tcl^3, and returns the result based on the co-residency probability computing algorithm and the detection rules.

A. Load pre-processor based on cubic spline interpolation

Load preprocessor is used to filter out the noises and yield a more regular view about the load trend. The necessity of load preprocessor is twofold. First, the raw measurements, which are variable due to the noises existing in the system, should get smoothing. Second, the correlation between the raw measurements should be strengthened, which leads to a more accurate prediction value.

Cubic Spline Interpolation (*CSI*) is considered to be a representative example of a non-linear numerical data analysis tool. Lower-order curves (with a degree less than 3) do not react quickly to load changes, while high-order curves

(with a degree higher than 3) are considered unnecessary complex, introduce undesired wiggles and are computationally too expensive.

For the definition of the cubic spline interpolation, some control points (t_j, cl_j) in the set of measured cache load should be chose firstly, where t_j is the time of measurement of cl_j. A cubic spline interpolation function $CS^J(t)$, based on J control points, is a set of J-1 piecewise third-order polynomials $p_j(t)$, where $j \in [1, J\text{-}1]$, with the following form

$$CS^J(cl) = \begin{cases} p_1(cl), & cl_1 \le cl < cl_2, \\ p_2(cl), & cl_2 \le cl < cl_3, \\ \ldots \\ p_{J-1}(cl), & cl_{J-1} \le cl < cl_J, \end{cases} \quad (1)$$

where p_i (i=1,2,…,J-1) is a third-order polynomial defined in (2)

$$p_i(cl) = a_i(cl - cl_i)^3 + b_i(cl - cl_i)^2 + c_i(cl - cl_i) + d_i \quad (2)$$

and $CS^J(t)$ should satisfy the following two properties.

Property 1. The piecewise third-order polynomials $CS^J(t)$ will interpolate all data points.

Property 2. $CS^J(t)$, $CS^{J'}(t)$ and $CS^{J''}(t)$ will be continuous on the interval $[cl_i, cl_n]$.

If we apply both properties, we can obtain the solution of a_i, b_i, c_i and d_i as the following from:

$$\begin{cases} a_i = (x_{i+1} - x_i)/6h, \\ b_i = x_i/2, \\ c_i = (cl_{i+1} - cl_i)/h - (x_{i+1} + 2x_i)/6h, \\ d_i = cl_i, \end{cases} \quad (3)$$

where h=l_i-l_{i-1}, and x_i can be got via the matrix equation:

$$\begin{pmatrix} 1\,4\,1\,0 & \ldots & 0\,0\,0\,0 \\ 0\,1\,4\,1 & \ldots & 0\,0\,0\,0 \\ 0\,0\,1\,4 & \ldots & 0\,0\,0\,0 \\ \vdots & \ddots & \vdots \\ 0\,0\,0\,0 & \cdots & 4\,1\,0\,0 \\ 0\,0\,0\,0 & \cdots & 1\,4\,1\,0 \\ 0\,0\,0\,0 & \cdots & 0\,1\,4\,1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{J-3} \\ x_{J-2} \\ x_{J-1} \\ x_J \end{pmatrix} = \frac{6}{h^2} \begin{pmatrix} cl_1 - 2cl_2 + cl_3 \\ cl_2 - 2cl_3 + cl_4 \\ cl_3 - 2cl_4 + cl_5 \\ \vdots \\ cl_{J-4} - 2cl_{J-3} + cl_{J-2} \\ cl_{J-3} - 2cl_{J-2} + cl_{J-1} \\ cl_{J-2} - 2cl_{J-1} + cl_J \end{pmatrix}. \quad (4)$$

Note that this system has J-2 rows and J columns, and is therefore under-determined. With the *natural splines* which defines that x_1=x_J=0, we can obtain the deterministic solution of the matrix equation eventually.

The CSI-based load preprocessor returns a new value after n cache load measures. Moreover, the cubic spline has the advantage of being reactive to load changes and independent of cache load metrics and workload characteristic.

Ultimately, the set consisting of cache load trend values is obtained, which is defined as L= (l_1,l_2,…,l_k), whose corresponding timestamp set is T=(t_1,t_2, …,t_k).

B. Load predictors based on linear regression

Load predictor is used to estimate the cache load in the near future, which plays a significant role in our proposed scheme. Through rendering the predictor, the detection scheme responds to the sudden changes of the system running state more effectively. The key issue is to improve the prediction accuracy.

To improve the accuracy of load prediction, the correlation between raw measurements should be strengthened. Existing work [7] shows that the correlation coefficient will get larger after the process of *CSI*-based preprocessor. Therefore, the load predictor defined in the paper takes as its input a set of load trend values $L(t_i)$ and returns a load trend value at time t_{i+w}. Therefore, a predicted load value set $\tilde{L} = (\tilde{l}_1, \tilde{l}_2, ..., \tilde{l}_v)$ will be obtained, where v is the number of predicted values.

In this section, we consider the load predictor as a linear system, so that the predictor $LP_w(L(t_i))$ based on the linear regression is used. The predictor is characterized by a couple of parameters: the prediction window w, which represents the size of prediction interval, and the past time window q, which is the size of load trend value set. Then the first trend value l_{i-q} and the last l_i are considered. $LP_w(L(t_i))$ is the line that intersects the two points (l_{i-q}, t_{i-q}) and (l_i, t_i), and returns $\tilde{l}_{i+w}$ that is the predicted value at time t_{i+w}

$$LP_w(L(t_i)) = \gamma t_{i+w} + \lambda, \quad (5)$$

where $\gamma = (l_i - l_{i-q})/q$, and $\lambda = l_{i-q} - \gamma t_{i-q}$.

C. Determination of co-residency probability

In the ideal case, detection result is a definite concept which has only two values: yes and no. However, in the real environments, the detection result is fuzzy because of the influence of various factors. Therefore, in this section, based on the similarity degree of normal cloud model, we put forward co-residency probability to quantitatively characterize the possibility of co-residency. The similarity degree refers to the distance between two normal cloud models. The closer distance between the two normal cloud models, the larger membership degree of the cross points will be obtained. Therefore, we take the membership degree to quantify the similarity degree.

Definition 1. *Similarity degree* (μ_{sim}) between normal cloud model C^1 and C^2 is the maximum membership degree of the crossing points of the two normal cloud models, whose definition domain is [0, 1].

According to the 3En principle, the crossing points should fall in the interval [Ex_1-3En_1, Ex_1+3En_1] or [Ex_2-3En_2, Ex_2+3En_2]. Ex_1 and En_1 refers to the expectation and entropy of C^1, and Ex_2 and En_2 refers to the expectation and entropy of C^2. In the following two cases, the similarity degree is defined to be 0. *(1)* Both of the two crossing points are not in the intervals. *(2)* There are no crossing points between C^1 and C^2.

Definition 2. *Co-residency probability* (μ) is defined as μ=1-μ_{sim}, which decreases with the increase of similarity degree and is used to quantify the possibility that malicious VM and victim VM are co-resident.

The co-residency probability computing algorithm consists of the following steps:

Step 1. Use (6)-(8) to compute the estimated value of key parameters of normal cloud model [8], which are Ex, En and He respectively. Ex refers to the mathematical expectation of the cloud model. En refers to entropy of the cloud model, representing the uncertainty measurement of a qualitative concept. He refers to the hyper-entropy of cloud model, representing the uncertain degree of entropy:

$$Ex = \frac{1}{v}\sum_{1\le i\le v} l_i, \tag{6}$$

$$En = \sqrt{\frac{\pi}{2}}\frac{1}{n}\sum_{1\le i\le v}\left|l_i - \widehat{\mathrm{Ex}}\right|, \tag{7}$$

$$He = \sqrt{\left|\frac{1}{v-1}\sum_{1\le i\le v}(l_i - \widehat{Ex})^2 - \widehat{En}^2\right|}. \tag{8}$$

Step 2. Use *forward normal cloud generator* algorithm to obtain both C^1 and C^2 based on these estimators. Limited to the length of paper, we will not deepen further the algorithm here. Readers who are interested can read [8].

Step 3. Compute two crossing points: cp_1 and cp_2:

$$\begin{cases} cp_1 = \dfrac{Ex_2En_1 - Ex_1En_2}{En_1 - En_2}, \\ cp_2 = \dfrac{Ex_1En_2 + Ex_2En_1}{En_1 + En_2}. \end{cases} \tag{9}$$

Step 4. Compute the membership degree of cp_1 and cp_2 respectively

$$\mu_C(x) = e^{-\frac{(x-Ex)^2}{2(\alpha * En)^2}}. \tag{10}$$

Compared to the classical definition of membership degree function, we introduce a new ingredient in (10), which is denoted as α and makes the similarity computing dynamically adapts to different systems built on different hardware platforms.

Step 5. Compute the similarity degree μ_{sim} of by computing the higher value of $\mu_{C^1}(cp_1)$ and $\mu_{C^1}(cp_2)$, which is defined in (11)

$$\mu_{sim} = Max(\mu_{C^1}(cp_1), \mu_{C^1}(cp_2)). \tag{11}$$

Step 6. Compute the co-residency probability which is defined as $\mu = 1 - \mu_{sim}$.

D. Determination Rule

As stated above, there are significant differences between the cache load changes when the VMs are co-resident and are not co-resident. Therefore, the threshold value (η) of co-residency probability can be used to classify the co-residency probability.

In this paper, we get three cache load sets, corresponding to three different stages during the process of accessing to the victim VM's services. We calculate ρ_1 and ρ_2 to depict the cache load changes produced by the victim VM. In the ideal case, if the malicious VM is co-resident with the victim VM, both of the two co-residency probabilities should be larger than η. Moreover, the expectation of the second load set should not be smaller than the expectation of the first and third set. Therefore, we illustrate the determination conditions as follows:

$$\rho_1 \ge \eta, \tag{12}$$

$$\rho_2 \ge \eta, \tag{13}$$

$$Ex_{Fcl^3} \ge Ex_{Scl_2^3}, \tag{14}$$

$$Ex_{Scl_1^3} \ge Ex_{Tcl^3}. \tag{15}$$

Finally, we conclude the determination rule: if all the four conditions are satisfied simultaneously, the victim VM is co-resident with the malicious VM; otherwise, they are not co-resident.

IV. EXPERIMENTS

The overall evaluation of the proposed scheme comprised of *(1)* experiments to verify the feasibility of our co-residency detection scheme in a virtualized computing environment and *(2)* experiments to compare our proposed scheme with the existing method to evaluate the efficacy.

All the experimental data were collected from our cloud computing platform based on KVM, which consists of 30 physical servers. The hardware configurations for each server are: Intel Xeon 2.13GHz CPU, 8GB main memory and 500GB hard disk. Ubuntu Linux was run in each VM. 2 VCPU, 512MB main memory and 8 GB hard disk were allocated to each VM. There were two kinds of applications running in the VMs. They were web server based on apache2 and file encryption based on RSA, which were web-intensive and computation-intensive applications. When the web server was deployed, JMeter was used to simulate multi-users to access the web site.

In this section, we considered three different cache load scenarios.

1) Steady scenario, S-scenario for short, described the situation that the running state of noisy VM was not changed, where the running state included "run" and "stop". "Run" referred that VM was launched and "stop" referred that VM was shut down.

2) Increasing scenario, I-scenario for short, described a sudden load increment produced by the noisy VM in the probing period.

3) Decreasing scenario, D-scenario for short, represented that noisy VM changes its state to "stop" in the probing period, which led to a sudden decrease to the cache load.

It took about 0.045s to complete one cache load measuring in our experimental setting. Therefore, we performed the *Prime-Probe* measuring one time each second in the following experiments.

B. Training the threshold

In this section, we aimed to evaluate the feasibility of the co-residency detection scheme and trained the threshold. In

the experiment, we divided the VMs into four categories. They were: *(1)* malicious VM; *(2)* victim VM which was co-resident with the malicious VM; *(3)* victim VM which was not co-resident with the malicious VM; *(4)* noisy VMs located in the same host with the monitoring VM. Noisy VMs changed the cache load scenario during the probing phase. RSA-based file encryption and apache2-based web server run in victim VM and noisy VM. The victim VM performed the co-residency detection scheme.

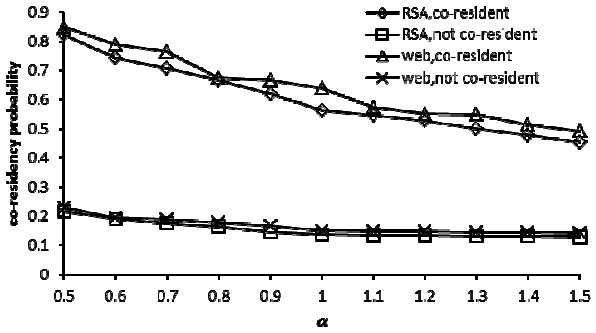

Fig. 2. Co-residency probability in S-scenario.

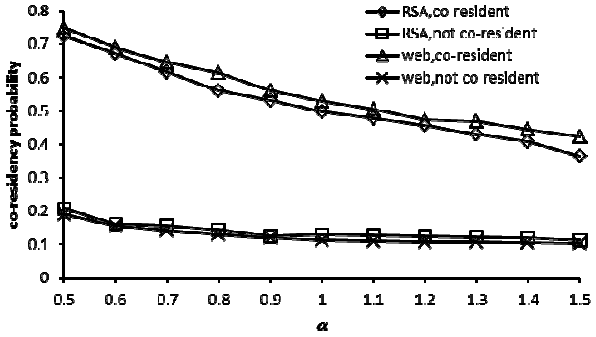

Fig. 3. Co-residency probability in I-scenario.

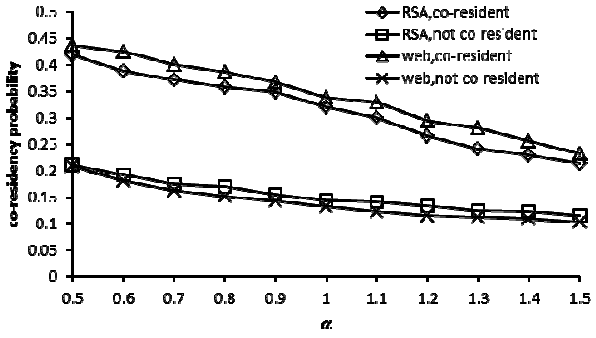

Fig. 4. Co-residency probability in D-scenario.

To ensure that η could clearly distinguish the case of co-residency from not co-residency, in this experiment, we assumed that noisy VM only changed its running states during the phase of sampling *Fcl*. And the state change pattern was in accordance with the S-scenario, D-scenario and I-scenario. In S-scenario, the noisy VM did not change its running state. In I-scenario, noisy VM changed its status from "stop" to "run". In D-scenario, noisy VM changed its status from "run" to "stop".

In this experiment, we performed the co-residency detection for 100 times and calculated the co-residency probability, when the VMs were co-resident and not co-resident respectively. In each detection, encryption application and web application were deployed in the victim VM and noisy VM. The experimental results were illustrated in Fig. 2, Fig. 3 and Fig. 4. In these figures, the co-resident probability as a function of α was illustrated in S-scenario, I-scenario and D-scenario, respectively.

From these three figures, we could observe that the co-residency probability decreased with the increasing of α, regardless of whether the victim VM and malicious VM were co-resident. Moreover, in each distinct load scenario, the co-residency probability when the VMs were co-resident was significantly different from the co-residency probability when the VMs were not co-resident, regardless of which application was deployed.

According to the experimental results depicted in the figures, assigning α to 0.5, co-resident probabilities in different cache load scenarios were always larger than 0.4 when the VMs were co-resident, while the probabilities were always smaller than 0.2 when the VMs were not co-resident. So, the co-residency probabilities were significant different when the VMs were co-resident and not co-resident, regardless of which load scenarios, when α was 0.5. Therefore, assigning the value between 0.2 and 0.4 to η, such as 0.3, the co-resident detection scheme could detect VMs co-residency properly and precisely, without knowing the load scenarios.

C. *Efficacy of co-residency detection scheme*

In Experiment *1)*, we assumed that the state changes of noisy VM only occurred in the stage of sampling *Fcl*. In this section, the efficacy of our proposed co-residency detection scheme was evaluated, without the constraints on the state changes of noisy VM. Therefore, in this testing, 8 typical state changes scenarios of the noisy VM were constructed.

We used the pattern "$state_1$->$state_2$->$state_3$" to depict the process of state changes during the detection, where $state_1$ referred to the state in the phase of sampling *Fcl*, $state_2$ referred to state in the phase of sampling *Scl,* and $state_3$ referred to the state in the phase of sampling *Lcl*. In the experiment, a random function r()={0,1} was used to control the state changes. If r() returned 0, state of the noisy VM was "run". If r() returned 1, state was "stop". In this way, the random load scenarios were obtained.

In each scenario, we produced 200 random experiments, including 100 experiments when the VMs were co-resident and 100 experiments when the VMs were not co-resident. Then we counted the *true detection rate* (*tdr*), which was defined in (16)

$$tdr = tt / at * 100\%. \tag{16}$$

Where *tt* referred to the number of successful detections, and *at* referred to the number of total detections.

In these experiments, we compared our proposed scheme with the other two schemes, where the scheme I calculated the similarity degree between *Fcl* and *Scl* to implement the detection [3], and the scheme II calculated the similarity degree between Fcl^3 and Scl^3, which was the advanced version of scheme I. The experimental results were showed in

Fig. 5.

As could be seen from Fig. 5, the results presented a higher true detection rate using our proposed scheme (VCDS), which is about 20% higher than using scheme I and scheme II. We considered the reason was twofold. First, load preprocessor and load predictor were used to process the raw measurements, which could decrease the impact of the noises which were introduced by the system and other co-resident VMs. This could be verified by the fact that the *tdr* of scheme II was higher than *tdr* of scheme I. Second, one more cache load set defined as *Tcl* was sampled, which increased the information about the cache load changes related to the victim VM. This could be verified by the fact that the *tdr* of scheme III was higher than *tdr* of scheme II as well.

On the other hand, Fig. 6 presented that the time costs of VCDS was larger than scheme I and scheme II, which was mainly because of the additional cache load set.

Ultimately, we can conclude that, although the time cost is increased, the efficacy of our proposed detection scheme is improved.

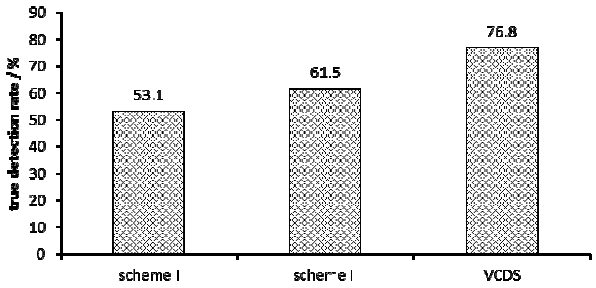

Fig. 5. True detection rate.

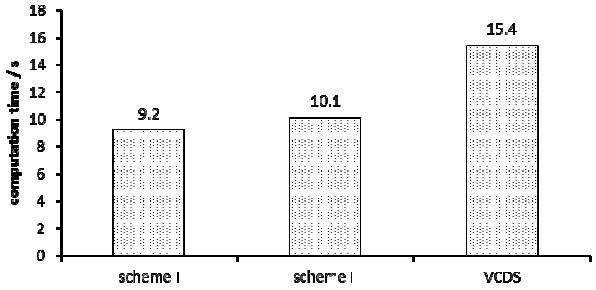

Fig. 6. Computation time.

V. CONCLUSIONS

In this paper, we proposed a novel VMs co-residency detection scheme via cache-based side channel attacks. Considering the interference from other co-resident virtual machines, this scheme sampled three cache load sets and used the techniques of load preprocessing, load predicting and normal cloud model to process the raw measurements, which was the first trail in the field of side channel analysis based on cache, to the best of our knowledge. The experimental results demonstrated that our scheme could improve the true detection rate effectively, with the interference of the noisy VM which was co-resident with the attack VM.

REFERENCES

[1] R. W. Huang, X. L. Gui, S. Yu, W. Zhuang, "Study of Privacy-Preserving Framework for Cloud Storage", *Computer Science and Information Systems*, vol. 8, no. 3, pp. 801–819, 2011. [Online]. Available: http://dx.doi.org/10.2298/CSIS100327029R

[2] H. Jin, G. F. Xiang, D. Q. Zou, S. Wu, F. Zhao, M. Li, W. D. Zheng, "A VMM-based intrusion prevention system in cloud computing environment", *Journal of Supercomputing*, to be published.

[3] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems", *in Proc. of the 16th Annual International Cryptology Conference*, Berlin, 1996, pp. 104–113.

[4] T. Ristenpart T, T. Eran, S. Hovav, S. Stefan, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds", *in Proc. of the 6th ACM Conference on Computer and Communications Security*, New York, 2009, pp. 199–212.

[5] Y. Zhang, J. Ari, O. Alina, and K.P Michael, "Home Alone: Co-residency detection in the cloud via side-channel analysis", i*n Proc. of the 2011 IEEE Symposium on Security and Privacy*, New York, 2011, pp. 313–328. [Online]. Available: http://dx.doi.org/10.1109/SP.2011.31

[6] E. Tromer, D. A. Osvik, A. Shamir, "Efficient cache attacks on AES, and countermeasure", *Journal of Cryptology*, vol. 23, no. 1, pp. 37–71, 2010. [Online]. Available: http://dx.doi.org/10.1007/s00145-009-9049-y

[7] M. Andreolini, S. Casolari, "Load prediction models in web-based systems", *in Proc. 1st International Conference on Performance Evaluation Methodologies and Tools*. New York, 2006, pp. 27–36.

[8] D. Y. Li, C. Y. Liu, W. Y. Gan, "A new cognitive model: cloud model", *International Journal of Intelligent Systems*, vol. 24, no. 3, pp. 357–375, 2009. [Online]. Available: http://dx.doi.org/10.1002/int.20340