Supplemental Material: Scalable Distributed Service Integrity Attestation for Software-as-a-Service Clouds

Juan Du, *Member, IEEE*, Daniel J. Dean, Yongmin Tan, *Member, IEEE*, Xiaohui Gu, *Senior Member, IEEE*, Ting Yu, *Member, IEEE*

1 Detailed Proofs and Additional Examples

Proposition 1: Given an inconsistency graph G, let C_G be a minimum vertex cover of G. Then the number of malicious service providers is no less than $|C_G|$.

Proof: We can prove Proposition 1 by contradiction. Suppose the number of malicious service providers is less than $|C_G|$. Then the graph formed by malicious nodes cannot cover the entire graph, which means there exists one edge that is not incident to any of the malicious nodes. Thus, the edge must be incident to two benign nodes. Since two benign nodes always agree with each other, this contradicts with the existence of an inconsistency link between them. □

Proposition 2: Given an integrated inconsistency graph G and the upper bound of the number of malicious service providers K, a node p must be a malicious service provider if and only if

$$|N_p| + |C_{G'_p}| > K \quad (1)$$

where $|N_p|$ is the neighbor size of p, and $|C_{G'_p}|$ is the size of the minimum vertex cover of the *residual* inconsistency graph after removing p and its neighbors from G.

Proof: We can prove Proposition 2 by contradiction. Suppose there exists a benign service provider p that satisfies $|N_p| + |C_{G'_p}| > K$. Since p is inconsistent with its neighbors, the neighbors must be malicious. Then the total number of malicious service providers can be calculated by adding the number of p's neighbors and the number of malicious service providers in the residual graph. According to Proposition 1, we can use the size of a minimum vertex cover to serve as the lower bound number of malicious service providers in the residual graph $|C_{G'_p}|$. Since the total number of malicious service providers is no more than K, $|N_p| + |C_{G'_p}| \leq K$, which contradicts with the assumption $|N_p| + |C_{G'_p}| > K$. □

• *Juan Du is with Amazon. Yongmin Tan is with MathWorks. The work was done when they were PhD students at the North Carolina State University. Daniel J. Dean, Xiaohui Gu, and Ting Yu are with Department of Computer Science, North Carolina State University. Their email addresses are duju@amazon.com, djdean2@ncsu.edu, yongmin.tan@mathworks.com, gu@csc.ncsu.edu, yu@csc.ncsu.edu*

```
PinpointMaliciousSPs(G, G_i)
1.  for every K ∈ [|C_G|, ⌊N/2⌋]
2.      Ω = ∅, R = ∅
3.      for every node p in G
4.          compute |N_p| + |C_{G'_p}|
5.          if (|N_p| + |C_{G'_p}| > K)
6.              Ω = Ω ∪ {p}
7.      final malicious node set R = R ∪ Ω
8.      if R = ∅
9.          continue
10.     else
11.         for every G_i
12.             compute M_i
13.             set Ω_i to the subset of Ω appearing in G_i
14.             if (Ω_i ∩ M_i ≠ ∅)
15.                 R = R ∪ M_i
16. return all sets of R
```

Fig. 1. **Malicious service provider pinpointing algorithm.**

Figure 1 shows the pseudo-code of our integrated attestation graph analysis algorithm, where R is the final set of malicious service providers.

For example, in Figure 2, the true malicious nodes are $\{p_7, p_8, p_9, p_{10}\}$. If we set $K = 4$, the inconsistency graph analysis returns $\Omega = \{p_7\}$. Furthermore, by checking the inconsistency graph of the function f_3, we can find $\Omega_3 = \{p_7\}$ has overlap with the minority clique $M_3 = \{p_7, p_{10}\}$. We can then infer p_{10} to be malicious as well.

2 Detailed Results of Analytical Study about Int Test.

We now present our analytical study results about IntTest.

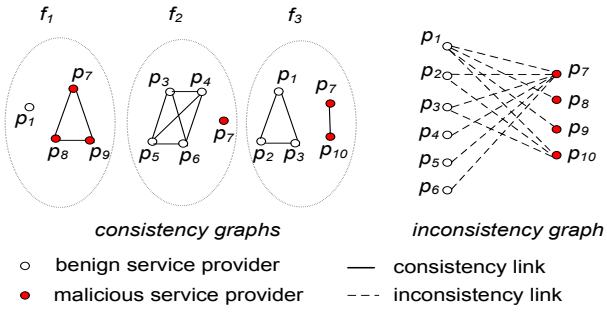
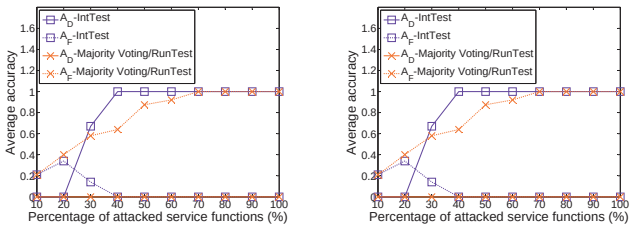


Fig. 2. **Another example for integrated graph analysis with four malicious nodes.**



(a) Aggressive attackers (b) Conservative attackers

Fig. 3. Malicious attackers pinpointing accuracy comparison with 40% service providers being malicious.

Proposition 3: Given an accurate upper bound of the number of malicious service providers K, if malicious service providers always collude together, IntTest has 0 false positive.

Proof: According to Proposition 2, any node identified through inconsistency graph must be malicious. Therefore, any node in the subset of malicious nodes identified through inconsistency graph, e.g., Ω_i, must be malicious. If, for all functions, $\Omega_i \cap M_i = \emptyset$, our algorithm returns set Ω as malicious set, which contains only malicious service providers. Otherwise, there exists some function f_i such that $\Omega_i \cap M_i \neq \emptyset$. For any $p_b \in (\Omega_i \cap M_i)$, p_b must be malicious because it belongs to Ω_i. Suppose our algorithm has false positives, which means there exists a benign node p_g, where $p_g \in M_i$ but $p_g \notin \Omega_i$. Since $p_g \in M_i$, p_g must be outside of the maximum clique. Thus, the maximum clique must be formed by malicious nodes. This indicates that malicious node p_b must disagree with at least one of the malicious nodes in the maximum clique, which contradicts with our assumption that attackers always collude together as a single group. □

Although our algorithm cannot guarantee zero false positive when there are multiple independent colluding groups, it will be difficult for attackers to escape our detection with multiple independent colluding groups since attackers will have inconsistency links not only with benign nodes but also with other groups of malicious nodes.

We now quantify the damage that collusive attackers can make without being detected. We assume that collusive attackers are intelligent in that they can select service functions to attack together in order to maximize the damage they can bring to the system. The damage is defined as follows.

Definition 5: The *Damage Degree*, denoted by D, is the number of the service functions on which malicious service providers misbehave without being detected.

Since attackers can escape detection by forming the majority in per-function consistency graphs, attackers can select service functions that have less benign service providers. Suppose there are n service functions, $f_1, ..., f_n$, ranked in the ascending order of the number of benign service providers participating in the function. The number of benign and malicious service providers participating in the function f_i are g_i and b_i, respectively. The total number of malicious service providers are b, where $b = b_1 + ... + b_n$. Thus, in order to escape detection in per-function consistency graphs, attackers need to take majority in all attacked functions, which means they can attack up to k functions at different time, where $g_k \leq b \leq g_{k+1}$. That is, attackers can only attack functions $f_1, ..., f_k$. For functions $f_{k+1}, ..., f_n$, attackers cannot form a majority so that any misbehavior on these functions will be detected by our algorithm.

However, attackers cannot form majority in all the k functions at the same time. The number of functions that attackers can attack simultaneously is significantly limited. If b satisfies the following equation,

$$\sum_{i=1}^{m} g_i \leq b \leq \sum_{i=1}^{m+1} g_i, \tag{2}$$

then attackers cannot attack more than m functions at the same time, which means the damage degree $D = m$.

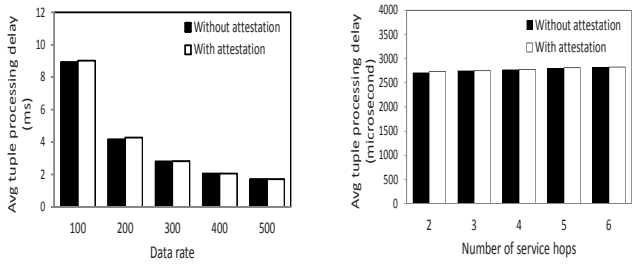
Moreover, a single attacker cannot participate in unlimited number of service functions. By attacking function f_i, the attacker may produce g_i inconsistency links with all the benign service providers provisioning f_i, where g_i is the number of benign service providers in that function. Suppose an attacker p_b provides functions $f_i, f_{i+1}, ...f_{i+k}$. In the global inconsistency graph, in order to escape detection, every single attacker p_b needs to limit the number of inconsistency links. If attackers are too greedy to attack more service functions or attack functions that they cannot form the majority, they will get detected by our algorithm.

3 Additional Results and Overhead Comparison

We show the results when the percentage of malicious service providers to 40% and repeat the above two sets of experiments. Figure 3(a) shows the comparison results under the aggressive attack scenarios while Figure 3(b) shows the comparison results under the conservative attack scenarios. The results show that IntTest still achieves better detection accuracy than the other alternatives. Note that when there are a high percentage of malicious attackers, majority voting based schemes fail to identify any attacker (i.e., $A_D = 0$), while IntTest can still detect all attackers when the attackers try to compromise many service functions.

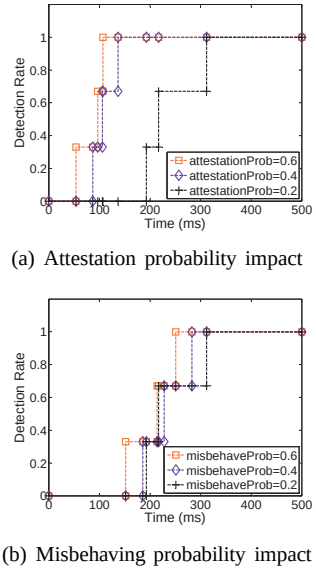
# of providers	Consistency graph	Inconsistency graph
200	4.22 ± 0.018 ms	1.64 ± 0.001 ms
400	15.89 ± 0.013ms	6.52 ± 0.004 ms
600	35.29 ± 0.095 ms	15.11 ± 0.015 ms
800	62.62 ± 0.021 ms	26.18 ± 0.350 ms
1000	100.00 ± 0.434 ms	39.98 ± 0.179 ms

TABLE 1
Graph analysis time in IntTest.



(a) (b)

Fig. 4. Performance impact of IntTest to distributed data processing services.



(a) Attestation probability impact

(b) Misbehaving probability impact

Fig. 5. Sampled sensitivity study results.

We also measure the computation overhead for the graph analysis. Table 1 shows the graph analysis time for both consistency graphs and inconsistency graph including both the mean and standard deviation values, where the number of service providers varies from 200 to 1000. The analysis time for consistency graphs is the sum of per-function analysis time. As the table shows, the total time for both consistency and inconsistency graph analysis is less than 140 milliseconds given 1000 service providers and 2000 service components in the system. Note that we start from a complete graph connected by consistency links only and IntTest only triggers the graph analysis algorithm when any new inconsistency links are captured by the probabilistic attestation. Although the global inconsistency graph analysis algorithm relies on the solution to the minimum vertex cover problem, a known NP-hard problem, we observe that the computation time is generally short. If all malicious service providers collude, the inconsistency graph will be bipartite because there are only edges between the benign and the malicious service providers. In this case, the minimum vertex cover is equivalent to the maximum matching problem that can be solved in polynomial time [1]. We can also employ approximation algorithms [2] if the inconsistency graph analysis overhead becomes the bottleneck in our system.

We now evaluate the impact of our integrity attestation scheme on the data processing delay, an important performance metric for data processing systems. The data processing delay is measured as the average tuple turnaround time, which is the duration between the time when the first data tuple enters the system and the time when the last data tuple leaves the system over the total number of tuples processed. Figure 4(a) shows the average per-tuple processing delay under different data rate. The results show that IntTest imposes little overhead to the dataflow processing delay. Note that the processing delay is lower under higher data rate. This is because the turnaround time is smaller when more data are sent into the system before the system reaches its maximum capacity. Figure 4(b) shows the average per-tuple processing delay under different numbers of service hops. The results show that our scheme only imposes about tens of microseconds extra delay. As the overhead of our attestation scheme is kept low, we also expect the network traffic impact to co-located applications to be low as well. Additionally, we can leverage VM resource capping mechanisms (e.g. cgroup in KVM) to isolate the performance impact to other applications. Note that our experiments are conducted in a production cluster system with high speed networks. Thus, the processing delay does not increase too much when we increase the number of service hops.

We now evaluate the impact of various system parameters on the effectiveness of our algorithm. Figure 5(a) shows the time to detect each malicious service provider under different attestation probability P_u. With higher attestation probability, IntTest has more opportunities to capture the sneaky occasional misbehavior of attackers. Thus, with a higher attestation probability, we can detect malicious service providers earlier. However, the system overhead, in terms of attestation traffic, would increase accordingly since IntTest performs attestation on more data. Figure 5(b) shows the detection rate under different misbehaving probabilities, where attestation probability is fixed at 0.2. The more frequently malicious service providers misbehave, the more opportunities are given to our scheme to capture the misbehavior. Therefore, it takes less time to detect malicious service providers with a higher misbehaving probability.

4 Limitation Discussion

Although our experimental results have shown that IntTest can achieve better scalability and higher detection accuracy than the state-of-the-art schemes (i.e., majority voting, consistency graph analysis only scheme [38]), IntTest still has a set of limitations that require further study.

Malicious service providers may still escape detection if they can manage to satisfy three conditions: 1) only attacking a limited number of service functions, 2) taking majority in all the attacked services, and 3) having less inconsistency links than benign service providers. However, IntTest makes it difficult for attackers to attack popular service functions since popular service functions often attract many different service providers for profit. The attackers can hardly take majority in those popular service functions. Moreover, service provisioning in SaaS clouds is not free, which involves registering services, paying resources and hosting fees, and passing the service hosting verification. Malicious attackers have to pay a high cost to take majority in popular services. In contrast, attacking unpopular services will have much smaller impact since only a few users rely on them. More importantly, the popularity of different service functions is only available to the portal node of the cloud. It is difficult, if not totally impossible, for individual service providers to guess the popularity of different service functions. Thus, it is highly possible that the attackers will try to compromise a randomly selected service and fail to take the majority, and thus expose themselves.

Let us assume that attackers can infer the popularity of different services from some side channels and only attack those unpopular services where they can take majority. Under those circumstances, it is more effective to apply challenge-based approaches (e.g., [3]–[6]) that rely on the portal or a trusted entity to redundantly compute results for verification. However, the portal node or a trusted entity may not have the required software or sufficient resources to redundantly compute *all* service functions. Thus, the benefit of our approach is to reduce the reliance on the challenge-based verification. Our approach can be combined with the challenge-based scheme. For example, we can use IntTest as the default attestation scheme and dynamically invoke the challenge-based verification only when the system cannot make the pinpointing decision or encounter conflicting pinpointing results.

The current prototype of the IntTest system does not support non-deterministic service functions where the service might return different results for the same input. We can use a user-defined distance function to partially alleviate the problem: as long as the distance between two results based on the distance function is within a certain threshold, we say two results are consistent. We can also leverage the techniques proposed in [39] to attest non-deterministic services.

5 Related Work

Remote attestation techniques often use a challenge-response paradigm to ensure that a remote software platform truthful executes a program that is not compromised or altered by attackers. Attestation can be performed at system-level or application-level. System-level attestation techniques [3]–[11] require a trusted entity (e.g., trusted hardware or secure kernel) to co-exist with the remote attested platform. For example, SWATT [7] computes a checksum of the memory whenever receiving a challenge. BIND provides a fine-grained code attestation scheme for distributed systems [10]. Alam et al. proposed a set of specification and verification schemes for attesting the behavior of business processes [12]. However, in SaaS clouds, it is often impractical to assume the existence of a trusted entity at the remote third-party service provider site. In contrast, our approach does not require any trusted entity to be deployed on the remote attested service provider site.

Application-level auditing schemes have been proposed under different distributed computing contexts such as peer-to-peer systems [13], volunteer computing systems [11], publish-subscribe systems [14], cloud storage systems [15], distributed web applications [16], and database systems [17]. Such schemes either construct integrity evidence through cryptographical transformation of application data [13], [15] or rely on emulation [16], [18] to detect deviation from expected execution results. Generally speaking, an auditor needs to challenge the untrusted party periodically. Auditors could be trusted parties [16], or a group of untrusted auditors [13], [19]. For example, Manrose et al. proposed a remote attestation technique that focuses on detecting misbehaviors of skipping computations for lower resource expenditure [11]. The worker nodes have to provide execution proofs with the help of compiler techniques. Thus, the verifier needs to know the internal workflow of the computation performed on the worker nodes for detecting cheating behaviors. Belenkiy et al. proposed an application-level verification approach that relies on the trusted entity (the boss) to redundantly compute results for verifying result correctness [18]. In comparison, IntTest supports black-box service integrity attestation in cloud systems, which does not require any internal knowledge about the third-party services. Moreover, our approach supports integrity attestations for large-scale SaaS cloud systems where the portal nodes do not have the software or sufficient resources to redundantly compute the results returned by different services.

Previous work has studied the problem of verifying the correctness of remote computations. Golle and Miranov proposed “uncheatable computations” [20] that allows the user to perform a small amount of local computation to verify the correctness of outsourced computations. In [21], the authors conducted a survey on result-checking and self-correcting programs for software correctness verification. However, result-checking is often limited to specific arithmetic functions. It is challenging to design a result-

checker for general computations. Gennaro et al. explored a cryptographic approach to verifying the computations that are outsourced to untrusted devices [22]. In comparison, IntTest does not require the remote service provisioning site or the portal node to perform any extra computation such as proof checking. However, IntTest can be combined with the above techniques when it cannot make the pinpointing decision or encounter conflicting pinpointing results.

Trust management in multi-party systems has been studied under different application contexts [8], [14], [23]–[25]. Generally, users or components of a distributed system, are evaluated according to some trust metrics. Higher trust scores are assigned to users or components who follow the rules honestly. In contrast, our approach evaluates different service providers by actively attesting them rather than performing passive monitoring on some pre-defined trust metrics. Qing et al. proposed a compromised core algorithm using inconsistency graphs to identify malicious nodes in sensor networks [26]. In contrast, IntTest employs both consistency and inconsistency relationships to pinpoint malicious service providers. Moreover, unlike sensor networks, SOA systems consist of distributed service providers offering diversified service functions.

Byzantine fault tolerance (BFT) techniques [19], [27]–[30] can pinpoint malicious nodes among replicated services given no more than one third of total replicas being faulty. However, the goal of BFT is to achieve consistency in replicated systems such as quorum systems and distributed database systems. In contrast, IntTest aims at pinpointing all malicious service providers in an SaaS cloud system that consists of different services. BFT requires all replicas communicate with each other all the time via a certain agreement protocol, which is impractical for large-scale cloud systems due to scalability and deployment challenges. In contrast, IntTest performs probabilistic replay-based attestation and employs comprehensive attestation graph analysis to achieve both scalability and high detection accuracy.

Our work is also related to Sybil defense techniques [31]–[34] that leverage social networks among the users constructed by observing social interactions among users, and rely on the quotient cut to expose sybil identities. In contrast, our approach does not assume any prior knowledge about third-party service providers and actively attests service providers using probabilistic data replay. Our pinpointing scheme provides integrated graph analysis algorithms (i.e., clique discovery in the consistency graph and minimum vertex cover in the inconsistency graph) for better accuracy. Moreover, different from open distributed systems such as peer-to-peer networks where anyone can join for free or little cost, SaaS cloud systems often impose a certain cost for each service provider to provide a service. Thus, it can be prohibitively expensive for attackers to launch Sybil attacks in SaaS clouds. Service providers cannot easily forge identities in SaaS clouds since they need to provide certificates for hosting a certain service and must be authenticated and approved by the cloud system.

Security protection for cloud systems has recently received much attention [15], [35]–[37]. Ristenpart et. al. explored the security holes of existing deployed cloud systems, and identified that current cloud deployments are vulnerable to a cross-VM side channel attack [35]. Erway et al. presented a dynamic provable data possession (PDP) framework for cloud storage systems [36]. In comparison, our work focuses on assuring distributed service integrity for SaaS clouds.

REFERENCES

[1] G. Bancerek, "Konig's theorem," *Journal of Formalized Mathematics*, vol. 2, pp. 2–3, 1990.

[2] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, unabridged ed. Englewood Cliffs, New Jersey, U.S.A.: Dover Publications, May 1998.

[3] J. Garay and L. Huelsbergen, "Software integrity protection using timed executable agents," in *Proceedings of ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Taiwan, Mar. 2006.

[4] T. Garfinkel, B. Pfaff, and et. al., "Terra: A virtual machine-based platform for trusted computing," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2003.

[5] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a tcg-based integrity measurement architecture," in *Proceedings of 13th USENIX Security Symposium*, San Diego, CA, Aug. 2004.

[6] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, "Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2005.

[7] A. Seshadri, A. Perrig, L. V. Doorn, and P. Khosla, "Swatt: Software-based attestation for embedded devices," in *IEEE Symposium on Security and Privacy*, May 2004.

[8] S. Berger, R. Caceres, and et. al., "TVDc: Managing security in the trusted virtual datacenter," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 1, pp. 40–47, 2008.

[9] E. Kaiser, W. Feng, and T. Schluessler, "Fides: Remote anomaly-based cheat detection using client emulation," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, 2009.

[10] E. Shi, A. Perrig, and L. V. Doorn, "Bind: A fine-grained attestation service for secure distributed systems," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2005.

[11] F. Monrose, P. Wyckoff, and A. D. Rubin, "Distributed execution with remote audit," in *Proceedings of ISOC Network and Distributed System Security Symposium (NDSS)*, Feb. 1999.

[12] M. Alam, M. Nauman, X. Zhang, T. Ali, and P. C. Hung, "Behavioral attestation for business processes," in *IEEE International Conference on Web Services*, 2009.

[13] A. Haeberlen, P. Kuznetsov, and P. Druschel, "Peerreview: Practical accountability for distributed systems," in *ACM Symposium on Operating Systems Principles*, 2007.

[14] M. Srivatsa and L. Liu, "Securing publish-subscribe overlay services with eventguard," *Proc. of ACM Computer and Communication Security (CCS)*, 2005.

[15] K. Bowers, A. Juels, and A. Oprea, "Hail: A high-availability and integrity layer for cloud storage," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, 2009.

[16] K. Vikram, A. Prateek, and B. Livshits, "Ripley: Automatically securing web 2.0 applications through replicated execution," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, 2009.

[17] M. Xie, H. Wang, J. Yin, and X. Meng, "Integrity auditing of outsourced data," in *International Conference on Very Large Data Base (VLDB)*, 2007.

[18] M. Belenkiy, M. Chase, C. Erway, and et. al., "Incentivizing outsourced computation," in *Proceedings of the 3rd international workshop on Economics of networked systems (NetEcon)*, Aug. 2008.

[19] B. Liskov and R. Rodrigues, "Tolerating byzantine faulty clients in a quorum system," in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2006.

[20] P. Golle and I. Mironov, "Uncheatable distributed computations," in *Topics in Cryptology CT-RSA 2001*, ser. Lecture Notes in Computer Science, D. Naccache, Ed. Springer Berlin / Heidelberg, 2001, vol. 2020, pp. 425–440.

[21] H. Wasserman and M. Blum, "Software reliability via run-time result-checking," *J. ACM*, vol. 44, pp. 826–849, November 1997.

[22] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Advances in Cryptology CRYPTO 2010*, ser. Lecture Notes in Computer Science, T. Rabin, Ed. Springer Berlin / Heidelberg, 2010, vol. 6223, pp. 465–482.

[23] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The EigenTrust Algorithm for Reputation Management in P2P Networks," in *Proceedings of the 12th International World Wide Web Conference*, 2003.

[24] M. Srivatsa, S. Balfe, K. G. Paterson, and P. Rohatgi, "Trust management for secure information flows," in *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, 2008.

[25] S. Pandit, D. H. Chau, S. Wang, and C. Faloutsos, "NetProbe: A Fast and Scalable System for Fraud Detection in Online Auction Networks," in *Proceedings of the 16th international conference on World Wide Web (WWW)*, 2007.

[26] Q. Zhang, T. Yu, and P. Ning, "A framework for identifying compromised nodes in wireless sensor networks," *ACM TISSEC*, vol. 11, no. 3, 2008.

[27] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems, 4(3)*, 1982.

[28] T. Ho, B. Leong, R. Koetter, and et. al., "Byzantine modification detection in multicast networks using randomized network coding," in *IEEE ISIT*, 2004.

[29] M. Castro and B. Liskov, "Practcal byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI)*, New Orleans, LA, 1999.

[30] B. Vandiver, H. Balakrishnan, B. Liskov, and S. Madden, "Tolerating byzantine faults in transaction processing systems using commit barrier scheduling," in *Proceedings of 21st ACM Symposium on Operating Systems Principles (SOSP)*, 2007.

[31] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "Sybilguard: Defending against sybil attacks via social networks," in *Proceedings of ACM SIGCOMM*, 2006.

[32] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, "Sybillimit: A near-optimal social network defense against sybil attacks," in *Proceedings of ACM SIGCOMM*, 2006.

[33] A. Mislove, A. Post, K. Gummadi, and P. Druschel, "Ostra: Leveraging trust to thwart unwanted communication," in *USENIX NSDI*, 2008.

[34] N. Tran, B. Min, J. Li, and L. Subramanian., "Sybil-resilient online content voting," in *USENIX NSDI*, 2009.

[35] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off my cloud! exploring information leakage in third- party compute clouds," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, 2009.

[36] C. Erway, C. Papamanthou, A. Kupcu, and R. Tamassia, "Dynamic provable data possession," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, 2009.

[37] P. Williams, R. Sion, and D. Shasha, "The blind stone tablet: Outsourcing durability," in *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS)*, 2009.

[38] J. Du, W. Wei, X. Gu, and T. Yu, "Runtest: Assuring integrity of dataflow processing in cloud computing infrastructures," in *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2010.

[39] M. Backes, P. Druschel, A. Haeberlen, and D. Unruhu, "Csar: A practical and provable technique to make randomized systems accountable," in *Proc. of Annual Network and Distributed System Security Symposium (NDSS)*, 2009.