Co-Location-Resistant Clouds

Yossi Azar[*]
Tel-Aviv University
azar@tau.ac.il

Seny Kamara
Microsoft Research
senyk@microsoft.com

Ishai Menache
Microsoft Research
ishai@microsoft.com

Mariana Raykova[*]
SRI International
mariana.raykova@sri.com

Bruce Shepherd[*]
McGill University
bruce.shepherd@mcgill.ca

Abstract

We consider the problem of designing multi-tenant public infrastructure clouds resistant to cross-VM attacks without relying on single-tenancy or on assumptions about the cloud's servers. In a cross-VM attack (which have been demonstrated recently in Amazon EC2) an adversary launches malicious virtual machines (VM) that perform side-channel attacks against co-located VMs in order to recover their contents.

We propose a formal model in which to design and analyze *secure* VM placement algorithms, which are online vector bin packing algorithms that simultaneously satisfy certain optimization constraints and notions of security. We introduce and formalize several notions of security, establishing formal connections between them. We also introduce a new notion of efficiency for online bin packing algorithms that better captures their cost in the setting of cloud computing.

Finally, we propose a secure placement algorithm that achieves our strong notions of security when used with a new cryptographic mechanism we refer to as a shared deployment scheme.

1 Introduction

Cloud computing platforms make computational resources available to clients as a service. With cloud computing, a client can outsource its computation and/or storage to a cloud provider, paying only for the resources used at any time. Cloud computing platforms can be roughly categorized as either public or private. In a private cloud, the underlying infrastructure (i.e., servers, network and storage) is owned and operated by the client whereas in a public cloud (e.g., Amazon EC2 or Microsoft Azure) it is owned and managed by a cloud provider and made available as a service. The benefits of using a public cloud include reliability, elasticity (i.e., computational resources can be increased and decreased quickly) and cost-savings. As such, many private and public organizations are considering migrating their IT infrastructures to a public cloud.

Roughly speaking, a cloud is composed of physical servers and of a controller. The servers are virtualized which means that a single physical server can execute several *virtual machines* (VM) concurrently. This concurrent execution is implemented through a *hypervisor* which is responsible for isolating the co-located VMs from each other and making sure that the server's resources are allocated fairly. The allocation of VMs to physical servers is handled by the controller which

[*]Research conducted in part while visiting Microsoft Research.

monitors the servers' resources and executes a placement algorithm. To use a cloud, a tenant creates and sends to the controller a *deployment*, i.e., a set of VMs that execute its computation. The cost-effectiveness of public clouds stems from several reasons but the most important is *multi-tenancy*, i.e., the cloud provider's ability to execute the workloads of several clients on the same physical server. Multi-tenancy is made possible through *virtualization*

Cross-VM attacks. While multi-tenancy is crucial to cloud computing it also introduces a number of security concerns, including the possibility of *cross-VM attacks*. In a cross-VM attack, a malicious VM bypasses the hypervisor-level isolation to attack co-located VMs. Possible cross-VM attacks range from taking over other VMs by exploiting vulnerabilities in the hypervisor and guest OS, to stealing secrets through side-channels attacks [15, 25].

Currently, the best solution to these attacks is single-tenancy, i.e., letting clients run their VMs on a dedicated server. While single-tenancy obviously mitigates cross-VM attacks, it does not scale since at some point it obviates the economic benefits of public clouds. The design of cross-VM mitigation techniques that preserve multi-tenancy is therefore an important and well-motivated problem in cloud security. Many recent works [22, 23, 14, 1, 17, 21, 7, 26, 11, 9, 20] have explored cross-VM mitigation techniques but all these approaches are restricted to cache-based attacks. In addition, these solutions rely on systems-level assumptions, i.e., they rely on the integrity of the servers' hardware, hypervisors and/or OSs.

Co-location attacks. A basic, but crucial, observation about cross-VM attacks is that they first necessitate the successful completion of a *co-location* attack; that is, an attack in which the adversary strategically creates and launches VMs so that they are co-located with its target VMs. As discussed by Ristenpart et al. [15], there are at least two different kinds of co-location attacks in public clouds. The first are what we refer to as *complete* co-location attacks, where the adversary wants to co-locate with each VM in a given set of target VMs. The second is what we refer to as a *fractional* attack, where the adversary only wants to co-locate with some fraction of the target VMs.

1.1 Our Contributions

In this work, we consider the problem of cross-VM attacks in public clouds, seeking solutions that do not rely on single-tenancy or on systems-level assumptions. At a very high-level, our focus is on mitigating co-location attacks since they are a necessary first step to performing cross-VM attacks. More concretely, our approach is to assign VMs to physical servers in such a way that attack VMs are rarely co-located with target VMs. To do this, we formalize and design *co-location-resistant* placement algorithms which, roughly speaking, protect VMs against complete and fractional co-location attacks. Our main placement algorithm uses randomization to place VMs in a manner that is unpredictable to the adversary and that reduces its probability of successfully completing a co-location attack.

We note that the naive strategy of placing VMs on servers chosen uniformly at random is not feasible in our setting since VMs cannot be placed arbitrarily in practice. Indeed, VM placement algorithms have to satisfy non-trivial optimization constraints which cannot be met by simply placing VMs at random. One of the major contributions of our work is the design of an algorithm that optimizes for these constraints while remaining co-location-resistant to the adversary.

Secure optimization. As far as we know, ours is the first work to consider the design of such "secure optimization" algorithms; that is, optimization algorithms that also provide some form of security. We believe the study of secure optimization algorithms is an interesting research direction at the intersection of algorithms, security and cryptography and could have applications, not only to cloud computing, but more generally to distributed systems.

Theoretical cloud security. While cloud security has attracted a lot of attention from the research community, there are no formal security models in which to rigorously analyze the security of cloud systems. Another major contribution of our work is to provide such a model for the study of security against cross-VM and co-location attacks. Though our main results are algorithmic in nature (with the exception of our notion of shared deployments from Section 6), our model is cryptographic. More precisely, we model and analyze the properties of our placement algorithm using the provable security methodology from Theoretical Cryptography. While we focus here on VM placement, we believe that our core model and principles can be extended in future work to analyze the security of cloud systems against other threats. As a concrete example, we believe our ideas could be extended and applied to hypervisor-level core scheduling to prevent cache-based attacks on multi-core servers.

1.2 Overview of Our Work

In our model, a cloud Cld consists of a set of virtualized physical servers $(\mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$ and a controller Ctrlr_Π that allocates VMs to servers according to a placement algorithm Π. VMs are created by running a deployment scheme Δ. Each VM includes a resource vector $\mathbf{r}$ which captures its resource requirements (e.g., I/O, CPU and memory requirements); each dimension of the vector corresponds to a particular resource. Similarly, each physical server has a capacity vector which determines the maximum amount of any resource it can provide. In our work, we require two main properties from such a cloud: *efficiency* and *security*.

Efficiency. In our context, the efficiency of a cloud refers to its total resource usage when executing deployments and not just the time to execute a VM. In fact, the most expensive resource when running a cloud is energy. So, intuitively, an efficient cloud is one that minimizes its total resource consumption. Efficiency in this sense is therefore determined by the controller and its placement algorithm Π. The problem of allocating VMs to servers is known to be an instance of the online vector bin packing problem, where one must allocate a stream of vectors to bins in such a way as to minimize the number of bins and without exceeding the capacity of the bins (see, e.g., [13, 8] for detailed discussions). The quality of online algorithms are evaluated using the notion of competitive ratio which bounds the ratio between the cost of the algorithm and the cost of the optimal algorithm over the worst-case input stream. For bin packing algorithms the cost is usually defined to be the number of bins used. We point out in this work, however, that the number-of-bins cost is inappropriate for cloud computing where efficiency is determined by the total amount of resources used.

We therefore propose a more relevant cost function which we refer to as *server-time* which is the sum over time of the number of utilized servers. We believe minimizing server-time in infrastructure clouds is crucial for at least two reasons. First, it seems to be correlated to energy consumption. [1] The second is that minimizing server-time in a public cloud increases the amount of resources available in the data center for other purposes (e.g., for use as a private cloud).

Security. The second property we want is *security* which, in our setting, corresponds to security against cross-VM attacks. As we will see, however, formalizing this notion is non-trivial and several interesting subtleties arise. The intuitive security guarantee that we expect from a cloud is *isolation*, which means that each VM is executed independently of other VMs and, in particular, in such a way that cross-VM attacks are not possible. We formalize this intuition in the ideal/real-world paradigm which is typically used to analyze the security of multi-party computation protocols. Roughly

[1] We note that while this seems to be intuitively the case, we are not aware of any experimental studies that validate this intuition.

speaking, our definition will guarantee that the execution of a tenant's deployment in a cloud is equivalent to an execution of the deployment by a trusted party. And since cross-VM attacks are impossible by definition when VMs are executed by a trusted party, they are also impossible in the cloud. As we will see, isolation is a very strong property that implies security beyond just protecting the honest tenant's inputs and/or the state of his VMs. For example, as described, isolation also protects the VMs' resource vectors (i.e., the amount of resources they need) which may not always be a desirable property. This leads us to distinguish between two notions of isolation: *weak* isolation which leaks the resource vectors and *strong* isolation which protects the resource vectors.

While isolation (either strong or weak) is clearly a desirable property from a cloud, in practice, one may be satisfied with something weaker. In particular, in some settings a guarantee that the adversary cannot co-locate with more than a certain fraction of one's deployment may be enough. We formalize this property as the notion of *co-location-resistance* (CLR). Informally, we say that a placement algorithm Π is $(n, q, t, \delta, \varepsilon)$-CLR if an adversary that submits at most q attack VMs can co-locate with at least a δ fraction of a t-size deployment with probability at most ε (here n is the number of servers in the cloud). CL-resistance is interesting for several reasons. First, as discussed above, it can be a useful security guarantee in and of itself. In addition, we also show in Section 4.2 that the $(n, q, t, 1/t, \varepsilon)$-CL-resistance of a placement algorithm Π gives an upper bound on the perfect isolation provided by a cloud $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$, where *perfect* refers to the adversary being computationally unbounded. Throughout, we refer to CL-resistance with $\delta = 1/t$ as *single* CL-resistance. It turns out that designing placement algorithms with very strong *single* CL-resistance (i.e., with very small ε) is very difficult to achieve.

This motivates us to consider a weaker notion of isolation: namely, *computational* isolation where the adversary is assumed to be computationally-bounded and where tenants can use cryptography to secure their deployments. In this setting, we introduce the notion of a *shared* deployment which guarantees that no information about the tenant's input can be recovered unless *all* VMs in the deployment are compromised. We show how to construct such deployments for arbitrary functions from multi-worker delegation protocols which were introduced in [6]. We then show in Section 6 that the *computational* isolation of a *shared* deployment is upper-bounded by the $(n, q, t, 1, \varepsilon)$-CLR of a placement algorithm Π. Throughout, we will refer to CL-resistance with $\delta = 1$ as *complete* CL-resistance.

Our algorithm. Having established a model, notions of efficiency and security definitions, we turn to the problem of designing CL-resistant placement algorithms in Section 5. Our algorithm is parameterized with a value λ which influences both its CL-resistance and its competitive ratio. Intuitively, the higher λ is the better its CL-resistance becomes and the worse its competitive ratio is. We analyze our algorithm with respect to both complete CL-resistance (i.e., when $\delta = 1$) and fractional CL-resistance (i.e., when $\delta < 1$). Combining this with our results connecting CL-resistance and isolation, we then show that our algorithm performs poorly with respect to *perfect* isolation but performs well with respect to *computational* isolation when used with a shared deployment scheme.

2 Preliminaries and Model

Notation. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \overset{\$}{\leftarrow} X$ to represent an element x being sampled uniformly from a set X. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. We refer to the ith element of a vector $\mathbf{v}$ as either v_i or $\mathbf{v}[i]$. Throughout k refers to the security parameter. A function $\nu : \mathbb{N} \to \mathbb{N}$ is negligible in k if for every positive polynomial $p(\cdot)$ and sufficiently large k, $\nu(k) < 1/p(k)$. $\mathsf{poly}(k)$ and $\mathsf{negl}(k)$ denote unspecified polynomial and negligible functions in k, respectively. We write $f(k) = \mathsf{poly}(k)$ to mean that there exists a polynomial $p(\cdot)$ such that for all sufficiently large k, $f(k) \leq p(k)$, and $f(k) = \mathsf{negl}(k)$ to mean that there exists a negligible function $\nu(\cdot)$ such that for all sufficiently large k, $f(k) \leq \nu(k)$. Throughout, PPT denotes probabilistic polynomial time.

Clouds. We provide an overview of our model of cloud computing. At a high-level, we view a cloud as a set of n physical servers and a controller. The servers are *virtualized* which means that each server can execute a number of VMs subject to the amount of computational resources it has. The assignment of VMs to physical servers is handled by the controller which constantly monitors each server's resources. To use a cloud, a tenant creates and sends a set of VMs to the controller. This set of VMs—referred to as the tenant's *deployment*—is a distributed instantiation of the tenant's computation. So for example, if the tenant wishes to deploy a web service in the cloud, then its deployment might consist of a set of VMs that each run a web server. If, on the other hand, the tenant just wishes to outsource the evaluation of a function f then its deployment would consist of a single VM that executes f on some input x.

A cloud $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$ consists of a controller and a set of n virtualized physical servers $\mathsf{Srv}_1, \ldots, \mathsf{Srv}_n$. Each virtualized server Srv_i has a d-dimensional capacity vector $\mathbf{b}_i = (c_1, \ldots, c_d)$, where each dimension corresponds to a resource provided by the server (e.g., I/O, CPU, disk) and where the value $c_j \in \mathbb{N}$ is the maximum amount of a resource provided by the server. Throughout we consider, without loss of generality, the case where all capacities are equal; that is, $c_i = c$ for all $1 \leq i \leq d$.

Virtual machines and deployments. A cloud executes VMs that are created and sent to the controller. Given a VM, the controller executes a placement algorithm Π to decide on which physical server to execute the VM. A virtual machine $\mathrm{vm} = (\mathcal{F}, x, \mathbf{r})$ consists of a functionality $\mathcal{F}$, an initial input x and a resource vector $\mathbf{r}$. The resource vector $\mathbf{r}$ is a d-dimensional vector over $[0, 1]$, where r_i is the amount of the ith resource needed by the VM.

To generate VMs for a cloud, a tenant runs a deployment scheme Δ that takes as input a functionality $\mathcal{F}$, an initial input x, a deployment size t and a set of resource vector $(\mathbf{r}_1, \ldots, \mathbf{r}_t)$. It returns a t-size deployment $\mathbf{vm} = (\mathrm{vm}_1, \ldots, \mathrm{vm}_t)$. For ease of exposition, we describe all of our results for *non-reactive* functionalities; that is, functionalities $\mathcal{F}$ that consist of a single function and no state.

3 Isolation

We now describe our main notion of security. In our setting, we are concerned with an adversarial tenant that submits malicious VMs to the cloud. The purpose of these VMs is to execute cross-VM attacks on honest VMs that are co-located on its server. Throughout this work, we assume that if a malicious VM is placed on a server Srv_i, then it can corrupt the entire server and learn the state and input of all the VMs on that server. This captures a worst-case scenario where malicious VMs can execute cross-VM attacks with certainty. Clearly, this can be relaxed and one could consider cases where malicious VMs succeed in their cross-VM attacks with some probability.

Intuitively, we say that a deployment scheme Δ is *isolated* in a cloud $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$ if, with high enough probability, a malicious tenant cannot recover any partial information about the tenants' inputs (which includes non-initial inputs in the case of a reactive deployment). We formalize this in the ideal/real-world paradigm typically used to define the security of secure multi-party computation protocols [3]. All parties below are assumed to be interactive Turing Machines.

Real-world execution. The real-world execution is a probabilistic experiment in which an honest tenant uses a cloud $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$ in the presence of an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ to compute a function f. At the beginning of the experiment, the tenant receives an input x, a deployment size t and a set of resource vectors $\mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_t)$ from the environment $\mathcal{Z}$. It also receives the auxiliary string z. The tenant runs $\Delta(f, x, t, \mathbf{R})$ to generate a deployment $\mathbf{vm} = (\mathrm{vm}_1, \ldots, \mathrm{vm}_t)$ which it submits to the controller. The adversary is given the size t of the deployment and the auxiliary string z and sends a set of attack VMs $(\mathbf{am}_1, \ldots, \mathbf{am}_q)$ and an execution schedule to the controller. The execution schedule is the order in which all VMs should be processed. The controller processes the honest and attack VMs according to $\mathcal{A}$'s execution schedule and assigns each one to a physical server according to Π. Once the VMs start running on their assigned servers, both the tenant and adversary are allowed to interact with their VMs.

Whenever an attack VM is placed on a physical server the adversary is allowed to corrupt all the VMs on that server. If $\mathcal{A}$ is semi-honest, then he sees the state and initial input to the VMs but cannot interfere with its processing. If, on the other hand, $\mathcal{A}$ is malicious, then it can also interfere with the VMs arbitrarily (we only consider semi-honest adversaries in this work but define security in the malicious model for completeness). At the end of the execution the tenant sends its output to the environment $\mathcal{Z}$ while $\mathcal{A}$ sends an arbitrary function of its view. $\mathcal{Z}$ then outputs a bit. We denote the random variable that outputs this bit by $\mathbf{Real}_{\Delta,\Pi,\mathcal{Z},\mathcal{A}}(n, q, t)$.

Ideal-world execution. The ideal-world execution is a probabilistic experiment in which the honest tenant, the cloud $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$ and the ideal adversary $\mathcal{S}$, all have access to a trusted party. At the beginning of the execution, the tenant receives an input x, a deployment size t and a set of resource vectors $\mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_t)$ from the environment $\mathcal{Z}$. It also receives the auxiliary string z. The tenant runs $\Delta(f, x, t, \mathbf{R})$ to generate a deployment $\mathbf{vm} = (\mathrm{vm}_1, \ldots, \mathrm{vm}_t)$ which it submits to the trusted party. The attacker is given the size t of the honest deployment together with the resource vectors $\mathbf{R}$ and the auxiliary string z. The adversary sends a set of attack VMs $(\mathbf{am}_1, \ldots, \mathbf{am}_q)$ to the trusted party which executes all the VMs. During these executions, the honest tenant and adversary are allowed to interact with their VMs.

At the end of the experiment, the tenant sends its output to the environment $\mathcal{Z}$ while $\mathcal{S}$ sends an arbitrary function of its view. The environment $\mathcal{Z}$ outputs a bit. We denote the random variable that outputs this bit by $\mathbf{Ideal}_{\Delta,\Pi,\mathcal{Z},\mathcal{S}}(n, q, t)$.

Definition 3.1 (Isolation). *A deployment scheme Δ is (n, q, t, ε)-isolated in a cloud $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$ if for all functions f, for all* PPT *adversaries $\mathcal{A}$, there exists an ideal* PPT *adversary $\mathcal{S}$ such that for all* PPT *environments $\mathcal{Z}$ and all auxiliary strings $z \in \{0,1\}^*$, the following expression is at most ε*

$$\left|\Pr\left[\,\mathbf{Real}_{\Delta,\Pi,\mathcal{Z},\mathcal{A}}(n, q, t) = 1\,\right] - \Pr\left[\,\mathbf{Ideal}_{\Delta,\Pi,\mathcal{Z},\mathcal{S}}(n, q, t) = 1\,\right]\right|,$$

where $\varepsilon = \mathsf{negl}(k)$. If $\mathcal{A}$, $\mathcal{S}$ and $\mathcal{Z}$ are unbounded, then we say that Δ is perfectly *(n, q, t, ε)-isolated in* Cld. *When n, q and t are clear from the context, we sometimes say that Δ is ε-isolated in* Cld.

Intuitively, the definition guarantees that anything an adversary can achieve in the real-world execution can be achieved in the ideal-world execution, where the VMs are isolated by definition since they are executed by a trusted party. Note, however, that in the ideal-world execution the adversary is given the resource vectors of the honest deployment which implies that the definition allows the adversary to learn this information in real-world executions.

Strong isolation. A stronger definition, which we refer to as *strong* isolation, results from not giving the adversary the resource vectors in the ideal-world execution. This distinction between weak and strong isolation highlights that it could be possible to protect the honest tenant's inputs without necessarily hiding the resource vectors of its deployment (of course this is provided the resource vectors are independent of the input). The resource vectors, however, could also be sensitive in some scenarios so we believe the notion of strong isolation is interesting as well.

Relationship to co-location-resistance. A simple but crucial observation about isolation and cross-VM attacks is that the latter require the completion of a *co-location attack*, i.e., the adversary first needs to co-locate its malicious VMs with the target's deployment. Since in infrastructure clouds VM placement is not under adversarial control, the adversary must create and launch its attack VMs in such a way as to influence the controller into co-locating them with the target's deployment. Concrete examples of such attacks were first described by Ristenpart et al. against Amazon EC2 in [15]. In that work, the authors describe two types of co-location attacks. The first are what we refer to as *complete* attacks, where the adversary's goal is to co-locate at least one attack VM with each VM from a target set (not necessarily all from the same deployment). The second are what we refer to as *fractional* attacks where the adversary just wants to co-locate with some fraction of a target set (again, not necessarily from the same deployment).

In Section 4, we formalize the notion of co-location resistance and in Sections 4 and 6, respetively, we show that the various forms of co-location-resistance imply either perfect or computational isolation (the latter when used in conjunction with a shared deployment). Establishing relationships between these notions serves several purposes. The first—as is usually the case in cryptography—is that it increases our confidence in the definitions. The second is that it allows us to use the notion of co-location-resistance which is easier to work with when analyzing secure placement algorithms (this is similar to the situation of game-based vs. simulation-based definitions in Cryptography). The third is that both notions—and their variants—may each prove useful in practice depending on the setting.

4 Secure Placement Algorithms

Here, we are concerned with the design and analysis of "secure" placement algorithms; that is, algorithms that place VMs in such a way that it is "hard" for the adversary to co-locate attack VMs with target VMs. To formally analyze the security of such algorithms we introduce and formalize the notion of *co-location resistance* which captures the intuitive notion of hardness above. In Theorem 4.3 in Section 4.2 below, we show that a certain variant of CL-resistance implies *weak* isolation.

Though we provide a cryptographic treatment of CL-resistance, it is important to note that our notion of security is fundamentally different than standard cryptographic notions. The security of standard cryptographic algorithms is typically quantified by the time-to-success ratio, i.e., the worst-case (over all adversaries within some class) ratio of the time invested by the adversary and its probability of breaking the algorithm. This is typically reflected, in the asymptotic setting, by the requirement that all polynomially-bounded adversaries be able to break the algorithm with probability at most $\mathsf{negl}(k) = k^{-\omega(1)}$, where k is the security parameter. In our setting, we are interested instead in an adversary's success probability as a function of the number of attack VMs it uses.

4.1 Competitive Ratio

In infrastructure clouds such as Amazon EC2 or Microsoft Azure, VMs are assigned to servers according to carefully designed placement algorithms. To minimize the amount of resources used, these algorithms typically try to pack as many VMs as possible on each server. Viewing each VM as a d-dimensional vector over $[0, 1]$ (with dimensions corresponding to some computational resource) and each server as a d-dimensional vector over $[0, c]$ (with c being the maximum amount of a resource made available by the server), the problem of VM assignment becomes an instance of the online vector bin packing problem. We refer the reader to [13, 8] for detailed discussions on the relationship between VM placement and the bin packing problem.

Bin packing. In the vector bin packing problem, we are given a sequence of d-dimensional vectors $(\mathbf{v}_1, \ldots, \mathbf{v}_n)$ over $[0, 1]$ and a set of n bins $(\mathbf{b}_1, \ldots, \mathbf{b}_m)$ each of capacity c (i.e., each $\mathbf{b}_i \in [0, c]^d$) and must assign the vectors to bins subject to never exceeding the bin capacity in any dimension. In the online version, we receive the sequence of vectors as a stream and must assign each $\mathbf{v}_i$ before seeing $\mathbf{v}_{i+1}$. Bin packing is a classic and well-studied problem in computer science. The offline vector bin packing problem is **NP**-hard. While there exists an asymptotic polynomial-time approximation scheme (PTAS) [2] for the one-dimensional case, it is not practical as the constants are too high. For dimension $d \geq 2$, the problem is APX-hard [24] which means that no PTAS exists unless $\mathbf{P} = \mathbf{NP}$.

Competitive ratio. The quality of an online optimization algorithm Π is evaluated using the notion of *competitive ratio* [18] which is the worst-case (over input sequences) ratio between the online performance of Π and the offline performance of the optimal algorithm. The performance of the algorithm is quantified using a *cost function* which is chosen to reflect the real-world cost of the algorithm. In the case of bin packing the cost function is typically chosen to be the total number of bins used. The online bin packing problem can be solved using the generalized first-fit algorithm [5] which is $O(d)$-competitive. In light of a recent lower bound of $\Omega(d^{1-\varepsilon})$ by Azar et al. [2] this is effectively optimal.

A new cost function. Like any online algorithm, the quality of CL-resistant placement algorithms should be evaluated using the competitive ratio. There are subtleties, however, in exactly how this should be done. In particular, we note that the standard cost function used for bin packing is not appropriate in our setting. The reason is that in the context of VM assignments the true cost of a packing algorithm is the total amount of resources expended by the servers/bins. While total resource consumption is correlated with the number of bins used in the *offline* setting, this is not true in the online case.

To see why, consider the following two simple algorithms. The first algorithm, Π_1, turns on one server per hour for a duration of n hours while the second algorithm, Π_2, turns on n servers in the first hour and keeps all servers on for n hours. Note that both algorithms use n servers but obviously the second algorithm is less efficient with respect to total resource consumption and, in particular, to energy.

This example illustrates the fact that we need a new cost function in order to evaluate the quality of a placement algorithm in the *online* setting. We describe such a function precisely in Definition 4.1 below. Intuitively, our new cost function, which we refer to as the server-time, works as follows. We partition the execution of a placement algorithm Π into T time steps. At Step 1, all the servers are assumed to be off. We also assume that once a VM is placed on a server, it runs perpetually. At each step, Π either receives a VM or not. If so, it assigns the VM to a server, choosing to place it either on a server that was already on or on a server that was previously off (after turning it on, of course). The cost of Π is defined as the sum over time of the number of servers that are on at each time step. Returning to our example above, note that under this new function, Π_1 has cost $n \cdot (n + 1)/2$ whereas Π_2 has cost n^2.

Definition 4.1 (Competitive ratio). *Consider the following probabilistic experiment between an adversary $\mathcal{A}$ and a challenger that places m VMs on n servers using an algorithm Π:*

[2]A PTAS is an algorithm that takes as input an instance of the problem and a value $\varepsilon > 0$ and is guaranteed to return a solution that is within a $(1 + \varepsilon)$ factor of optimal.

- $\mathbf{Cost}_{\Pi,\mathcal{A}}(n, m, \mathrm{T})$*: Let* $\mathrm{ON}_0 = \emptyset$ *and, for each time step* $1 \leq i \leq \mathrm{T}$*, set* $\mathrm{ON}_i = \mathrm{ON}_{i-1}$ *and proceed as follows. If the adversary* $\mathcal{A}$ *outputs a virtual machine* vm_i*, then the challenger places* vm_i *according to* Π*. If* Π *placed* vm_i *on a server that was previously off then add that server to* ON_i*. The experiment outputs* $\sum_{i=1}^{\mathrm{T}} |\mathrm{ON}_i|$*.*

We say that Π *is* $(n, m, \mathrm{T}, \delta)$*-competitive if for all (unbounded) adversaries* $\mathcal{A}$*,*

$$\mathbf{Cost}_{\Pi,\mathcal{A}}(n, m, \mathrm{T}) \leq \delta \cdot \min_{\Pi^\star} \left\{ \mathbf{Cost}_{\Pi^\star,\mathcal{A}}(n, m, \mathrm{T}) \right\}.$$

4.2 Co-Location Resistance

In a co-location attack, the adversary seeks to co-locate attack VMs with some fraction of target VMs. More precisely, given t target VMs $(\mathbf{tm}_1, \ldots, \mathbf{tm}_t)$ the adversary launches q attack VMs $(\mathbf{am}_1, \ldots, \mathbf{am}_q)$ so that a δ fraction of the target VMs are co-located with at least one attack VM. Intuitively, we say that a placement algorithm Π is CL-resistant if the probability of this occurring is bounded by some small value ε. We formalize this intuition as a probabilistic experiment between an adversary $\mathcal{A}$ and a cloud $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$ with a controller that runs the placement algorithm Π.

Definition 4.2 (Co-location-resistance)**.** *Consider the following probabilistic experiment between an adversary* $\mathcal{A}$ *and a cloud* $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$*:*

- $\mathbf{CLR}_{\Pi,\mathcal{A}}(n, q, t, \delta)$*: the adversary* $\mathcal{A}$ *generates* q *attack machines* $(\mathbf{am}_1, \ldots, \mathbf{am}_q)$*,* t *target machines* $(\mathbf{tm}_1, \ldots, \mathbf{tm}_t)$ *and an execution schedule. The controller processes all the VMs according to the adversary's execution schedule and places them according to* Π*. The experiment outputs* 1 *if at least* $\delta \cdot t$ *target VMs are co-located with at least one of the* q *attack VMs and* 0 *otherwise.*

We say that Π *is* $(n, q, t, \delta, \varepsilon)$*-CL-resistant if for all adversaries* $\mathcal{A}$*,*

$$\Pr\left[\, \mathbf{CLR}_{\Pi,\mathcal{A}}(n, q, t, \delta) = 1 \,\right] \leq \varepsilon.$$

When n*,* q *and* t *are clear from the context, we sometimes say that* Π *is* (δ, ε)*-CLR.*

A few words about our definition are in order. First, note that the adversary $\mathcal{A}$ can choose both the attack VMs and the target VMs. This captures the worst-case scenario where a client chooses its VMs according to the instructions of the adversary. The implication is that if one can show that an algorithm Π is secure according to this definition, an adversary will have at most an ε probability of co-location *even in the worst-case.* Another note about the definition is that the adversary can schedule attack VMs before target VMs. This captures the worst-case scenario with respect to the distribution of VMs on the servers right before the target VMs are placed.

Alternative definitions. Our notion of CL-resistance is very general and captures several scenarios of interest depending on the parameter ranges. For instance, setting $\delta = 1$ captures what we refer to as *complete* CL-resistance, where the adversary wishes to co-locate attack VMs with *all* the target VMs. Similarly, by setting $\delta = 1/t$, we get the notion of *single* CL-resistance where the adversary is only interested in co-locating with at least 1 of the t target VMs. We refer to the case where $0 < \delta < 1$, as *fractional* CL-resistance.

The range of t is also important as it determines the types of deployments that can be handled. For example, for single VM deployments we are only interested in the setting $t = 1$ and $\delta = 1$.

In other cases, where the honest tenant deploys several VMs (e.g., for cluster computing or for a load-balanced web server) we are interested in $t > 1$ and $\delta \geq 1/t$.

Relationship to isolation. We now show that single CL-resistance implies weak isolation. The intuition is straightforward: if Π is $(1/t, \varepsilon)$-CLR then with probability at least $1 - \varepsilon$ none of the attack VMs will be co-located with the honest deployment and therefore the adversary will not learn any information about the honest tenant's inputs. We point out, however, that this does not imply *strong* isolation. In fact, for our proof to go through we explicitly need the simulator to receive the resource vectors of the honest deployment. This suggests that even if the placement algorithm is CL-resistant, the adversary could still learn information about the deployment's resource vectors. Intuitively, this makes sense since CL-resistance only guarantees that, with some probability, the attack VMs will not be co-located with some fraction of the deployment which a-priori does not imply any privacy for the resource vectors. To see why, consider the case of complete CL-resistance, from which we can get a lower bound on the probability that *none* of the attack VMs are co-located with the deployment. Clearly, this is the best possible case from the perspective of protecting the honest tenant's inputs. In such a scenario, the only co-locations that occur are between attack VMs so the adversary only learns his own inputs from a cross-VM attack. Note, however, that even from this seemingly innocuous information he can infer something about the current placement of VMs in the cloud. [3] For example, it will learn that a particular server is executing a specific set of his attack VMs. The placement of VMs on servers, however, is a function of the resources required by *both* the honest and malicious VMs so learning anything about the current placement of VMs in the cloud can reveal information about the honest VMs resource vectors.

Theorem 4.3. *Let Δ be a deployment scheme and Π be an $(n, q, t, 1/t, \varepsilon)$-CL-resistant placement algorithm. Then Δ is (n, q, t, ε)-isolated in $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$.*

Proof. Let $\mathcal{A}$ be an arbitrary real-world adversary and consider the ideal adversary $\mathcal{S}$ that works as follows. Given the resource vectors of the honest deployment, $\mathcal{S}$ simulates $\mathcal{A}$ by feeding it the auxiliary string z. When $\mathcal{A}$ returns a set of q attack VMs and an execution schedule, $\mathcal{S}$ sends the attack VMs to the trusted party. $\mathcal{S}$ then creates a dummy target deployment $(\mathbf{tm}_1, \ldots, \mathbf{tm}_t)$ that consists of VMs with the same resource vectors as the honest deployment but that execute arbitrary functions over arbitrary inputs. It then simulates Ctrlr_Π, feeding it the dummy deployment and the attack VMs according to $\mathcal{A}$'s execution schedule. Whenever an attack VM is co-located with another attack VM, $\mathcal{S}$ sends their contents to $\mathcal{A}$. Throughout, $\mathcal{S}$ forwards any messages between attack VMs and $\mathcal{A}$. If at any point during this simulation, Ctrlr_Π co-locates an attack VM with a dummy VM, $\mathcal{S}$ aborts. At the end of the simulation, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Let $\mathcal{E}$ be the event that no target VMs are co-located with attack VMs. Conditioned on $\mathcal{E}$, it follows by construction that $\mathcal{A}$'s simulated view is distributed exactly as in a $\mathbf{Real}_{\Delta,\Pi,\mathcal{Z},\mathcal{A}}(n, q, t)$ execution and therefore so is its output. Notice also that by construction Ctrlr_Π's view is distributed exactly as in a $\mathbf{CLR}_{\Pi,\mathcal{A}}(n, q, t, 1/t)$ experiment so it follows by the CL-resistance of Π that the event $\mathcal{E}$ occurs with probability at least $1 - \varepsilon$. Putting the two together, it follows that Cld is ε-isolated. ∎

5 Our Placement Algorithm

We now describe and analyze our placement algorithm. The algorithm is very simple and is described in detail in Fig. 1. It is a randomized *stateful* vector bin packing algorithm that is parameterized

[3] This is assuming the adversary can distinguish between his own attack VMs, which can be done trivially

Let Srv_1 through Srv_n be n servers such that $\mathsf{Srv}_i \in [0,c]^d$, where $d \geq 1$ is the dimension and $c \geq 2$ is the capacity of the servers. Let $\lambda \geq 2$ be a parameter of the algorithm.

```
Algorithm $\Pi_\lambda(\mathrm{vm})$:

  1. Choose a subset of $\lambda$ open servers $\textsc{Opn} \subseteq \{\mathsf{Srv}_1, \ldots, \mathsf{Srv}_n\}$;
  2. Set $\textsc{Emp} = \{\mathsf{Srv}_1, \ldots, \mathsf{Srv}_n\}/\textsc{Opn}$.
  3. Repeat
     (a) choose a server $\mathsf{Srv}$ from $\textsc{Opn}$ uniformly at random and let $\boldsymbol{\ell} \in [0,c]^d$ be its current load;
     (b) assign vm to $\mathsf{Srv}$;
     (c) if $\boldsymbol{\ell}[i] + \mathrm{vm}[i] > c - 1$ for at least one $i \in [d]$,
         i. remove $\mathsf{Srv}$ from $\textsc{Opn}$ and add it to $\textsc{Cl}$;
         ii. remove an empty server from $\textsc{Emp}$ and add it to $\textsc{Opn}$
```

Figure 1: A CL-resistant placement algorithm

by a value $\lambda \geq 2$. It takes as input a vector $\mathrm{vm} \in [0,1]^d$ and outputs a server identifier $i \in [n]$.

Here, we assume the server capacity c is at least 2. This is in order to remove a dependence on c in the competitive ratio. Since VM resource vectors are in $[0,1]^d$, this is equivalent to assuming that no VM will use up more than 1/2 of a resource available on a server.

At a high level, the algorithm works as follows. It labels servers as either off or on and open or closed. Off and on refer to whether the server is powered on (i.e., consuming energy) and open and closed refer to whether the server can receive more VMs or not. Note that a closed server can be on. When referring to closed servers, we do not count servers that are off. The algorithm will always keep exactly λ servers open and assign an incoming VM to one of these servers Srv uniformly at random. If any of Srv's resources increases to being larger than $c - 1$ due to the new VM (where c is the capacity of Srv) then Srv is closed and a new server is opened. Note that VM resource vectors are in $[0,1]^d$ so the algorithm never assigns VMs to a server that does not have the required capacity.

In the following Sections we analyze our algorithm's competitive ratio and its security against complete and fractional CL attacks.

5.1 Competitive Ratio

We now prove that our algorithm is a good vector bin packing algorithm in the sense that its competitive ratio is a factor of $\lambda + 2d$ off from optimal, where d is the number of resources provided by the servers and λ is the number of servers the algorithm keeps open at all times.

Theorem 5.1. *The algorithm Π_λ from Fig. 1 is (n, T, δ)-competitive, where $\delta = \lambda + 2d$.*

Proof. At each time step i, the total resources needed by $(\mathrm{vm}_1, \ldots, \mathrm{vm}_i)$ is

$$R_i = \sum_{j=1}^{i} \sum_{l=1}^{d} \mathrm{vm}_j[l],$$

so the number of servers needed at step i by any algorithm is at least $R_i/(d \cdot c)$. In the following we let $\textsc{On}_i^\star$ denote the set of servers opened by an optimal algorithm for the instance; hence

$$|\textsc{On}_i^\star| \geq R_i/(d \cdot c). \qquad (1)$$

Recall that we wish to bound

$$\delta = \frac{\sum_{i=1}^{\mathrm{T}} |\textsc{On}_i|}{\sum_{i=1}^{\mathrm{T}} |\textsc{On}_i^\star|}. \tag{2}$$

Let $\textsc{Opn}_i$ and $\textsc{Cl}_i$ be the open and closed servers at time i by our algorithm, and $\boldsymbol{\ell}(\textsc{Opn}_i)$ and $\boldsymbol{\ell}(\textsc{Cl}_i)$ denote the load (i.e., total resources) on the open and closed servers at time i, respectively. Note that by construction each of the closed servers has at least one of its dimensions filled up to $c-1$ and therefore $\boldsymbol{\ell}(\textsc{Cl}_i) \geq |\textsc{Cl}_i| \cdot (c-1)$. It follows that

$$\boldsymbol{\ell}(\textsc{Opn}_i) = R_i - \boldsymbol{\ell}(\textsc{Cl}_i) < R_i - |\textsc{Cl}_i| \cdot (c-1),$$

from which we have that

$$R_i > \boldsymbol{\ell}(\textsc{Opn}_i) + |\textsc{Cl}_i| \cdot (c-1) > |\textsc{Cl}_i| \cdot (c-1). \tag{3}$$

Note next that

$$\sum_{i=1}^{\mathrm{T}} |\textsc{On}_i| = \sum_{i=1}^{\mathrm{T}} |\textsc{Opn}_i + \textsc{Cl}_i| = \mathrm{T} \cdot \lambda + \sum_{i=1}^{\mathrm{T}} |\textsc{Cl}_i|,$$

where the second equality is because $|\textsc{Opn}_i| = \lambda$ for all i since Π_λ always keeps λ servers open by construction. Hence by Eq. (2) we have

$$\delta = \frac{\mathrm{T} \cdot \lambda + \sum_{i=1}^{\mathrm{T}} |\textsc{Cl}_i|}{\sum_{i=1}^{\mathrm{T}} |\textsc{On}_i^\star|} \leq \lambda + \frac{\sum_{i=1}^{\mathrm{T}} |\textsc{Cl}_i|}{\sum_{i=1}^{\mathrm{T}} |\textsc{On}_i^\star|}, \tag{4}$$

where the second inequality follows from the fact that $\sum_i |\textsc{On}_i^\star| \geq \mathrm{T}$. We now use Eqs. (1) and (3) to upper bound the second term in Eq. (4), resulting in

$$\frac{\sum_{i=1}^{\mathrm{T}} |\textsc{Cl}_i|}{\sum_{i=1}^{\mathrm{T}} |\textsc{On}_i^\star|} \leq \frac{\sum_{i=1}^{\mathrm{T}} |\textsc{Cl}_i|}{\sum_{i=1}^{\mathrm{T}} R_i/(d \cdot c)} \leq \frac{d \cdot c}{c-1} \cdot \frac{\sum_{i=1}^{\mathrm{T}} |\textsc{Cl}_i|}{\sum_{i=1}^{\mathrm{T}} |\textsc{Cl}_i|} \leq 2d,$$

where the final inequality follows since $c \geq 2$ and hence $c/(c-1) \leq 2$.

■

5.2 Complete Co-Location Resistance

Here, we analyze the complete CL-resistance of our algorithm (i.e., for $\delta = 1$). At a high-level, our analysis is based on a balls and bins process and proceeds in three steps. First, we show in Lemma 5.2 a bound on the probability that each bin among a given subset of bins contains at least one ball. As far as we know—and perhaps surprisingly—no concrete analysis of this particular balls and bins process has appeared prior to this work.

We note that while the process can be viewed as a coupon collector problem where the bins are the coupons, we cannot apply the Erdos-Renyi Theorem [4] since it is asymptotic and only holds for infinitely many bins. We therefore believe that Lemma 5.2 is of independent interest. Intuitively, Lemma 5.2 will help us bound the probability that—conditioned on the target VMs being placed on some number of servers—our algorithm will co-locate each target VM with at least one attack VM.

In the second phase of our analysis we prove in Lemma 5.3 a lower bound on the probability that at least $t/2$ bins are occupied when throwing t balls in b bins uniformly at random. Intuitively, this Lemma will allow us to analyze the number of servers on which our algorithm places the target VMs. Finally, in Theorem 5.4 we combine both Lemmas to analyze our algorithm's co-location-resistance.

Lemma 5.2. *Consider a process that throws balls uniformly at random into b bins and let C be a subset of these bins. If the process throws $q \le b \cdot \log |C|$ balls, then the probability that each bin in C has at least one ball is at most*

$$1/\exp\left(\gamma \cdot \left(\left(1 - \frac{q}{b \cdot \log |C|}\right) \cdot \log |C|\right)^2\right),$$

if $|C| \ge 2$, where γ is some constant strictly less than 1. If $|C| = 1$, then the probability is at most

$$1 - \left(\frac{1}{4}\right)^{q/b}.$$

Proof. This process can be viewed as a variant of the coupon collector problem where the bins in C are the coupons. Let epoch i denote the set of samples between the time where we found the ith and $(i+1)$th new coupons. So epoch 0, for example, is the set of samples made until a ball is first placed in a bin in C. Let X_i be the length of epoch i (i.e., the number of samples needed to collect the $(i+1)$th coupon once we have the ith) and let $X = \sum_{i=0}^{c-1} X_i$ be the total number of samples needed to find all $|C|$ coupons.

Clearly, if $|C| = 1$,

$$\Pr\left[\, X \le q \,\right] = 1 - \left(1 - \frac{1}{b}\right)^q \le 1 - \left(\frac{1}{4}\right)^{q/b},$$

so we focus on the case $|C| \ge 2$. Since the X_i are non-negative independent random variables, it follows by [12] that

$$\Pr\left[\, \mathbb{E}[X] - X \ge a \,\right] \le 1/\exp\left(\frac{a^2}{2 \cdot \sum_i \mathbb{E}[X_i^2]}\right).$$

Setting $a = \alpha \cdot \mathbb{E}[X]$ we have,

$$\Pr\left[\, X \le (1-\alpha) \cdot \mathbb{E}[X] \,\right] \le 1/\exp\left(\frac{\alpha^2 \cdot \mathbb{E}[X]^2}{2 \cdot \sum_i \mathbb{E}[X_i^2]}\right). \tag{5}$$

Note that X_i satisfies a geometric distribution with parameter $p_i = (|C| - i)/b$ since, in epoch i, the probability of finding a new coupon is $(|C| - i)/b$. It follows that $\mathbb{E}[X_i] = 1/p_i = b/(|C| - i)$ and therefore that

$$\mathbb{E}[X] = \sum_{i=0}^{|C|-1} \mathbb{E}[X_i] \ge b \cdot \sum_{i=0}^{|C|-1} \frac{1}{|C| - i} \ge b \cdot \log |C|, \tag{6}$$

where the first equality is by the linearity of expectation. Since X_i is geometric, we also have $\mathsf{Var}[X_i] = (1 - p_i)/p_i^2$ and

$$\mathbb{E}[X_i^2] = \mathbb{E}^2[X_i] + \mathsf{Var}[X_i] = \frac{2 - p_i}{p_i^2} \le \frac{2b^2}{(|C| - i)^2}.$$

From the above, it follows that

$$\sum_{i=0}^{|C|-1} \mathbb{E}[X_i^2] \le \sum_i \frac{2b^2}{(|C| - i)^2} \le \beta \cdot b^2, \tag{7}$$

where β is some constant which is about $12/\pi^2$. Combining this with (5) and (6), we obtain

$$\begin{aligned}
\Pr[\, X \leq (1-\alpha)\cdot \mathbb{E}[X]\,] &\leq 1/\exp\left(\frac{\alpha^2 \cdot \mathbb{E}[X]^2}{2\beta \cdot b^2}\right) \\
&\leq 1/\exp\left(\frac{\alpha^2 \cdot b^2 \log^2 |C|}{2\beta \cdot b^2}\right) \\
&= 1/\exp\left(\frac{\alpha^2 \cdot \log^2 |C|}{2\beta}\right)
\end{aligned}$$

Setting $q = (1-\alpha)\cdot \mathbb{E}[X]$, i.e., $\alpha = 1 - q/\mathbb{E}[X]$, we have

$$\Pr[\, X \leq q\,] = 1/\exp\left(\gamma \cdot \left(\left(1 - \frac{q}{b\cdot \log|C|}\right)\cdot \log|C|\right)^2\right),$$

where $\gamma = 1/(2\cdot\beta) \approx \pi^2/24$ as long as $|C| \geq 2$.

■

Lemma 5.2 upper-bounds the probability that each bin in C holds at least one of q balls. As will become clear in the proof of Theorem 5.4 this Lemma allows us to upper bound the probability that our algorithm will co-locate at least one attack VM with each one of t target VMs. The bound, however, depends on the size of the subset C which, intuitively, will correspond to the number of servers on which the t target VMs are located. In the following Lemma we upper bound the probability that less than $t/2$ bins will be occupied when throwing t balls, or in other words, the probability that our algorithm places the t target VMs on less than $t/2$ servers.

Lemma 5.3. *Consider a process that throws t balls into b bins uniformly at random. If $t \leq b/e$, then the probability that there are at most $t/2$ occupied bins is at most $2^{-t/2}$.*

Proof. The probability that at most m bins are occupied is at most

$$\binom{b}{m}\cdot\left(\frac{m}{b}\right)^t \leq \left(\frac{eb}{m}\right)^m \cdot \left(\frac{m}{b}\right)^t = e^m \cdot \left(\frac{m}{b}\right)^{t-m}.$$

Setting $m = t/2$, we get $m/b \leq 1/2e$ and the above expression is at most

$$e^{t/2}\cdot\left(\frac{1}{2e}\right)^{t/2} = \frac{1}{2^{t/2}}.$$

■

Our main Theorem. We now turn to our CL-resistance analysis and Theorem 5.4. We discuss our Theorem and its implications below.

Theorem 5.4. *If $t \leq \lambda/e$ and $q < \lambda \cdot \log(t/2)$ then the algorithm Π_λ, described in Fig. 1, is $(n, q, t, 1, \varepsilon)$-CL-resistant for*

$$\varepsilon = 2^{-t/2} + 1/\exp\left(\gamma\cdot\left(\left(1 - \frac{q}{\lambda\cdot\log(t/2)}\right)\cdot\log(t/2)\right)^2\right),$$

for some constant $\gamma < 1$.

Proof. Let $\mathcal{A}$ be an arbitrary adversary and consider a $\mathbf{CLR}_{\Pi_\lambda,\mathcal{A}}(n,q,t)$ experiment. In the following we omit the subscripts Π_λ and $\mathcal{A}$ and the parameters (n, q_1, q_2, t) for visual clarity. Let S be the random variable denoting the number of servers on which the target VMs are located. Recall that Π_λ assigns attack VMs uniformly at random amongst a set of λ open servers. The VM assignments can therefore be viewed as a balls and bins process. We then have,

$$\begin{aligned}\Pr[\,\mathbf{CLR}=1\,] &= \Pr[\,\mathbf{CLR}=1 \mid S < t/2\,]\cdot\Pr[\,S<t/2\,]\\ &\quad + \Pr[\,\mathbf{CLR}=1 \mid S \geq t/2\,]\cdot\Pr[\,S\geq t/2\,]\\ &\leq 2^{-t/2} + \Pr[\,\mathbf{CLR}=1 \mid S\geq t/2\,],\end{aligned}$$

where the inequality follows from upper bounding the first and fourth terms in the first equality by 1 and upper bounding the second term using Lemma 5.3.

The probability that $\mathbf{CLR} = 1$ conditioned on $S \geq t/2$, can be bounded using Lemma 5.2 by setting $b = \lambda$ and $|C| = i$ for $t/2 \leq i \leq t$. Note, however, that this probability is maximized when $S = t/2$ so we have,

$$\begin{aligned}\Pr[\,\mathbf{CLR}=1\,] &\leq 2^{-t/2} + \Pr[\,\mathbf{CLR}=1 \mid S = t/2\,]\\ &\leq 2^{-t/2} + 1/\exp\left(\gamma\cdot\left(\left(1-\frac{q}{\lambda\cdot\log(t/2)}\right)\cdot\log(t/2)\right)^2\right).\end{aligned}$$

■

Remarks. Note that the bound on CL-resistance we establish holds for a certain range of parameters. In particular, as long as the number of target VMs t is at most λ/e and the number of attack VMs q is less than $\lambda\cdot\log(t/2)$. In this regime, our algorithm achieves a good complete CL-resistance for large t.

In large public infrastructure clouds, the number of servers n can be very large. For example, Amazon EC2 is rumored to have anywhere from $150K$ [19] to $450K$ [10] servers. Taking the smallest estimates of $n = 150K$ and assuming that a cloud provider is willing to have at least 1/3 of its servers on at any time, we could set $\lambda = 50K$ in our algorithm. Our analysis would then hold for deployments of size at most 18,394 and adversaries with a budget of at most 456,331 attack VMs, both of which are very large. The tradeoff in efficiency is that the placement would be $50K + 2d$ off of the optimal placement in terms of server-time.

On the other end of the spectrum, assuming the cloud provider is willing to have less than 1% of its servers on at any time (say 0.5%) we could set $\lambda = 1K$ in our algorithm. Our analysis would then hold for deployments of size at most 3678 and adversaries with a budget of at most 3912 attack VMs. Here, the tradeoff in efficiency would be $1K + 2d$ off of the optimal placement in terms of server-time. In this regime, where $\lambda = 1K$, Theorem 5.4 bounds the complete CL-resistance of the algorithm by 0.009 for deployments of size $t = 100$ against an adversary with $1K$ attack VMs. For even smaller λ, say $\lambda = 100$, the analysis would hold for deployments of size up to about 18 and adversaries with budgets of up to 290 attack VMs with efficiency that is $100 + 2d$ off from optimal.

Another important observation about our bound is that it is not tight. This is due to our analysis overestimating the probability of co-location by ignoring the fact that target VMs on closed servers cannot be co-located with new attack VMs. This indicates that, in practice, our algorithm will likely perform better than what our analysis suggests.

5.3 Fractional Co-Location Resistance

In this Section we analyze the fractional CL-resistance of our algorithm (i.e., for $\delta < 1$). The overall structure of our analysis is similar to the targeted case: we first show in Lemma 5.5 a bound on the probability that a δ fraction of a given set of bins is occupied when throwing q balls. In Theorem 5.6 below, we then combine Lemma 5.5 and Lemma 5.3 to prove our main bound.

Lemma 5.5. *Consider a process that throws balls uniformly at random into b bins and let C be a subset of these bins. If the process throws q balls, then the probability that at least $\delta \cdot |C|$ of the bins in C have at least one ball is at most*

$$1/\exp\left(\frac{\delta \cdot |C|}{6}\right)$$

if $q \leq \delta \cdot b/2$ and at most

$$1/\exp\left(\frac{\delta \cdot |C|}{6} \cdot \left(\frac{\delta \cdot b}{q} - 1\right)^2\right),$$

if $\delta \cdot b/2 < q < \delta \cdot b$.

Proof. Let X_i be the random variable that outputs 1 if the ith bin from C has at least one ball. The probability that $X_i = 0$ is at least $(1 - 1/b)^q$ so

$$\Pr\left[\, X_i = 1\,\right] \leq 1 - \left(1 - \frac{1}{b}\right)^q.$$

Now, if $X = \sum_{i=1}^{t} X_i$ is the total number of bins that have at least one ball, we have

$$\mathbb{E}[X] = \sum_{i=1}^{|C|} \mathbb{E}[X_i] \leq |C| \cdot \left(1 - \left(1 - \frac{1}{b}\right)^q\right) \leq |C| \cdot q/b. \tag{8}$$

since $1 - (1 - 1/b)^q \leq q/b$. Because X_1 through X_t are negatively associated, we have

$$\Pr\left[\, X > (1+\alpha) \cdot \mathbb{E}[X]\,\right] \leq 1/\exp\left(\frac{\alpha^2 \cdot \mathbb{E}[X]}{2+\alpha}\right),$$

for $\alpha \geq 0$ by a Chernoff bound. Setting $(1+\alpha) \cdot \mathbb{E}[X] = \delta \cdot |C|$, we have

$$\Pr\left[\, X > \delta \cdot |C|\,\right] \leq 1/\exp\left(\frac{\alpha^2 \cdot \delta \cdot |C|}{(2+\alpha)(1+\alpha)}\right), \tag{9}$$

with

$$\alpha = \frac{\delta \cdot |C|}{\mathbb{E}[X]} - 1 \geq \frac{\delta \cdot |C| \cdot b}{q \cdot |C|} - 1 = \frac{\delta \cdot b}{q} - 1, \tag{10}$$

where the first inequality follows from Eq. 8. We consider only $q < \delta \cdot b$ and break our analysis into two cases: $q \leq \delta \cdot b/2$ and $\delta \cdot b/2 < q < \delta \cdot b$. If $q \leq \delta \cdot b/2$, we have $\alpha \geq 1$ and therefore

$$\frac{\alpha^2}{(2+\alpha)\cdot(1+\alpha)} \geq \frac{1}{6}.$$

Combining this with Eq. 9, we have

$$\Pr\left[\, X > \delta \cdot |C|\,\right] \leq 1/\exp\left(\frac{\delta \cdot |C|}{6}\right).$$

We now consider the case $\delta \cdot b/2 < q < \delta \cdot b$ which, by Eq. 10, implies that $0 < \alpha < 1$. Since $\alpha < 1$, we have

$$\frac{\alpha^2}{(2+\alpha)\cdot(1+\alpha)} > \frac{\alpha^2}{6} \geq \frac{(\delta \cdot b/q - 1)^2}{6},$$

where the second inequality follows from Eq. 10. Again, combining this with Eq. 9 we obtain

$$\Pr[\, X > \delta \cdot |C| \,] \leq 1/\exp\left(\frac{\delta \cdot |C|}{6} \cdot \left(\frac{\delta \cdot b}{q} - 1\right)^2\right).$$

■

We now turn to analyzing the fractional CL-resistance of our algorithm. We make use of Lemmas 5.3 and 5.5 in our Theorem.

Theorem 5.6. *The algorithm Π_λ, described in Fig. 1, is $(n, q, t, \delta, \varepsilon)$-CL-resistant for*

$$\varepsilon = 2^{-t/2} + 1/\exp\left(\frac{\delta \cdot t}{12}\right)$$

when $q \leq \delta \cdot \lambda/2$ and

$$\varepsilon = 2^{-t/2} + 1/\exp\left(\frac{\delta \cdot t}{12} \cdot \left(\frac{\delta \cdot \lambda}{q} - 1\right)^2\right),$$

when $\delta \cdot \lambda/2 < q < \delta \cdot \lambda$.

Proof. The proof is similar to that of Theorem 5.4. We consider an adversary $\mathcal{A}$ in the experiment $\mathbf{CLR}_{\Pi_\lambda,\mathcal{A}}(n, q, t, \delta)$ and in the following we omit the subscripts and parameters for visual clarity. Let S be the random variable denoting the number of servers on which the target VMs are located. Since Π_λ assigns attack VMs uniformly at random amongst a set of λ open servers, we obtain the following from Lemma 5.3:

$$\begin{aligned}\Pr[\,\mathbf{CLR} = 1\,] &= \Pr[\,\mathbf{CLR} = 1 \mid S < t/2\,] \cdot \Pr[\,S < t/2\,] \\ &\quad + \Pr[\,\mathbf{CLR} = 1 \mid S \geq t/2\,] \cdot \Pr[\,S \geq t/2\,] \\ &\leq 2^{-t/2} + \Pr[\,\mathbf{CLR} = 1 \mid S \geq t/2\,].\end{aligned}$$

We use Lemma 5.5 to bound the probability that $\mathbf{CLR}^{\mathsf{unt}}_{\Pi_\lambda,\mathcal{A}} = 1$ conditioned on $S \geq t/2$ by setting $b = \lambda$ and $|C| = i$ for $t/2 \geq i \geq t$. We consider two cases, the first being when $q \leq \delta \cdot \lambda/2$ and the second when $\delta \cdot \lambda/2 < q < \delta \cdot \lambda$. Note that in both cases, the bound of Lemma 5.5 is decreasing in $|C| = S$ therefore, if $S \geq t/2$, it is maximized at $S = t/2$. This gives us,

$$\Pr[\,\mathbf{CLR} = 1\,] \leq 2^{-t/2} + 1/\exp\left(\frac{\delta \cdot t}{12}\right)$$

when $q \leq \delta \cdot \lambda/2$ and

$$\Pr[\,\mathbf{CLR} = 1\,] \leq 2^{-t/2} + 1/\exp\left(\frac{\delta \cdot t}{12} \cdot \left(\frac{\delta \cdot \lambda}{q} - 1\right)^2\right)$$

when $\delta \cdot \lambda/2 < q < \delta \cdot \lambda$.

■

5.4 Perfect Isolation

While we are mainly interested in *computational* isolation, for completeness we provide a Corollary that quantifies the perfect isolation provided by our placement algorithm. The Corollary results from combining Theorems 4.3 and 5.6 and essentially shows that our algorithm performs poorly with respect to perfect isolation.

Corollary 5.7. *Let Δ be an arbitrary deployment scheme and Π be the algorithm described in Fig. 1. Then Δ is (n, q, t, ε)-isolated in $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$, where*

$$\varepsilon = 2^{-t/2} + 1/e^{1/12} \approx 2^{-t/2} + 0.92$$

when $q \leq \lambda/2t$ and

$$\varepsilon = 2^{-t/2} + 1/\exp\left(\frac{1}{12} \cdot \left(\frac{\lambda}{t \cdot q} - 1\right)^2\right) \geq 2^{-t/2} + 0.92,$$

when $\lambda/2t < q < \lambda/t$.

6 Shared Deployments and Computational Isolation

As Theorem 4.3 shows, single CL-resistance bounds perfect isolation but, unfortunately, Corollary 5.7 shows that our algorithm has poor single CL-resistance. On the other hand, Theorem 5.4 shows that our algorithm achieves good complete CL-resistance. Intuitively, what these results show is that, against our algorithm, an adversary has a good probability of co-locating with at least one out of t target VMs but a low probability of co-locating with all t.

In this Section, we show how to take advantage of complete CL-resistance through the use of cryptography. At a high-level, our approach is to assume the adversary is computationally-bounded and to cryptographically "split" a tenant's computation among a set of VMs in such a way that the tenant's secrets can only be recovered if the adversary co-locates with *all* the VMs in the set.

We then show in Theorem 6.2 that if a computation is executed using a shared deployment, its *computational* isolation in a cloud is upper bounded by the *complete* CL-resistance of the cloud's placement algorithm. Combining this with Theorem 5.4, we show in Corollary 6.3 that our algorithm provides good computational isolation.

6.1 Shared Deployments

A shared deployment scheme Σ takes as input a security parameter 1^k, a functionality f, an input x, a deployment size t and a set of resource vectors $\mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_t)$. It outputs a shared deployment $\mathbf{vm} = (\mathrm{vm}_1, \ldots, \mathrm{vm}_t)$. Intuitively, the security guarantee a shared deployment scheme should provide is that no computationally-bounded adversary can learn any partial information about the input unless it corrupts at least ϕ out of the t VMs. We formalize this in the following definition.

Definition 6.1 (Secure shared deployment)**.** *Consider the following probabilistic experiment between an adversary $\mathcal{A}$ and a cloud $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$:*

- $\mathbf{Shrd}_{\Sigma,\mathcal{A},\mathcal{F}}(\phi, t)$*: the adversary $\mathcal{A}$ sends two inputs x_0 and x_1 for the functionality $\mathcal{F}$ to the challenger. The challenger samples a bit b uniformly at random and computes $\mathbf{vm} \leftarrow \Sigma(1^k, \mathcal{F}, x_b, t)$. The adversary then chooses which subset $C \subseteq [n]$ of ϕ VMs to corrupt and given, $\{\mathrm{vm}_i\}_{i \in C}$, outputs a bit b'. The experiment outputs 1 if $b' = b$ and 0 otherwise.*

We say that a shared deployment Σ is (ϕ, t, ε)-secure if for all PPT *adversaries $\mathcal{A}$, for all functionalities $\mathcal{F}$,*

$$\Pr\left[\,\mathbf{Shrd}_{\Sigma,\mathcal{A},\mathcal{F}}(\phi, t) = 1\,\right] \leq \frac{1}{2} + \varepsilon.$$

Constructing shared non-reactive deployments. Using fully-homomorphic encryption, one can construct a $(1, 1, \mathsf{negl}(k))$-secure deployment scheme by storing in a single VM the functionality $\mathcal{F}$ and an FHE encryption of x_1 and having the VM execute the functionality homomorphically. One can avoid the use of FHE, however, by using a multi-worker delegation (MWD) protocol. MWD protocols were first considered in [6], where a construction based on secret sharing and secure function evaluation was proposed. We refer the reader to Appendix B for an overview of MWD protocols and an overview of the concrete construction from [6]. We sketch here how a MWD protocol leads directly to a $(t-1, t, \mathsf{negl}(k))$-secure (non-reactive) deployment scheme.

Let $\mathsf{MWD} = (\mathsf{Encode}, \mathsf{Wrk}_1, \ldots, \mathsf{Wrk}_t, \mathsf{Decode})$ be a MWD protocol. Σ takes as input a security parameter 1^k, a function f, an input x, a size t and a set of resource vectors $(\mathbf{r}_1, \ldots, \mathbf{r}_t)$. It computes $(s_1, \ldots, s_t) \leftarrow \mathsf{Encode}(1^k, f, x, t)$ and returns t VMs $(\mathrm{vm}_1, \ldots, \mathrm{vm}_t)$ such that $\mathrm{vm}_i = (\mathsf{Wrk}_i, \perp, s_i, \mathbf{r}_i)$.

6.2 Computational Isolation

In Theorem 6.2 below, we show that for any placement algorithm Π, the computational isolation of a shared deployment scheme Σ in a cloud $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$ is negligibly-close to Π's *complete* CL-resistance.

Theorem 6.2. *If Σ is a $(t-1, t, \mathsf{negl}(k))$-secure shared deployment scheme and Π is $(n, q, t, 1, \varepsilon)$-CLR, then Σ is computationally $(\varepsilon + \mathsf{negl}(k))$-isolated in $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$.*

Proof. Let $\mathcal{A}$ be an arbitrary real-world PPT adversary and consider the ideal PPT adversary $\mathcal{S}$ that works as follows. Given the resource vectors $\mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_t)$ of the honest deployment, $\mathcal{S}$ simulates $\mathcal{A}$ by feeding it the auxiliary string z. When $\mathcal{A}$ returns a set of q attack VMs and an execution schedule, $\mathcal{S}$ sends the attack VMs to the trusted party. $\mathcal{S}$ then chooses some arbitrary input x and generates a dummy target deployment $(\mathbf{tm}_1, \ldots, \mathbf{tm}_t) \leftarrow \Sigma(1^k, f, x, t, \mathbf{R})$. It then simulates Ctrlr_Π, feeding it the target deployment and the attack VMs according to $\mathcal{A}$'s execution schedule. Whenever an attack VM is placed on a (simulated) server, $\mathcal{S}$ sends the state and inputs of all co-located VMs to $\mathcal{A}$. Throughout, $\mathcal{S}$ forwards any messages between attack VMs and $\mathcal{A}$. If at any point during this simulation, Ctrlr_Π co-locates an attack VM with up to t target VMs, $\mathcal{S}$ aborts. At the end of the simulation, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Note that the simulation of Ctrlr_Π is indistinguishable from its execution in a $\mathbf{Real}_{\Sigma,\Pi,\mathcal{Z},\mathcal{A}}$ experiment and, therefore, so is the placement of VMs. This is because the dummy target deployment has the same resource vectors as the real deployment. But since the VM placements are indistinguishable, so is the adversary's corruption set (i.e., the VMs it is co-located with). Now, let $\mathcal{E}$ be the event that at most $t-1$ VMs of the dummy deployment are co-located with attack VMs. It follows by the $(t-1, t, \mathsf{negl}(k))$-security of Σ that, conditioned on $\mathcal{E}$, $\mathcal{A}$ cannot distinguish between the real and dummy deployments. So, conditioned on $\mathcal{E}$, both the corruption set and the deployments are indistinguishable. It follows then that, conditioned on $\mathcal{E}$, $\mathcal{A}$'s view and, therefore, its output is also indistinguishable.

We also have that the simulated view of Ctrlr_Π is distributed exactly as in a $\mathbf{CLR}_{\Pi,\mathcal{A}}$ experiment, so the probability that $\mathcal{E}$ occurs is exactly $1 - \Pr\left[\,\mathbf{CLR}_{\Pi,\mathcal{A}}(n, q, t, 1) = 1\,\right]$ which, by the complete CL-resistance of Π, is at least $1 - \varepsilon$. In other words, the simulation will fail with probability at most $\varepsilon + \mathsf{negl}(k)$ from which the theorem follows.

■

6.3 Our Algorithm's Computational Isolation

We give a Corollary that quantifies the computational isolation provided by our placement algorithm. The Corollary results from combining Theorems 5.4 and 6.2 and shows that our algorithm performs well with respect to computational isolation.

Corollary 6.3. *Let* Σ *be a* $(t-1, t, \mathsf{negl}(k))$*-secure shared deployment scheme and* Π *be the algorithm described in Fig. 1. If* $t \leq \lambda/e$ *and* $q < \lambda \cdot \log(t/2)$ *then* Σ *is computationally* (n, q, t, ε)*-isolated in* $\mathsf{Cld} = (\mathsf{Ctrlr}_\Pi, \mathsf{Srv}_1, \ldots, \mathsf{Srv}_n)$*, where*

$$\varepsilon = 2^{-t/2} + 1/\exp\left(\gamma \cdot \left(\left(1 - \frac{q}{\lambda \cdot \log(t/2)}\right) \cdot \log(t/2)\right)^2\right) + \mathsf{negl}(k),$$

for some constant $\gamma < 1$.

7 Future Directions

There are many future directions suggested by this work. The most immediate are to establish a bound on our algorithm's CL-resistance that is tighter and/or applicable to a wider range of parameters. Another problem is to find new algorithms with better CL-resistance and, in particular, better *single* CL-resistance which in turn would imply better perfect isolation. Designing algorithms with better tradeoffs between competitive ratio and CL-resistance would also prove to be very useful in practice. Another direction of interest is to achieve *strong* isolation, where the VM resource vectors are protected.

In addition to finding improved algorithms, an interesting line of results would be to establish lower bounds. For example, finding a lower bound on single CL-resistance and, therefore, on perfect isolation would be very interesting (though lower bounds for any δ would be welcome). Similarly, establishing lower bounds on the tradeoffs between competitive ratio and CL-resistance would help us better understand the inherent "cost of security" in this context.

Acknowledgments

The authors thank Yuval Peres for bringing [12] to our attention. Much of this work occurred while Y. Azar, M. Raykova and B. Shepherd were visiting Microsoft Research. Y. Azar and B. Shepherd thank the Theory Group who hosted them during this time (2011-12 academic year).

Y. Azar is supported in part by the Israel Science Foundation (grant No. 1404/10) and by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11). B. Shepherd gratefully acknowledges support from NSERC Discovery and Accelerator Grants.

The work by SRI International was funded by the US Department of Homeland Security (DHS) Science and Technology Directorate (S&T) under contract no. HSHQDC-10-C-00144. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DHS or the US government.

References

[1] A. Aviram, S. Hu, B. Ford, and R. Gummadi. Determinating timing channels in compute clouds. In *ACM Cloud Computing Security Workshop (CCSW)*, pages 103–108, 2010.

[2] Y. Azar, I. Cohen, S. Kamara, and B. Shepherd. Tight bounds for online vector bin packing. In *ACM Symposium on Theory of Computing (STOC '13)*, pages 961–970, 2013.

[3] R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1), 2000.

[4] P. Erdos and A. Renyi. On a classical problem of probability theory. *Magyar Tud. Akad. Mat. Kutató Int. Közl.*, 6:215–220, 1961.

[5] M. R Garey, R. L. Graham, D. S. Johnson, and A. Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, 21(3):257–298, 1976.

[6] S. Kamara and M. Raykova. Secure outsourced computation in a multi-tenant cloud. Technical report, Workshop on Cryptography and Security in Clouds,, 2011.

[7] T. Kim, M. Peinado, and G. Mainar-Ruiz. System-level protection against cache-based side channel attacks in the cloud. In *USENIX Security*, 2012.

[8] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder. Validating heuristics for virtual machines consolidation. Technical Report MSR-TR-2011-9, Microsoft Research, 2011.

[9] P. Li, D. Gao, and M. Reiter. Mitigating access-driven timing channels in clouds using stopwatch. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, 2013.

[10] H. Liu. Amazon data center size. `http://huanliu.wordpress.com/2012/03/13/amazon-data-center-size/`, 2012.

[11] R. Martin, J. Demme, and S. Sethumadhavan. Timewarp: rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. *ACM SIGARCH Computer Architecture News*, 40(3):118–129, 2012.

[12] A. Maurer. A bound on the deviation probability for sums of non-negative random variables. *Journal of Inequalities in Pure and Applied Mathematics*, 4(1):15, 2003.

[13] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder. Heuristics for vector bin packing. Technical report, Microsoft Research, 2011.

[14] H. Raj, R. Nathuji, A. Singh, and P. England. Resource management for isolation enhanced cloud services. In *ACM Cloud Computing Security Workshop (CCSW)*, pages 77–84, 2009.

[15] 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 *ACM conference on Computer and communications security (CCS '09)*, pages 199–212. ACM, 2009.

[16] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[17] J. Shi, X. Song, H. Chen, and B. Zang. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 194–199, 2011.

[18] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[19] M. Tremante. Amazon web services' growth unrelenting. `http://news.netcraft.com/archives/2013/05/20/amazon-web-services-growth-unrelenting.html`, 2013.

[20] V. Varadarajan, T. Ristenpart, and M. Swift. Scheduler-based defenses against cross-vm side-channels. In *Usenix Security*, 2014.

[21] B. Vattikonda, S. Das, and H. Shacham. Eliminating fine grained timers in xen. In *ACM Cloud Computing Security Workshop (CCSW)*, pages 41–46, 2011.

[22] Z. Wang and R. Lee. New cache designs for thwarting software cache-based side channel attacks. *ACM SIGARCH Computer Architecture News*, 35(2):494–505, 2007.

[23] Z. Wang and R. Lee. A novel cache architecture with enhanced performance and security. In *IEEE/ACM International Symposium on Microarchitecture (MICRO).*, pages 83–93, 2008.

[24] G. Woeginger. There is no asymptotic PTAS for two-dimensional vector packing. *Information Processing Letters*, 64(6):293–297, 1997.

[25] Y. Zhang, A. Juels, M. Reiter, and T. Ristenpart. Cross-vm side channels and their use to extract private keys. In *ACM Conference on Computer and Communications Security (CCS '12)*, pages 305–316, 2012.

[26] Y. Zhang and M. Reiter. Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 827–838, 2013.

A The Ideal/Real-World Paradigm

We give an overview of the ideal/real-world paradigm [3] which is often used to analyze the security of cryptographic protocols. In this framework one compares the real-world execution of a protocol to an ideal evaluation of a functionality $\mathcal{F}$. An n-party functionality

$$\mathcal{F}(x_1, \ldots, x_n) = (y_1, \ldots, y_n),$$

takes as input a collection of n strings $(x_1, \ldots, x_n)$, where x_i is the input of the ith party, and returns output y_i to the ith party.

Real-world execution. The real-world execution takes place between an environment $\mathcal{Z}$, n players P_1 through P_n and an adversary $\mathcal{A}$. At the beginning of the execution, the environment $\mathcal{Z}$ and adversary $\mathcal{A}$ receive an auxiliary input $z \in \{0,1\}^*$. In addition, the adversary receives a set $I \subset [n]$ of parties to corrupt (we assume here the adversary is static). The environment then chooses inputs $(x_1, \ldots, x_n)$ for each player and sends input x_i to P_i. The parties execute the protocol Π on their inputs.

At the end of the execution, the environment $\mathcal{Z}$ is given the outputs of the honest parties and an adversarially-chosen function of $\mathcal{A}$'s view and is required to output a bit b. The random variable that returns the output of $\mathcal{Z}$ is denoted $\mathbf{Real}_{\Pi,\mathcal{Z},\mathcal{A},I}(k)$.

Ideal-world execution. The ideal-world execution takes place between an environment $\mathcal{Z}$, n players P_1 through P_n and an ideal adversary $\mathcal{A}'$, all of whom have access to an ideal functionality $\mathcal{F}$ (i.e., a trusted party that computes $\mathcal{F}$). The execution begins with the environment $\mathcal{Z}$ and adversary $\mathcal{A}$ receiving the auxiliary input $z \in \{0,1\}^*$. In addition, the adversary receives the set of corrupted parties $I \subset [n]$. The environment $\mathcal{Z}$ chooses inputs $(x_1,\ldots,x_n)$ for each party and sends x_i to P_i.

The honest parties send their input x_i to the ideal functionality $\mathcal{F}$ while the corrupted parties send $x_i' = x_i$ if $\mathcal{A}'$ is semi-honest and arbitrary values x_i' (i.e., not necessarily x_i) if $\mathcal{A}'$ is malicious. Output delivery works as follows. If a corrupted party sends $\perp$, $\mathcal{F}$ aborts the execution and returns $\perp$ to all parties. Otherwise, it computes $\mathbf{y} \leftarrow \mathcal{F}(\mathbf{x})$ and sends $\{y_i\}_{i\in I}$ to the ideal adversary $\mathcal{A}'$. The latter decides whether to abort or continue the execution. If $\mathcal{A}'$ aborts, $\mathcal{F}$ sends $\perp$ to all the honest parties. If $\mathcal{A}'$ continues, $\mathcal{F}$ sends y_i to party P_i. At the end of the execution, the environment $\mathcal{Z}$ is given the outputs of the honest parties and an adversarially-chosen function of $\mathcal{A}'$'s view and is required to output a bit b. The random variable that returns the output of $\mathcal{Z}$ is denoted $\mathbf{Ideal}_{\mathcal{F},\mathcal{Z},\mathcal{A}',I}(k)$.

Secure emulation. Roughly speaking, we say that a protocol Π securely emulates $\mathcal{F}$ if a real-world execution of Π is equivalent to an ideal-world execution of $\mathcal{F}$. This is formalized by requiring that the outputs of the real- and ideal-world executions be negligibly-close.

Definition A.1 (Secure emulation (static case))**.** *Let $\mathcal{F}$ be an ideal n-party functionality, Π be an n-party protocol and $\phi < n$. We say that Π (ϕ, n)-securely emulates $\mathcal{F}$ if for all* PPT *adversaries $\mathcal{A}$, there exists a* PPT *ideal adversary $\mathcal{A}'$ such that for all* PPT *environments $\mathcal{Z}$, all $I \subset [n]$ such that $|I| \leq \phi$,*

$$\left|\Pr\left[\,\mathbf{Real}_{\Pi,\mathcal{Z},\mathcal{A},I}(k) = 1\,\right] - \Pr\left[\,\mathbf{Ideal}_{\mathcal{F},\mathcal{Z},\mathcal{A}',I}(k) = 1\,\right]\right| \leq \mathsf{negl}(k).$$

Hybrid executions. Hybrid executions are similar to real-world executions except that all parties have access to some ideal functionality $\mathcal{F}' \neq \mathcal{F}$. Hybrid executions allow for modular proofs of security in the ideal/real-world paradigm by allowing one to first prove that the $\mathcal{F}'$-hybrid is indistinguishable from the ideal-world execution of $\mathcal{F}$ and then replacing the ideal functionality $\mathcal{F}'$ in the hybrid with a concrete protocol that securely computes $\mathcal{F}'$. We say that a protocol Π securely emulates $\mathcal{F}$ in the $\mathcal{F}'$-hybrid model if the execution of Π in the $\mathcal{F}'$-hybrid is computationally indistinguishable from the ideal-world execution of $\mathcal{F}$.

B Multi-Worker Delegation

We provide an overview of MWD protocols which were introduced in [6]. We review their syntax and recall the construction of [6] in Section B.1. The protocol is based on SFE and secret sharing, both of which are described below. While the notion of MWD protocols and a concrete construction were presented in [6], a proof of security was deferred to the full version of that work. In Appendix B.1 below we provide such a proof.

Syntax. An MWD protocol [6] consists of a set of algorithms $\mathsf{MWD} = (\mathsf{Encode}, \mathsf{Wrk}_1, \ldots, \mathsf{Wrk}_n, \mathsf{Decode})$ such that Encode takes as input a security parameter k, a function f, an input value x and a threshold ϕ. It outputs a set of n encodings $(s_1,\ldots,s_n)$. Wrk_i, for $i \in [n]$, takes as input an encoding s_i, runs an interactive protocol with $\{\mathsf{Wrk}_j\}_{j \neq i}$ and returns an encoded output d_i. Decode takes as input a set of encoded outputs $(s'_1, \ldots, s'_n)$ and returns a decoded value y. Informally, we say that a multi-worker delegation protocol is (ϕ, n)-secure, for $\phi < n$, if an adversary that corrupts at most ϕ workers cannot learn any useful information about x or y and if $\mathsf{Decode}(s'_1, \ldots, s'_n)$ outputs $\perp$ if any worker deviates from the protocol.

With a multi-worker delegation protocol MWD, a client can securely outsource the evaluation of a function f over an input x to n workers by generating n encodings $(s_1, \ldots, s_n) \leftarrow \mathsf{Encode}(1^k, f, x)$ and sending s_i to the ith worker. The workers compute the encoded outputs $s'_i \leftarrow \mathsf{Wrk}(s_i)$ and return them to the client who recovers $y \leftarrow \mathsf{Decode}(s'_1, \ldots, s'_n)$.

Security. The security of MWD protocols can be formalized in the ideal/real-world paradigm using the ideal $(n+1)$-party functionality

$$\mathcal{F}_{\mathsf{MWD}}(x, \perp, \ldots, \perp) = (f(x), \perp, \ldots, \perp),$$

where the first party is assumed to be the client and the remaining parties are the workers. More precisely, we say that an MWD protocol is (ϕ, n)-secure if it (ϕ, n)-securely emulates the functionality $\mathcal{F}_{\mathsf{MWD}}$.

B.1 A Concrete Multi-Worker Delegation Protocol

We recall the MWD protocol from [6]. The protocol makes black-box use of SFE and secret sharing, which we describe next.

Secret sharing. A threshold secret sharing scheme consists of two efficient algorithms $\mathsf{SS} = (\mathsf{Share}, \mathsf{Recover})$ that work as follows. Share takes as input a secret x from some input space, a number of shares $n \in \mathbb{N}$ and a threshold $\phi < n$ and outputs n shares $(s_1, \ldots, s_n)$. We sometimes make explicit the randomness used by Share by writing $(s_1, \ldots, s_n) \leftarrow \mathsf{Share}(x, n; r)$, where r refers its coins. $\mathsf{Recover}$ takes as input a set of ϕ shares and outputs a secret x.

SS is correct if $\mathsf{Recover}$ returns x when it is given any subset of $(\phi + 1)$ shares of x. It is (ϕ, n)-hiding if, given at most ϕ shares, no adversary can learn any partial information about the secret x. The hiding property is formalized by requiring that for all efficient adversaries $\mathcal{A}$, there exist a PPT simulator $\mathcal{S}$ such that for all secrets x, for all n and all $\phi < n$, $\mathcal{S}$ can generate n shares that are indistinguishable from real shares. Shamir secret sharing [16] provides an efficient instantiation of threshold secret sharing that is information theoretically hiding (i.e., it holds even against a computationally unbounded adversary).

Secure function evaluation. A secure function evaluation protocol $\mathsf{SFE} = (\mathsf{SFE}_1, \ldots, \mathsf{SFE}_n)$ allows n parties to evaluate an n-ary function $f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$ (where the ith party holds input x_i) without revealing any information to each other about their inputs beyond what can be inferred from the output. For $i \in [n]$, SFE_i takes as input a security parameter k, the function f and an input x_i and executes an interactive protocol with $\{\mathsf{SFE}_j\}_{j \neq i}$. At the end of this execution, SFE_i returns $f_i(\mathbf{x})$. We say that SFE is secure if it securely emulates the ideal functionality $\mathcal{F}_{\mathsf{SFE}_f}$ which takes as input $\mathbf{x}$ in the domain of f and returns $f(\mathbf{x})$.

The protocol [6]. Let $\mathsf{SS}_1 = (\mathsf{Share}_1, \mathsf{Recover}_1)$ and $\mathsf{SS}_2 = (\mathsf{Share}_2, \mathsf{Recover}_2)$ be two secret sharing schemes and $\mathsf{SFE} = (\mathsf{SFE}_1, \ldots, \mathsf{SFE}_n)$ be an n-party secure function evaluation protocol. We note that, in practice, SS_1 and SS_2 can be instantiated with the same concrete construction and that we only distinguish between the two schemes for ease of exposition. The protocol works as follows.

Given a security parameter k, a function f, an input x and a threshold ϕ, the Encode algorithm computes and outputs $(s_1, \ldots, s_n) \leftarrow \mathsf{Share}_1(x, n, \phi)$. For all $i \in [n]$, the algorithm Wrk_i runs

$\mathsf{SFE}_i(1^k, f', s_i; r_i)$; that is, it runs the algorithm for the ith player of SFE to securely compute the function f' defined as

$$f'((s_1, r_1), \ldots, (s_t, r_n)) = \mathsf{Share}_2\big(f(\mathsf{Recover}_1(s_1, \ldots, s_n)), n, \phi; r_1 \oplus \cdots \oplus r_n\big).$$

The result of securely computing f' is a set of n shares $(s'_1, \ldots, s'_n)$ which are the encoded outputs. Given these encoded outputs, the Decode algorithm computes and outputs $y \leftarrow \mathsf{Recover}_2(s'_1, \ldots, s'_n)$.

Security. Before formally proving the security of the protocol, we provide some intuition. Consider an adversary $\mathcal{A}$ that corrupts workers and learns the share they received from the client. We argue, informally, that as long as at least one worker remains uncorrupted, the adversary will not learn any information about either the input x or the output $f(x)$ of the client. The confidentiality of the input follows from: (1) the security of SFE, which guarantees that corrupted workers will not learn any information from the SFE execution about the honest worker's input which includes its share of x and its randomness r_i; and (2) from the security of the first secret sharing scheme which guarantees that as long as at least one share of x is unknown to the adversary, it cannot learn any information about x.

The confidentiality of the output follows from the security of the second secret sharing scheme which guarantees that if at least one share of the output $f(x)$ remains unknown to the adversary then no information about it can be recovered. This last property, however, holds only if the randomness used to generate the shares of $f(x)$ is uniform but this is guaranteed to hold if at least one worker is honest because $r_1 \oplus \cdots \oplus r_n$ is uniformly distributed as long as at least one r_i is.

We formalize this intuition in the following Theorem which is in the $\mathcal{F}_{\mathsf{SFE}^{f'}}$-hybrid model.

Theorem B.1. *If SS_1 and SS_2 are hiding, then the protocol MWD described above (ϕ, n)-securely emulates $\mathcal{F}_{\mathsf{MWD}}$ in the $\mathcal{F}_{\mathsf{SFE}^{f'}}$-hybrid model.*

Proof. Let $\mathcal{S}_{\mathsf{SS}_1}$ and $\mathcal{S}_{\mathsf{SS}_2}$ be the simulators guaranteed to exist by the hiding property of SS_1 and SS_2, respectively. Let $\mathcal{A}$ be the real-world adversary and $\mathcal{A}'$ the ideal-world adversary for MWD, respectively. Given a set of parties to corrupt I and an auxiliary input z, the ideal adversary $\mathcal{A}'$ works as follows. It computes $(s_1, \ldots, s_n) \leftarrow \mathcal{S}_{\mathsf{SS}_1}(1^k)$ and $(s'_1, \ldots, s'_n) \leftarrow \mathcal{S}_{\mathsf{SS}_2}(1^k)$. It simulates n workers $\mathsf{Wrk}_1, \ldots, \mathsf{Wrk}_n$, the real-world MWD adversary $\mathcal{A}(I, z)$ and the ideal functionality $\mathcal{F}_{\mathsf{SFE}^{f'}}$. It then sends Wrk_i share s_i. Note that whenever $\mathcal{A}$ corrupts Wrk_i, it receives the simulated share s_i. Next, the simulated workers $\mathsf{Wrk}_1, \ldots, \mathsf{Wrk}_n$ use the ideal $\mathcal{F}_{\mathsf{SFE}^{f'}}$ functionality to compute f'. $\mathcal{A}'$ simulates $\mathcal{F}_{\mathsf{SFE}^{f'}}$ and sends back s'_i to Wrk_i. During the $\mathcal{F}_{\mathsf{SFE}^{f'}}$ simulation, if $\mathcal{A}$ chooses to abort then $\mathcal{A}'$ asks its trusted party to abort. Finally, $\mathcal{A}'$ returns whatever $\mathcal{A}$ outputs.

Next, we show that the output of an $\mathcal{F}_{\mathsf{SFE}^{f'}}$-hybrid execution is indistinguishable from the output of an $\mathbf{Ideal}_{\mathcal{F}_{\mathsf{MWD}}, \mathcal{Z}, \mathcal{A}', I}(k)$ execution. We do this using a sequence of games.

Game_0 consists of the $\mathcal{F}_{\mathsf{SFE}^{f'}}$-hybrid execution: namely, after receiving its input x, the client sends shares $(s_1, \ldots, s_n) \leftarrow \mathsf{Share}_1(x, n, \phi)$ to the n workers who use the ideal $\mathcal{F}_{\mathsf{SFE}^{f'}}$ functionality to securely compute $f'((\widetilde{x_1}, \widetilde{r_1}), \ldots, (\widetilde{x_n}, \widetilde{r_n}))$, where $\widetilde{x_i} = x_i$ and $\widetilde{r_i}$ is chosen uniformly at random if Wrk_i is uncorrupted. If the adversary does not abort the ideal $\mathcal{F}_{\mathsf{SFE}^{f'}}$ computation, each worker Wrk_i sends its output s'_i to the client who computes $y \leftarrow \mathsf{Recover}_2(s'_1, \ldots, s'_n)$. Note that since $\mathcal{A}$ corrupts at most $\phi < n$ workers, if no aborts occur then at least one r_i will be uniformly distributed and, therefore, the shares $(s'_1, \ldots, s'_n)$ computed by the $\mathcal{F}_{\mathsf{SFE}^{f'}}$ functionality will be generated honestly, i.e., according to Share_2.

In Game_1, we replace the shares $(s'_1, \ldots, s'_n)$ with simulated shares $(s'_1, \ldots, s'_n) \leftarrow \mathcal{S}_{\mathsf{SS}_2}(1^k)$. Since $\mathcal{A}$ corrupts at most $\phi < n$ workers we have that: (1) the "real" shares generated in Game_1 are generated honestly; and (2) the simulated shares $(s'_1, \ldots, s'_n)$ are indistinguishable from honestly generated shares by the hiding property of SS_2. It follows that the simulated shares in Game_1 are indistinguishable from the ones in Game_0 and, in turn, that $\mathcal{A}$'s view is indistinguishable as well.

In Game_2 we replace the shares of x in Game_1 with simulated shares $(s_1, \ldots, s_n) \leftarrow \mathcal{S}_{\mathsf{SS}_1}(1^k)$. Since $\mathcal{A}$ corrupts at most $\phi < n$ workers, it sees at most ϕ out of n shares so its view in Game_1 and Game_2 is indistinguishable, otherwise the hiding property of SS_1 would be violated. Note, however, that Game_2 is the $\mathbf{Ideal}_{\mathcal{F}_{\mathsf{MWD}},\mathcal{Z},\mathcal{A}',I}(k)$ execution from which the Theorem follows.

■

C Preliminary Simulation Results

We implemented our algorithm and ran preliminary simulations over several sets of parameters. We chose parameters based on publicly available *estimates* about cloud services such as [10] and [19]. We assume a cloud with $n = 200K$ servers. We consider three values for our algorithm's number of open servers λ: $60K$, $100K$, $150K$. For each of these settings, we consider deployments of size t ranging from 100 to 500. The number q of attack VMs ranges from $1K$ to $150K$ (guaranteeing that $q < \lambda$).

We ran simulations of our algorithm for each combination of parameters and computed the percentage of target VMs that were co-located with attack VMs. When executing a simulation for t target VMs and q attack VMs, we proceed as follows. For $1 \leq i \leq t + q$, we create a VM and assign it to be a target VM with probability $t/(q + t)$ and to be an attack VM with probability $q/(q + t)$. VMs are assigned their resource requirement by choosing, for each dimension, a random value from 0 to 1/10-th of the server capacity which is $1K$ in all our simulations.

Our preliminary results are for 1-dimensional VMs and servers (i.e., $d = 1$) and are described in Fig. 2 for $\lambda = 150K$ and in Fig. 3 for $\lambda = 100K$. We observe that the results are consistent with the bounds for fractional co-location resistance from Theorem 5.6. More specifically, when $q \leq \delta \cdot \lambda$, the fraction of target VMs co-located with attacks VMs is less than δ. For example, when $\delta = 1/3$, we can see in Figure 2 that the fraction of co-located target VMs is less than 33% when $q \leq \delta \cdot \lambda = 50K$. The same holds in Figure 3 when $q < 34K$, in Figure 4 when $q \leq 20K$, and in Figure 5 when $q < 3300$. We also observe that for smaller values of δ the co-location fraction in our simulations are closer to δ for $q = \delta \cdot \lambda$, which tells us that our bounds may be tighter for smaller values of δ.

We also consider two settings with higher dimensional resource/capacity vectors. Here, we run simulations with $\lambda = 60K$ for both $d = 3$ and $d = 6$. The results of the simulations are shown in Figs. 6 and 7 conform to our bounds.

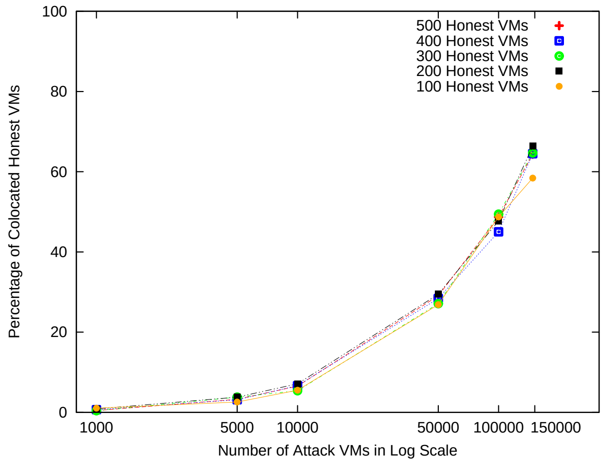

Figure 2: $\lambda = 150K$, $d = 1$

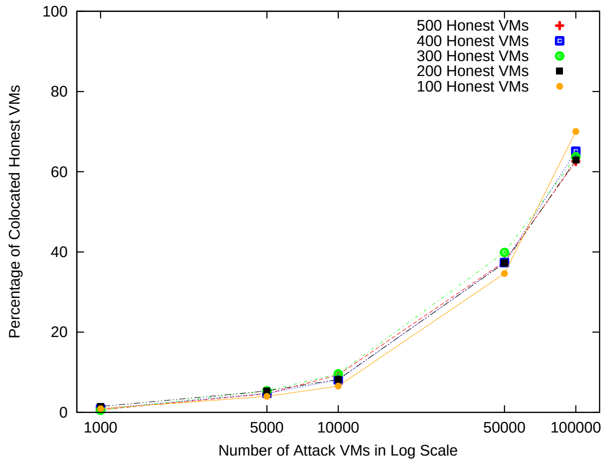

Figure 3: $\lambda = 100K$, $d = 1$

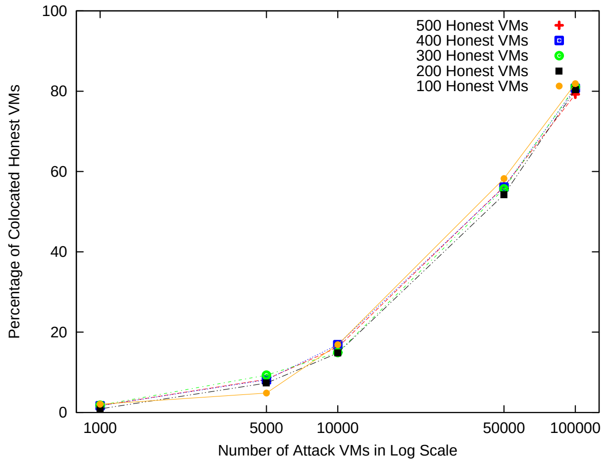

Figure 4: $\lambda = 60K$, $d = 1$

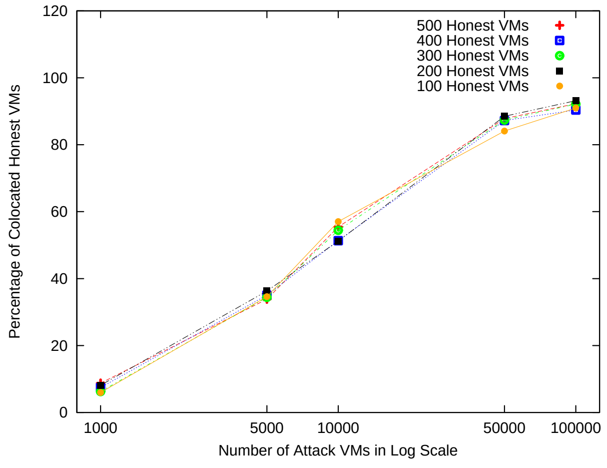

Figure 5: : $\lambda = 10K$, $d = 1$

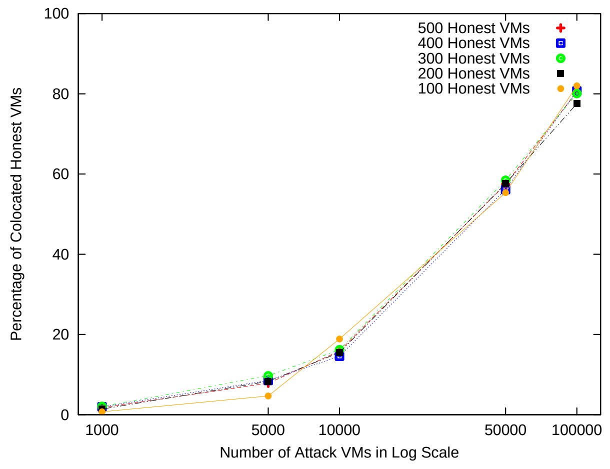

Figure 6: $\lambda = 60K$, $d = 3$

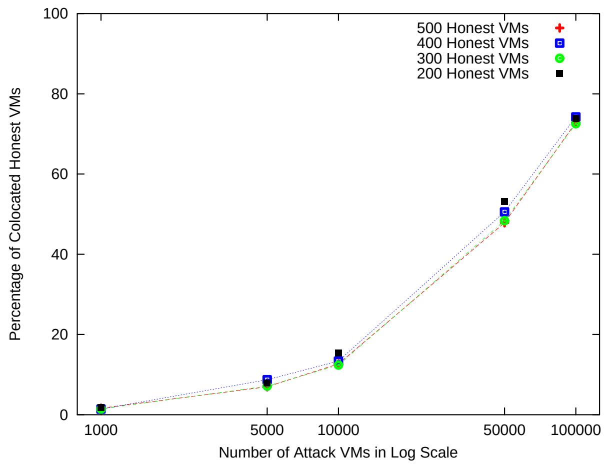

Figure 7: $\lambda = 60K$, $d = 6$