# Bootstrapping and Maintaining Trust in the Cloud

Nabil Schear, Patrick T. Cable II, Thomas M. Moyer, Bryan Richard, Robert Rudd
*MIT Lincoln Laboratory*
{nabil, cable, tmoyer, bryan.richard, robert.rudd}@ll.mit.edu


## Abstract

Today's infrastructure as a service (IaaS) cloud environments rely upon full trust in the provider to bootstrap security. Cloud providers do not offer the ability to create hardware-rooted cryptographic identities for IaaS cloud resources or sufficient information to verify the integrity of systems. Trusted computing protocols and hardware like the TPM have long promised a solution to this problem. However, these technologies have not seen broad adoption because of their complexity of implementation and because they are disconnected from and incompatible with existing security technology like IPsec, Vault, Puppet, or LUKS. In this paper we introduce a scalable trusted cloud key management system called `keylime`. Our system provides an end-to-end solution for both bootstrapping hardware rooted identities for IaaS machines and for the regular checking of the runtime integrity state of the full cloud stack. The key insights of our system are i) a clean separation between the trusted computing layer and the higher level security services that use it, ii) a novel key bootstrapping protocol that incorporates both integrity measurement by an external trusted verifier and the owners intent to spawn resources, and iii) a full trusted computing architecture implementation that is compatible with IaaS services that offer bare metal, virtual machines, or containers. Our evaluation of `keylime` shows that its bootstrapping protocol introduces minimal delay in the creation of IaaS resources and that our system can scalably verify the runtime integrity of thousands of cloud nodes with less than 5 seconds of delay in detecting violations.

This work is sponsored by the Assistant Secretary of Defense for Research & Engineering under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

## 1 Introduction

The proliferation and popularity of infrastructure-as-a-service (IaaS) cloud computing services such as Amazon Web Services, Google Compute Engine, Rackspace, et. al. means that they are increasingly hosting sensitive, private, and business critical data and applications. The only enforceable guarantees that cloud tenants have on the security of these sensitive resources are through legal service-level agreements that loosely define the security responsibilities of providers and tenants. This legal model does not provide tenants with timely and controllable mechanisms with which to react, or even detect, when adversaries strike.

IaaS cloud service providers do not furnish the building blocks necessary to establish a trusted environment for hosting mission-critical applications and data. Tenants have limited ability to verify the underlying platform when they deploy their software and data to the cloud and to ensure that the platform remains in a good state for the duration of their computation. Additionally, current practices restrict tenants' ability to establish unique, unforgeable identities for individual nodes that are tied to a hardware root of trust. Often, identity is based solely on a software-based cryptographic solution or unverifiable trust in the provider. For example, tenants often pass unprotected secrets to their IaaS machines via the cloud provider or use easily spoofed networking information to identify nodes.

Commodity trusted hardware, like the Trusted Platform Module (TPM) [39], has long been proposed as a technical solution for bootstrapping trust, detecting changes to system state that might indicate compromise, and establishing cryptographic identities. To establish a cryptographic node identity with a hardware root of trust, nodes validate their environment and, in turn, are validated by the environment before being issued a cryptographic identity. Cloud nodes can also provide periodic measurements to other hosts to prove they are in a known

state. Technologies like secure boot [21, 19, 1], Trusted Network Connect [38, 36], and Microsoft Bitlocker [28] rely upon the TPM to enable OS boot integrity, secure networking, and secure storage respectively.

Unfortunately, trusted computing technology has not been widely deployed in IaaS cloud environments due to a variety of challenges. TPMs are currently limited to physical platforms, and introduce unreasonable bottlenecks, as they are slow (∼500 msecs. to generate a single digital signature). The limitation to physical platforms only, often means that, if used, only the provider would have access to the trusted hardware, not the tenants. Measuring the *run-time integrity* of an application is an open challenge, and is made even more challenging by long-running applications, such as servers that are running in the cloud. Establishing a hardware root of trust from a virtual machine means an explicit link to the hardware on the physical system which requires multiplexing tpm operations using vTPMs [4, 40] and taking care during VM migration [23]. Despite its small unit cost (as little as $2 for each TPM[1]), many cloud providers typically do not include a TPM chip in their custom developed IaaS hardware systems. Prior research in this space has focused on enabling trusted computing at the provider layer, but these protections do not extend to the tenant layer [2, 31] Trusted computing techniques, protocols, and key management are complex and have so far been directly integrated into the high level services that they enable (like encrypted communications or storage) [25]. This means that existing security services in use today are disconnected from and incompatible with existing trusted computing protocols and techniques.

In this paper, we propose `keylime`, an end-to-end IaaS trusted computing cloud key management service that addresses these challenges. The key insight of our work is to utilize trusted computing to bootstrap identity in the cloud and provide integrity measurement but then allow high level services that leverage these capabilities to exist and operate independently of the trusted computing layer. `keylime` provides support for secure system provisioning, long term cryptographic identity keys for each node, and scalable periodic attestation of system state using an out of band verifier. We provide a simple and clean interface which can then integrate with existing security technologies (see Figure 1). We demonstrate the compatibility of our system by showing its integration with `cloud-init`[2], IPSec, Puppet[3], Vault[4], and LUKS encrypted disks. Our system provides a rich set of powerful security services, rooted in hardware, that can scale to wide-spread use of IaaS services.

[1]According to a recent search on `http://www.digikey.com`
[2]`http://launchpad.net/cloud-init`
[3]`http://puppetlabs.com/`
[4]`http://hashicorp.com/blog/vault.html`

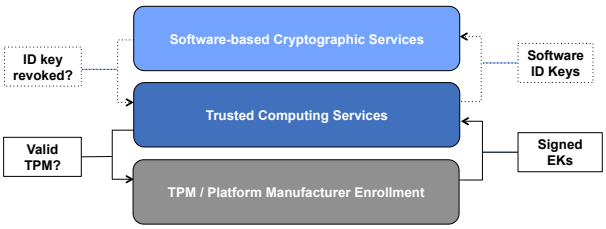

Figure 1: Simplified interactions between service layers

Our work encourages current IaaS providers to adopt TPMs by making trusted computing accessible and easy to integrate for current customers. We show how a virtualized TPM implementation can be layered with hardware TPMs to create a common IaaS trusted computing layer.

Our principal contributions in this paper are:

- A fully integrated cloud trusted computing architecture that provides a clean interface to standard cryptographic services for use in IaaS clouds.
- A novel bootstrap key derivation protocol that enables centralized integrity measurement using an out of band verifier while also ensuring a tenant's intent to derive a key.
- In addition to supporting bare metal cloud environments, we provide a full solution for TPM enrollment, storage key boot strapping, and periodic integrity measurement that is compatible with virtualized environments like virtual machine or container hosting.

We show that `keylime` can scale to handle thousands simultaneous nodes and perform integrity checks on nodes at a rate of 380 quotes verified per second. We present and evaluate multiple options for deploying our integrity measurement verifier both in the cloud, in a low cost appliance, and on-premises We show that the overhead of secure provisioning imposes only a *sim*2s overhead on creating new VMs. Finally, we find that our system can respond to integrity measurement violations and respond by revoking access in high level services in as little as 150ms. No other published or commercial trusted computing implementation provides either the scale, end-to-end integration, support for both load and runtime integrity, or the ease of adoption of our solution.

## 2 Background

**Trusted Computing** Trusted computing (TC) provides the building blocks for creating systems that are amenable to integrity monitoring. The TPM, as specified

by the Trusted Computing Group[5], is a low-power cryptographic processor that provides key generation, protected storage, and cryptographic operations. The protected storage includes a set of Platform Configuration Registers (PCRs) where the TPM stores SHA-1 hashes. The TPM also uses these registers to store *measurements* of integrity-relevant components within the system.

In order to store a new measurement in a PCR, the `extend` operation concatenates the existing PCR value with the new measurement, hashes that value, and stores the resulting hash in the register. This hash chain allows a verifier to confirm that a set of measurements reported by the system has not been altered. This report of measurements is called an *attestation*, and relies on the `quote` operation, which accepts a nonce and a set of PCRs. The TPM reads the PCR values, and then signs the nonce and PCRs with a key that is only accessible by the TPM. The key the TPM uses to sign quotes is called an attestation identity key (AIK). This attestation can include measurements of the BIOS, firmware, bootloader, hypervisor, OS, and applications, depending on the configuration of the system. The AIK itself is protected using the Storage Root Key (SRK), which is generated by the platform owner during initialization. The Endorsement Key (EK) is generated by the TPM manufacturer, and uniquely identifies the TPM. The private EK never leaves the TPM, and is never erased. The SRK is erased each time the TPM is reset to factory default settings.

**Integrity Measurement** To measure a system component, the underlying component must be "trusted computing-aware". The BIOS in systems with a TPM already supports measurement of firmware and bootloaders. TPM-aware bootloaders exist that can measure hypervisors and operating systems [21, 19, 1]. To measure applications, the operating system must support measurement of applications that are launched, such as the Linux Integrity Measurement Architecture [30] or the Policy-Reduced IMA [20]. One limitation of approaches like IMA is the inability to monitor the *run-time state* of the applications, which Nexus aims to address with a new OS that makes trusted computing a first-class citizen, and supporting introspection to validate run-time state [35].

Several proposals exist for providing *run-time integrity monitoring* including LKIM [24] and DynIMA [11]. These systems ensure that a *running system* is in a known state, allowing a verifier to validate not only that what was loaded was known, but that it has not been tampered with while it was running. These systems often work by monitoring data structures that are critical to the integrity of the system, and ensuring that invariants are not violated, e.g., the process queues in the Linux kernel in the case of LKIM. Some systems attempt to gather runtime integrity from arbitrary user-space applications but they often have limited ability to detect application corruption [11].

In addition to operating system validation, others have leveraged trusted computing and integrity measurement to support higher-level services, such as protected access to data when the client is offline [22], or to enforce access policies on data [26]. Others have proposed mechanisms to protect the server from malicious clients, e.g., in online gaming [3], or applications from a malicious operating system [9, 10, 16]. However, these proposals do not account for the challenges of migrating applications to a cloud environment, and often assume existing infrastructure to support trusted computing key establishment and agreement.

**IaaS Cloud Services** In the Infrastructure-as-a-Service (IaaS) cloud service model, users can request an individual compute resource (also referred to as a node). Users provision nodes in two ways: either by uploading a whole image to the provider or by configuring a paired-down base image that the provider makes available. Users unsure of their technology stack often begin by customizing a provider-supplied image, then create their own images (using a tool like `Packer`[6]) to decrease the amount of time it takes for a node to become ready.

A standard mechanism across cloud providers exists for end-users to specify bootstrapping information called cloud-init. cloud-init has a variety of plugins and accepts a YAML-formatted description of what bootstrapping actions should be taken. Examples of such actions include: adding users, adding package repositories, or running arbitrary scripts. These bootstrapping instructions are not encrypted, meaning that a provider could intercept secrets passed via the bootstrapping instructions. Users of cloud computing resources "at scale" typically spawn new cloud instances using an application programming interface and pass along enough bootstrapping information to allow the instance to communicate with a configuration management platform (such as Cfengine [7], Chef, Puppet, etc.) for further application/instance-specific configuration. In Section 4.1 we cover how `keylime` eliminates the identity problem involved with bootstrapping configuration management tools with cloud-init.

## 3 Design

In order to address the limitations of current approaches, i.e.,fully trusting cloud providers and only having legal SLAs, we can consider the union of trusted computing

[5] `http://trustedcomputinggroup.org`

[6] `https://www.packer.io/`

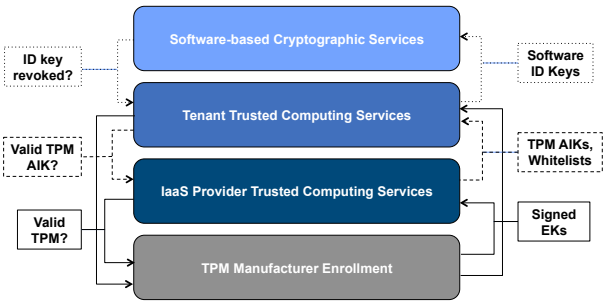


Figure 2: Full IaaS service interactions.

and IaaS to provide a hardware root-of-trust that tenants can leverage to establish trust in the infrastructure, and furthermore their own systems running on that infrastructure. This section considers the threats that `keylime` addresses, and how to leverage existing trusted computing constructs in a virtual environment without also introducing unreasonable latency.

## 3.1 Threat Model

Our goal in this paper is minimize trust in the cloud provider while carefully accounting for all concessions we must make to enable bootstrapping of trusted computing services. We assume the cloud provider is semitrusted, i.e., they are organizationally trustworthy but are susceptible to compromise or malicious insiders. We assume the cloud provider has processes, technical controls, and policy in place to limit the impact of compromise from spreading across their entire infrastructure. Thus, in the semitrusted model, we assume that some fraction of the cloud provider's resources is under the control of adversaries. Specifically, we assume that the adversary can monitor or manipulate a fraction of the network or cloud storage arbitrarily. We assume that the adversary may *not* physically tamper with any hypervisor CPU, bus, memory, or TPM. We further assume that the hypervisor or other isolation kernel software in use by the provider *does not* have the ability to read the memory (and subsequently steal the ephemeral keys) of tenants' compute resources (e.g., by using CloudVisor [41]). We assume that TPM manufactures have created the appropriate endorsement, platform, and conformance credentials and have a service where we can query the validity of those of credentials. Applying this threat model overall, we expect and design for cloud compromise, but we assume there is some trustworthy base of resources on which we can bootstrap trusted computing.

## 3.2 Architecture

To introduce the architecture of `keylime` we first describe a simplified architecture for managing trusted computing services for a single organization, or cloud


Figure 4: Simplified registrar enrollment protocol. This is similar to the TCG specification for Privacy CA's, and serves as a simplified, but notionally compatible, protocol.

tenant, without virtualization (see Figure 3). We then expand this simplified architecture into the full `keylime` architecture, providing extensions that allow layering of provider/tenant trusted computing services and supporting multiple varieties of IaaS execution isolation (i.e., bare metal, virtual machines, or containers). Figure 5 depicts the full system architecture.

The first step in bootstrapping the architecture is to create a tenant-specific registrar. This node will store and certify the AIKs of the TPMs in the tenant's infrastructure. In the simplified architecture, the tenant registrar can be hosted outside the cloud in the tenant's own infrastructure or could be hosted on a physical system in the cloud. The registrar does not store any tenant secrets a priori, so the tenant can decide to trust the registrar's private signing key only after it attests its integrity state to the tenant. We expand on how to virtually host the tenant registrar and cloud verifier in the cloud in Section 3.2.1.

The next step is to enroll cloud node TPMs into the tenant-specific TPM registrar. We can leverage existing standards for the creation and validation of AIKs and credentials by creating a TCG Privacy CA [37]. Alternatively, we can emulate that process more simply by creating a registrar that simply stores TPM AIK public keys in a key/value store indexed by node UUID. The simplified registrar can ensure that the TPM endorsement credentials are valid by querying the TPM manufacturer. Importantly, the generation, and installation, of the EK by the TPM manufacturer is the only operation in which the adversary may not interfere. We give the simplified node enrollment protocol in Figure 4.

The core component of `keylime` is an out of band cloud verifier (CV) similar to the one described by Schiffman et al. [32]. Each cloud organization will have at least one CV that is responsible for verifying the system state of the organization's IaaS resources. The CV relies upon the tenant registrar for validating that the AIKs used to sign TPM quotes are valid. The registrar, CV, and the associated cloud node services with which they interact are the only components in `keylime` that manage and use keys and public key infrastructures associated with the TPM.

The CV participates in a three party key derivation protocol (we describe in detail in Section 3.2.2) where the CV and tenant cooperate to derive a key at the cloud

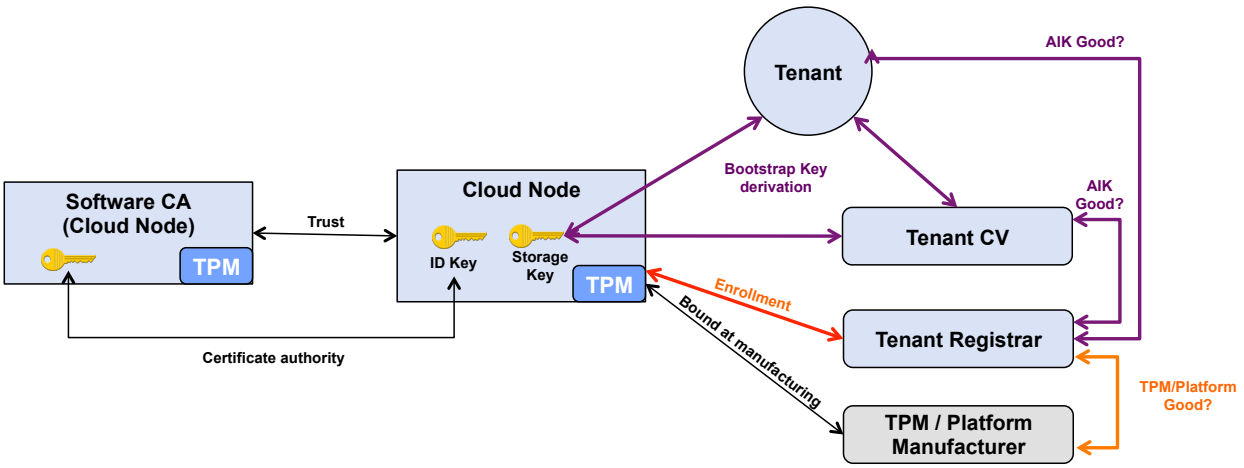


Figure 3: Physical trusted computing architecture

node to support initial storage decryption. This protocol is akin to the method by which a user can use the TPM to decrypt his or her disk in a laptop. To allow the decryption key to be used to boot the laptop, the user must enter a password (demonstrating the user's intent) and TPM PCRs must match a set of whitelisted integrity measurements (demonstrating the validity of the system that will receive the encryption key). In an IaaS cloud environment, there is neither a trusted console where a user can enter a password nor is there a way to pre-seed the TPM with the key and measurement whitelist. Our protocol uses secret sharing to solve these problems by relying externally upon the CV for integrity measurement and by having the tenant directly interact with the cloud node to demonstrate intent to derive a key. The tenant uses the initial bootstrap key to protect tenant secrets and trust relationships. The tenant can use this storage key to unlock either its disk image or to unlock tenant-specific configuration provided by cloud-init. Once the CV has finished bootstrapping a cloud node, it periodically polls each cloud node's integrity state to determine if any runtime policies have been violated. The CV is responsible for notifying higher level services of integrity measurements that violate the tenant's policy.

To cleanly link trust and integrity measurement rooted in the TPM to higher level services, we create a parallel software-only PKI and a simple service to manage it. We refer to this service as the software CA. To bootstrap this service, we use the key derivation bootstrap protocol to create a cloud node to host the service. Since the bootstrap key derivation protocol ensures that the node can only derive a key if the tenant authorizes it and if the node's integrity state is approved, we can encrypt the private key for the software CA and pass it to the node upon provisioning. Once established, we can then start other cloud nodes and securely pass them keys signed by this CA. The security of this association, the bootstrapping of relevant keys, and user intent to create new resources are again ensured using the bootstrap key derivation protocol. Once established, the software CA identity credentials that each node now possess can be directly used by standard cryptographic tools and services like IPsec or Puppet.

To complete the linkage between the trusted computing services and the software identity keys, we need a mechanism to revoke keys in the software PKI when integrity violations occur in the trusted computing services layer. The CV is responsible for notifying the software CA of these violations. The CV includes metadata about the nature of the integrity violation which allows the software CA to have policy for how to respond. The software CA in turn can support standardized methods for certificate revocation like certificate revocation lists or by hosting an OCSP responder. To support push notifications of failures, the software CA can also publish signed notifications to which services that directly support revocation actions can subscribe (e.g., to trigger a re-key in secret manager like Vault).

This architecture handles bootstrapping and infrastructure creation with as few assumptions as possible. The root of trust established by the TPM EK created by the TPM manufacturer and the establishment of a tenant registrar extend directly to high level crypto services like the ephemeral keys protecting data transmitted between cloud nodes using IPsec.

### 3.2.1 Layering Trust

We next expand this architecture to work across the layers of virtualization common in today's IaaS environments. Our goal is to create the architecture described previously that cleanly links common security services to a trusted computing layer in a *cloud tenant's* environ-

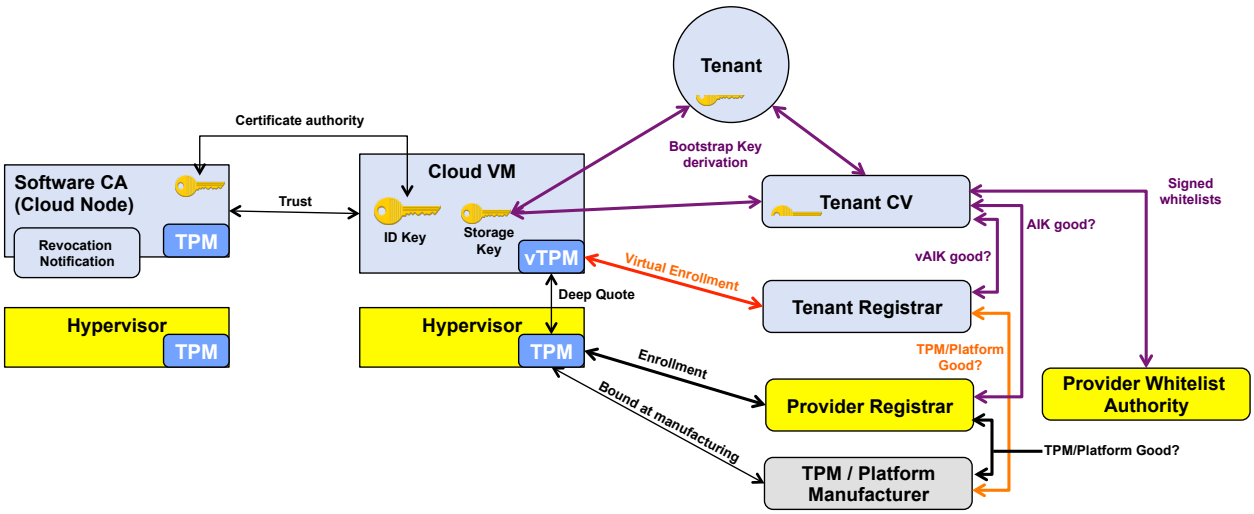


Figure 5: Virtualized trusted computing architecture

ment. Thus, in an VM hosting environment like Amazon AWS or OpenStack, we aim to create trusted computing enabled software CAs and tenant nodes inside of virtual machine instances. Note that, in a bare-metal provisioning environment like IBM Softlayer, HaaS [14], or OpenStack Ironic, we can directly utilize the above simplified architecture where there is no trust layering.

We observe that IaaS-based virtual machines, or physical hosts all provide a common abstraction of on-demand isolated execution. Each form of isolated execution in turn needs a root of trust on which to build trusted computing services. Due to the performance limitations of typical TPMs (e.g., taking 500 or more milliseconds to generate a quote), direct multiplexed use of the physical TPM will not scale to the numbers of nodes that can be hosted on a single modern system. As described by Berger et al. [4] and as implemented in Xen [12], we advocate for a virtualized implementation of the TPM. Each VM has its own software TPM (called a vTPM) whose trust is in turn rooted in the hardware TPM of the hosting system.

The TPM interface to client software is the same as though that software were interacting with a real TPM. The only exception to this, is that the client can request a *deep-quote* that will get a quote from the hardware TPM in addition to getting the state of the vTPM. These quotes are linked together by including a hash of the vTPM quote and nonce in the hardware TPM quote. Deep quotes suffer from the slow performance of hardware TPMs, but as we'll show in this section, we can limit the use of deep quotes while still maintaining reasonable performance and scale.

To assure a chain of trust that is rooted in hardware, we need the IaaS provider to replicate some of the trusted computing service infrastructure in their environment and allow the tenant trusted computing services to query it. Specifically, the provider must establish a registrar for their infrastructure, must publish an up-to-date signed list of the integrity measurements of their infrastructure, and may even have their own CV. The tenant CV will interact with the whitelist authority service and the provider's registrar to verify deep quotes collected by the infrastructure.

Despite the fact that most major IaaS providers run closed-source customizations of open source hypervisors and would provide what amounts to opaque integrity measurements [17], we find there is still value in verifying the integrity of the providers services. We limit the risk by assuming that the provider is semitrusted and they have controls in place that limit an adversary from both subverting the code loading process on all hypervisors *and simultaneously* disrupting the publishing and signing process for measurements.

As in Section 3.2, we begin with the establishment of a tenant registrar and cloud verifier. There are multiple options for hosting these services securely: i) in a bare metal IaaS instance with TPM, ii) on-tenant-premises in tenant-owned hardware, iii) in a small trusted hardware appliance deployed to the IaaS cloud provider, and iv) in a IaaS virtual machine. The first three of these options rely upon the architecture and protocols we've already discussed. The last option requires the tenant to establish an on-tenant-premises CV and use that to bootstrap the tenant registrar and CV. This on-tenant-premises CV identifies and checks the integrity of the tenant's virtualized registrar and CV, who then in turn are responsible for the rest of the tenant's virtualized infrastructure.

The primary motivations for a tenant choosing between these options are the scale of IaaS instances in their environment, bandwidth between the tenant and the

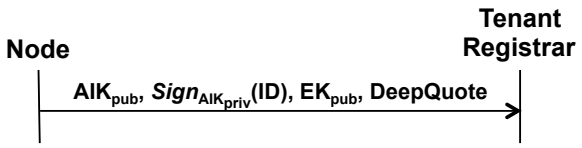


Figure 6: Virtualized Instance registrar enrollment protocol.

IaaS cloud, and cost. Option 1 provides maximum performance but at higher cost. Option 2 will by limited by bandwidth and would be more costly to maintain resources outside of the cloud. Option 3 is a good trade-off between cost and performance for a small cloud tenant. Finally, Option 4 provides compatibility with current cloud operations, good performance and scalability, and low cost. This comes at the expense of increased complexity. In Section 5, we examine the performance trade-offs of these options including a low-cost registrar and CV we implemented on a Raspberry Pi.

Once we have created the tenant registrar and CV, we can begin securely bootstrapping nodes into the environment. As before, the first node to establish is a virtualized software CA and we do by creating a private signing key offline and protecting it with a key that will be derived by the bootstrap key derivation protocol. The following process will be the same for all tenant cloud nodes. When a node boots, it will get a vTPM from the IaaS provider. We can enroll the vTPM into the tenant registrar by generating the appropriate vTPM credentials (EK, SRK, AIK) and by requesting a deep quote bound to those credentials from the hardware TPM on the host on which it runs (see Figure 6). The tenant then can query the TPM manufacturer to determine that the hardware TPM backing the vTPM is a valid hardware device. They then ask the provider registrar if the AIK from the Deep Quote is valid. Finally, the tenant registrar will request the latest valid integrity measurements from the provider and check the PCRs in the Deep Quote. Once enrolled, the node can use the bootstrap key derivation protocol to unlock its private CA signing key and begin operation.

When considering the cost of performing a Deep Quote, it becomes necessary for the provider to carefully consider the additional latency that is introduced by accessing the physical TPM. The Deep Quote provides a link between the vTPM and the physical TPM of the machine, and as such new enrollments should always include Deep Quotes. When considering if Deep Quotes should be used as part of periodic attestation, it becomes necessary to understand what trusted computing infrastructure the provider has deployed. If the provider is doing nothing, then Deep Quotes will only reflect the binding between the vTPM and the physical TPM. If the provider provides measurement whitelists and access to the measurements, the tenant can perform integrity verification. Finally, the provider can deploy `keylime` and provide tenants with access to the integrity state of the end nodes. Finally, the provider must be concerned with how to service many tenant requests for Deep Quotes without introducing unreasonable latency. One approach is to leverage batch attestations, as proposed in [29, 31], where multiple Deep Quote requests are serviced by a single Deep Quote.

### 3.2.2 Key Derivation Protocol

We now introduce the details of our bootstrap key derivation protocol. The goal of this protocol is for the cloud tenant to obtain key agreement with a cloud node they have provisioned in an IaaS system. The protocol relies upon the CV to provide integrity measurement of the cloud node during the protocol. The tenant also directly interacts with the cloud node to demonstrate their intent to spawn that resource and allow it to decrypt sensitive contents. However, the tenant does not directly perform integrity measurement. This separation of duties is beneficial because the attestation protocols may operate in parallel and it simplifies deployment by centralizing all integrity measurement, white lists, and policy in the CV. Figure 7 depicts the messages the parties exchange.

To begin the process, the tenant generates a *fresh* random symmetric encryption key $K$. The cloud tenant uses AES-GCM to encrypt the sensitive data to pass to the node $d$ with $K$, denoted $Enc_K(d)$. The tenant then performs trivial secret sharing to split $K$ into two parts $U$, which the tenant will retain and pass directly to the cloud node and $V$, which the tenant will share with the CV to provide to the node upon successful verification of the node's integrity state. To obtain these shares the tenant generates a secure random value $V$ the same length as $K$ and computes $U = K \oplus V$.

In the next phase of the protocol, the tenant requests the IaaS provider instantiate a new resource (i.e., a new virtual machine). The tenant sends $Enc_K(d)$ to the provider as part of the resource creation. The data $d$ may be configuration meta-data like a cloud-init script[7]. Upon creation, the provider returns a unique identifier for the node *uuid* and an IP address at which the tenant can reach the node[8].

After obtaining the node *uuid* and IP address, the tenant must notify the CV of their intent to boot a cloud node and to begin verification (see area $A$ in Figure 7. The tenant connects to the CV over a secure channel, such as mutually authenticated TLS, and provides $v$, *uuid*, node

[7]Because $K$ may not be re-used in our protocol, the cost of re-encrypting large disk images for each node may be prohibitive. We advocate for creating small sensitive data packets like a cloud-init script, and then establish local storage encryption with ephemeral keys.

[8]If the provider does not offer a unique ID service, the tenant can instead create *uuid* themselves and give it to the provider with along with $Enc_K(d)$.

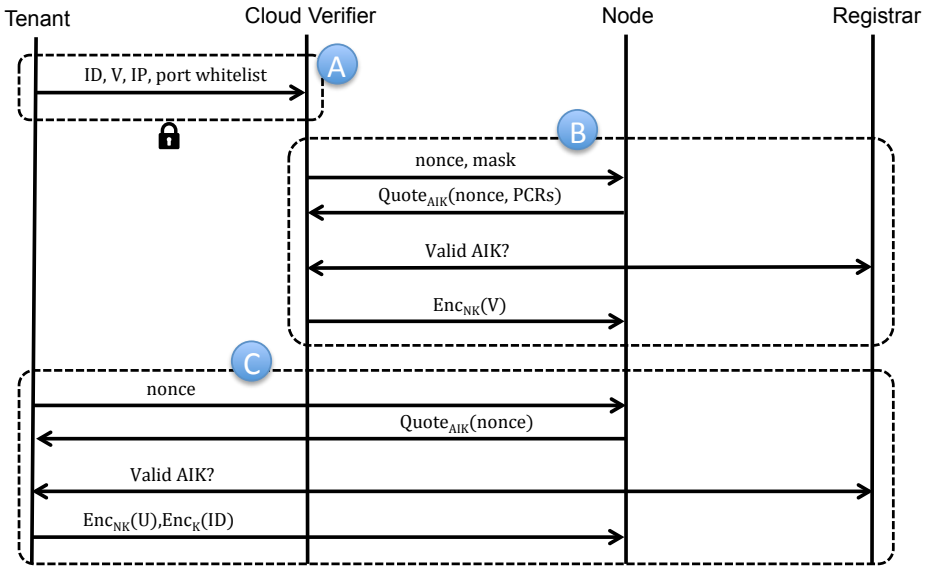


Figure 7: Three Party Bootstrap Key Derivation Protocol.

IP, and a TPM policy. The TPM policy specifies a white list of acceptable measurements to expect from the TPM of the cloud node. At this point the CV and tenant can begin the attestation protocol in parallel.

The attestation protocol consists of two message volleys, the first for the initiator to request a TPM quote and the second for the initiator to provide a share of $K$ to the cloud node upon successful validation of the quote. Since we use this protocol to bootstrap keys into the system, there are no existing keys with which we can establish a mutually authenticated secure channel using TLS or IPSec. Thus, this protocol must securely transmit a share of $K$ over an untrusted network. We accomplish this by creating an ephemeral asymmetric key pair on the node, denoted $NK$, outside of the TPM and then use the TPM quote to bind this key to the identity of the TPM. We do so by including a hash of $NK_{pub}$ along with the nonce in the extra data field in the quote. The initiator can then encrypt one share of $K$ using $NK_{pub}$ and securely return it to the cloud node.

The attestation protocol of our scheme is shared between the interactions of the tenant and the cloud node and that of the CV and cloud node with only minor differences between them. The tenant and CV are both the initiators of this protocol. The overall goal of this attestation protocol is to allow the cloud node to attest to its identity and integrity state and then to obtain one of the shares of $K$. The tenant provides $U$ and the cloud verifier provides $V$ to the cloud node. The differences arise in how the TPM quotes are validated. To avoid needing to sequence all three parties, we designed the protocol to operate in parallel. We also wish to centralize the adjudication of detailed integrity state to the cloud verifier. Thus, the TPM quote that the tenant requests serves only to verify the identity of the cloud node's TPM and says nothing about its integrity state. Since the tenant generates a fresh $K$ for each IaaS resource creation request, we are not concerned with leaking $U$ to a node with either an invalid TPM or unexpected integrity state. Furthermore, since $V$ by itself is not valuable, the CV cannot be subverted to launch resources without user intent.

We now describe this protocol in detail. The initiator first sends a fresh nonce $n$ to the cloud node along with a mask indicating which PCRs the cloud node should include in their quote. In the case of the CV, the tenant specifies the mask as part of the TPM policy exchanged earlier. For the tenant quote request, this PCR mask is empty as the purpose of this exchange is only to prove to the tenant that the registrar certifies the cloud node has a known TPM. The initiator requests a quote from the TPM with the given policy. The external data of the quote is set to $SHA1(n|NK_{pub})$. The node then returns $Quote_{AIK}(extdata, PCRs), NK_{pub}$ to the initiator.

The initiator then confirms that the AIK is valid according to the tenant registrar. If the initiator is the CV, then it will also check the other PCRs to ensure they are valid according to the tenant-specified whitelist. If the node is virtual, then the quote to the CV will also include a deep quote of the underlying hardware TPM. The CV will in turn validate it as described in the previous section.

Upon successful verification, the initiator can then return their share of $K$. Thus the tenant sends $Enc_{NK}(U)$ and the cloud verifier sends $Enc_{NK}(V)$ to the node. The cloud node can now recover $K$ and proceed with the boot/startup process. The cloud node does not retain $K$ or

$V$. To support reboot or migration, the cloud node stores $U$ in the TPM NVRAM to avoid needing the tenant to interact again. After rebooting, the node must again request verification by the CV to obtain $V$ and re-derive $K$. We assume that vTPM state will migrate along with the cloud node to which it is bound to to avoid losing $U$.

# 4 Implementation

We implemented `keylime` in ∼3.2k lines of Python in four components: registrar, node, CV, and tenant. The registrar offers a REST-based web service for enrolling node AIKs, and their TPM endorsement credentials. It also supports a query interface for checking the keys for a given node UUID.

The node component runs on the IaaS machine, VM, or container and is responsible for responding to requests for quotes and for accepting shares of the bootstrap key $K$. It provides an unencrypted REST-based web service for these two functions. As described earlier, the protocols for exchanging data with the node are secure at the protocol level and do not require additional transport layer encryption like TLS. The node uses the IBM Software Trusted Platform module library code [18] and directly interacts with the TPM rather than going through a Trusted Software Stack (TSS) like Trousers. We chose this library because it is compatible with a TPM software emulator (for testing) and with the `/dev/tpm0` interface to a real TPM (for operational use). We extended the IBM library to provide command line utilities for creating and verifying quotes and deep quotes. We then call those utilities from Python.

The cloud verifier hosts a TLS-protected REST-based web service for control. Through this authenticated interface, tenants can add and remove nodes to be verified and also query their current integrity state. Upon being notified of a new node, the CV enqueues metadata about the node onto the `quote_request` queue where a configurable pool of worker processes will then request a deep quote from the node. The CV verifies the quote by checking the signing key with the registrar and verifying the PCR values from the quote match the whitelist provided by the tenant. Upon successful verification of the quote, the CV will provide $V$ to the node via an HTTP POST through a distinct pool of worker processes. The CV uses PKCS#1 OEAP and with RSA 2048 keys to protect shares of $K$ in transit. Since, the node caches it's copy of $V$, the CV can now bypass the provide $V$ pool altogether in future iterations and proceed to obtain quotes from the node as fast as possible. If the node reboots or otherwise looses $V$, it will notify the CV that it needs $V$ in the response to a request for a quote. The CV also caches the nodes AIK and only periodically checks the registrar to determine that a node's TPM is still valid.

The tenant generates a random 256 bit AES key and encrypts and authenticates the bootstrap data using AES with Galois Counter Mode [27]. The tenant uses trivial XOR-based secret sharing to split $K$ into $V$ and $U$. The tenant script executes a simplified version of the the same protocol that the CV uses. The tenant will poll for a valid connection to the node's REST service so that the tenant script can be called as soon as the tenant requests a new node from the IaaS provider. The tenant script always directly checks with the registrar to determine if the quote signing AIK is valid and owned by the tenant.

Upon receiving $U$ and $V$, the node can then combine them to derive $K$. During the protocol to provide $U$, the tenant also supplies the cipher text of $E_K(ID)$ to the node. This provides the node with a quick check to determine if $K$ has been derived correctly and will decrypt the ID. To limit the impact of rouge CVs or tenants connecting to the node's unauthenticated REST interface, the node stores all received $U$ and $V$ values and iteratively tries each combination to find the correct $K$. Once the node has correctly derived $K$, it mounts a small in-memory file system using `tmpfs` and writes the key there for other applications to access. After correctly deriving $K$, the node stores $U$ in its TPM NVRAM so that the tenant only needs to demonstrate their intent to spawn a resource at bootstrapping time and does not have to interact with the node later in the node's life cycle.

## 4.1 Integration

While the bootstrapping key derivation protocol of `keylime` is generic and can be used to decrypt arbitrary data, we believe the most natural cloud use-case for it is to decrypt a small IaaS node-specific package of data. The node can then start with this small trusted package and continue the bootstrapping process from there. To enable this use-case we have integrated `keylime` with the cloud-init package, the combination we call trusted-cloud-init. As described in Section 2, cloud-init is widely adopted mechanism to deploy machine-specific data to IaaS resources. It is compatible with a variety of cloud platforms including Amazon EC2, OpenStack, Google Cloud Platform, and Microsoft Azure. To integrate `keylime` and cloud-init, we patched cloud-init to support AES-GCM decryption of the `user-data` (where cloud-init stores tenant scripts and data). We modified the upstart system in Ubuntu Linux to start the node process of `keylime` early in the boot process before cloud-init. We then configure cloud-init to find the key that `keylime` creates in the `tmpfs` mounted file system. After successful decryption, cloud-init can delete the key and scrub it from memory. Since cloud-init, by default, writes the decrypted `user-data` to the disk in `/var/lib/cloud`, we take care to ensure that the ap-

propriate locations are either on encrypted volumes (e.g., using LUKS) or are mounted on non-persistent memory-backed file systems like `tmpfs`.

To support transparent integration with an IaaS platform, we created a wrapper for the OpenStack Nova command line interface that enables trusted-cloud-init. Specifically, our wrapper intercepts calls to `nova boot` and automatically encrypts the provided `user-data`. It then executes the provided `nova boot` command and passes the encrypted `user-data` to OpenStack. It then calls the `keylime` tenant which begins the bootstrapping protocol. This allows existing OpenStack users to transparently use `keylime` and cloud-init without needing to fully trust the OpenStack provider not to tamper or steal sensitive the contents of their `user-data`.

To support applications that need node identities that do not manage their own PKIs, we implemented a simple software CA. The tenant provisions the software CA itself by creating the CA signing key offline and delivering it to a new node using trusted-cloud-init. Also delivered via trusted cloud-init are certificates for the software CA and tenant to mutually authenticate each other over an encrypted channel. To demonstrate the clean separation between the trusted computing layer and the software key management layer, we use the ZMQ Curve secure channel implementation [15]. This system use elliptic curve cryptography and other techniques that are dissimilar from the cryptographic algorithms, keys, and other techniques in use at the trusted computing layer. Despite these differences we can securely link both layers together.

To enroll a new node, the tenant first generates a node identity key pair using the software CA client. The client supports a plugin architecture that allows the tenant to specify what type of key pairs to create (our implementation currently supports RSA certificates). The tenant then connects securely to the software CA over ZMQ using the certificates provisioned earlier. The software CA supports a similar plugin architecture to the client that can sign the certificate with the appropriate private key and algorithm. Once the tenant has received the new node's signed certificate, he or she can provision a new node with this signed certificate and the corresponding public key using trusted-cloud-init. The software CA also supports receiving notifications from the CV if a node later fails integrity measurement. The software CA hosts a certificate revocation list that other services can check.

## 4.2 Demonstration Applications

We next describe how `keylime` can securely bootstrap and handle revocation for existing non-trusted computing-aware applications and services common to IaaS cloud deployments.

**IPsec** To enable secure network connectivity similar to TNC [38], we implemented cloud-init scripts to automatically encrypt all network traffic between a tenant's IaaS resources. The scripts use the OpenStack API to query for the tenant's subnets and public IP addresses and then build configurations for the Linux IPsec stack and `raccoon`[9]. We provision RSA keys from the software CA to enable `raccoon` to use the IKE protocol to establish an ephemeral AES key. This enables the security of the ephemeral keys encrypting packets between nodes to be directly linked back to the trusted computing layer.

**Puppet** Cloud operations personnel often use automated configuration management tools like Puppet, Chef, or Salt to configure their cloud resources. Furthermore, they often provision cryptographic identities, encryption keys, and other sensitive resources with these tools. This represents an opportunity for `keylime` to support both bootstrapping and runtime integrity checking. To demonstrate this, we have integrated `keylime` with Puppet. We do so by generating the signed RSA keys that Puppet uses to communicate with the Puppet master at the tenant prior to node creation and then send them to the newly created node using trusted-cloud-init (similarly to the software CA described earlier). These steps bypass the need to either use the insecure `autosign` option in the Puppet master to blindly accept new nodes or to have an operator manually approve/deny certificate signing requests from new nodes. To support continuous attestation and integrity measurement, we implemented a plug-in for the CV that will notify the tenant's Puppet master when a node fails its integrity measurements. The master can then revoke that node's access to check in and receive the latest configuration data.

**Vault** While tools like Puppet are often used to provision secrets and keys, the full life cycle support for the management of those secrets is not directly supported by Puppet and must be implemented ad hoc in the Puppet manifests. Tenant operators can instead use a dedicated secret management system that supports these operations directly. To demonstrate this, we have integrated `keylime` with Vault a cloud-compatible secret manager. Since Vault can use certificates to authenticate to the Vault Server, we can bootstrap the system similar to Puppet and the Software CA. We directly tie `keylime` to secret management using a CV plugin. The plugin notifies Vault to both revoke access to a node that fails integrity measurement and to re-generate any keys to which that node had access providing forward secrecy.

[9] `http://ipsec-tools.sourceforge.net/`

Table 1: Average TPM Operation Latency (milliseconds)

| | TPM | vTPM | DeepQuote |
|---|---|---|---|
| **Create Quote** | 757.4 | 89.69 | 1515.9 |
| **Check Quote** | 5.97 | 6.02 | 5.99 |

**LUKS** To demonstrate our ability to provision secrets directly apart from the provisioning of identities as we've already shown, we implemented a trusted-clout-init script that provides the key to unlock a LUKS volume on boot. In future work, we plan to enable this feature to work in a pre-boot environment so that we can decrypt a LUKS protected root partition. We also could integrate with the `overlayroot`[10] disk encryption system that automatically encrypts any data added to a static image.

## 5 Evaluation

In this section we evaluate the overhead and scalability of `keylime` in a variety of scenarios. We ran our experiments on a private OpenStack cluster, a Xen 4.5 host, and a Raspberry Pi 2. In OpenStack, we used standard instance flavors where the m1.small has 1 vCPU, 2GB RAM, and a 20GB disk, and the m1.large has 4 vCPUs, 8GB RAM, an an 80GB disk. We used Ubuntu Linux 14.10 as the guest OS in OpenStack instances. The Xen host had one Xeon E5-2640 CPU with 6 cores at 2.5Ghz, 10Gbit NIC, and 64 GB RAM and ran Xen 4.5 on top of Ubuntu Linux 15.04. The Xen host has a WinBond TPM. The Raspberry Pi 2 had one ARMv7 with 4-cores at 900Mhz,1GB RAM, an a 100Mbit NIC. The Pi ran Raspbian 7. We ran each of the following experiments for 1-2 minutes and present averages of the performance we observed.

**TPM Operations** We first establish a base line for the performance of TPM operations with the IBM client library and our Python wrapper code. We benchmarked both TPM quote creation and verification on the Xen host (Table 1). We collected the physical TPM measurements on the same system with a standard (non-Xen) kernel. We then collected both vTPM and Deep Quote measurements from a VM running on Xen. As expected, operations that require interaction with the physical TPM are very slow. Verification times, even for deep quotes that include two RSA signature verifications, are comparatively quick.

**Key Derivation Protocol** We next investigate the latency at the CV of different phases of our protocol. In Figure 8, we show the averaged results of hundreds of trails of the CV with a 100 cloud node vTPM equipped

[10] `http://blog.dustinkirkland.com/2012/08/introducing-overlayroot-overlayfs.html`

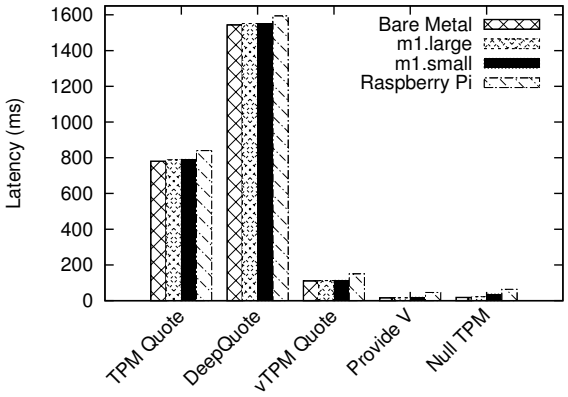


Figure 8: Latency of different stages of the `keylime` bootstrapping protocol.

VMs. We also benchmarked the latency of the protocol phases assuming zero latency from the TPM (Null TPM). We see that latency for the quote retrieval process is primarily affected by slow TPM operations. Virtual nodes doing runtime integrity measurement after bootstrapping benefit from much lower latency for vTPM operations. The results from the Null TPM trails indicate that our network protocol and other processing impose very little additional overhead, even on the relatively modestly powered Raspberry Pi.

**Scalability of Cloud Verifier** Next we want to establish the maximum rate at which the CV can get and check quotes for sustained runtime measurement. This will define the trade-off between the number of nodes a single CV can handle and the latency between when an integrity violation occurs and the CV detects it. To show the maximum throughput of the CV, we emulate a zero latency TPM in the cloud node by returning a pre-constructed quote. The CV still does all the standard processes to check the quote. We show the average number of quotes verified per second for each of our CV deployment options in Figure 9. The primary factor affecting CV scalability is the number of cores and RAM available. These options provide a variety of choices for deploying the CV. For small cloud tenants a low cost VM or dedicated hardware appliance can easily verify hundreds of virtual machines with moderate latency (5-10s). For larger customers, a larger cloud instance or dedicated hardware can scale to thousands with similar latency. For high security environments, either option can provide sub-second detection and response time. In future work, we plan to implement a priority scheme that allows the tenant to schedule the rate of verifications for different classes of nodes.

Lastly, we show how our CV architecture can scale by adding more cores and parallelism. We use the bare

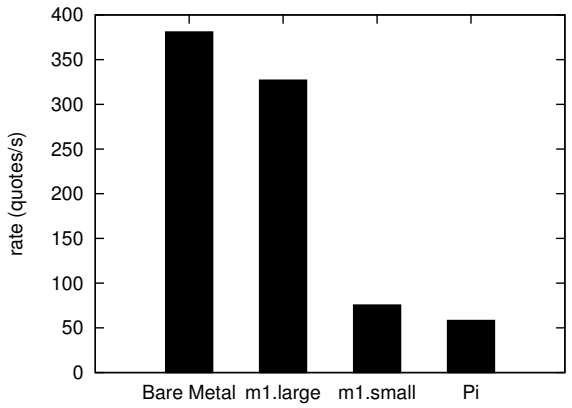


Figure 9: Maximum CV quote checking rate of `keylime` against 500 cloud nodes

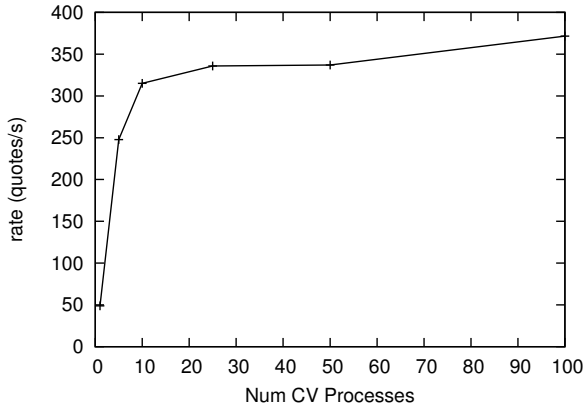


Figure 10: Parallelism of CV

metal CV and show the average maximum rate of quotes retrieved and checked per second for 500 nodes in Figure 10. We see linear speed-up until we exhaust the parallelism of the host CPU and the concurrent delay of waiting for many cloud nodes. In future work we plan to investigate how the design of an event-based architecture that uses the `select` system call could further improve scalability. However, our system is still fundamentally limited by the latency of the check quote routine. We also plan to implement that function in C to further improve performance.

**On-Premises Cloud Verifier** Finally, we investigate the performance of the CV when hosted at the tenant site away from the cloud. We show the results of our bare metal system's quote verification rate and the bandwidth used for a variety of network delays we inserted using the `comcast` tool[11] in Table 2. These results show that it is possible to run the CV on-premises at the tenant site with only moderate reduction in quote checking rate and *sim*1Mbit/s of bandwidth.

[11] `https://github.com/tylertreat/Comcast`

Table 2: On-Premises bare metal CV with 500 Cloud Nodes

| **Network RTT** (ms) | **Rate** (quotes/s) | **Bandwidth** (Kbits/s) |
|---|---|---|
| **25ms** | 302 | 994.2 |
| **50ms** | 283 | 931.6 |
| **75 ms** | 280 | 921.7 |
| **100 ms** | 259 | 852.6 |
| **150 ms** | 212 | 697.9 |

## 6 Related Work

Many of the challenges that exist in traditional enterprise networks exist in cloud computing environments as well. However, new challenges and threats exist due to several factors, including shared tenancy and additional reliance on the cloud provider to provide a secure foundation. Many of these challenges, both old and new, are surveyed in [5, 17]. In order to address some of the challenges, many have proposed trusted computing as a solution.

Many of the existing specifications for trusted computing rely on trusted hardware, and assume a single "owner" of the system. With the advent of cloud computing, this assumption is no longer valid. The creators of the TPM specification are actively working to extend support for trusted computing into these virtual environments [40]. Additionally, others have explored ways to virtualize the TPM, while still providing a hardware-backed root of trust for each virtual TPM [4]. In [8], another TPM extension for cloud environments, the authors assume a trustworthy cloud provider and modify existing trusted computing standards. Additionally, the question of scalability of trusted computing remains, as the TPM is slow, and placing it on the critical path will unacceptably limit throughput and increase latency.

To address the issues of scalability, a myriad of proposals exist to monitor a cloud infrastructure, and allow for validation of the virtual machines controlled by the tenants of the cloud [33, 32, 34]. The cloud verifier pattern proposed by Schiffman, et al., allows a single verifier to validate trust in the cloud nodes, and in turn the cloud verifier "vouches" for the integrity of the cloud nodes. This removes the need for tenants to validate the integrity of the cloud nodes prior to instantiating instances on a cloud node. The tenant simply provides their integrity verification criteria to the verifier, and the verifier ensures that the tenant's integrity criteria are satisfied as part of scheduling resources on the cloud nodes. These systems focus on validating integrity of the underlying cloud infrastructure, ignoring validation of higher-level services..

One proposal for higher-level validation of services provides a cryptographically signed audit trail that the hypervisor provides to auditors [13]. The audit trail captures the execution of the applications within the vir-

tual machine. This proposal does not provide a trusted foundation for the audit trail, and assumes a benign hypervisor. Another proposal looks at providing cryptographic protections of data for ZooKeeper, a distributed coordination service [6], again without a hardware root of trust. Bleikertz, et al., propose to use trusted computing provide cryptographic services for cloud tenants. Their Cryptography-as-a-Service (CaaS) system relies on trusted computing, but does not address the challenges of establishing initial keys and requires modifications to the hypervisor that cloud providers are unlikely to support.

The two works that are closest in nature to `keylime` are Excalibur [31] and CloudProxy Tao [25]. Excalibur aims to address the scalability problems of trusted computing by leveraging ciphertext policy attribute-based encryption (CPABE). This encryption scheme allows data to be encrypted using keys that represent attributes. Systems with matching attributes are able to decrypt the data. Using Excalibur, clients can encrypt sensitive data, and be assured that a node will only be given access to the data if the policy (the specified set of attributes) is satisfied. However, Excalibur does not provide integrity measurement, so a compromised node will not be detected until a new key is required to access data.

The CloudProxy system also provides building blocks to establish trusted services in a cloud environment. The Tao environment relies on trusted computing to establish integrity of the nodes and software in the system, and provide the building blocks for higher-level services. The current Tao system assumes a bare-metal OS running with trusted hardware, and is ill-suited for virtualized environments. Furthermore, many of the proposed trusted computing protocols *and* higher-level crypto services like secure storage and networking are non-standard, limiting the adoption. Our system provides a standardized mechanism to the bootstrap trusted computing layer and compatibility with existing higher-level services that rely on cryptographic keys.

## 7 Conclusion

In this paper, we have shown that `keylime` provides a fully integrated solution to bootstrap and maintain hardware-rooted trust in elastically provisioned IaaS clouds. We have demonstrated that we can securely bootstrap hardware-rooted cryptographic identities into cloud nodes, and leverage those identities in higher-level security services, without requiring each service to become trusted computing-aware, as is required by other proposals. `keylime` also uses a novel key derivation protocol that incorporates a tenant's *intent* to provision new cloud resources. Finally, we have demonstrated the myriad of deployment scenarios for our system's critical component, the cloud verifier. We performed detailed empirical analysis of those scenarios. Overall, `keylime` introduces as little as 2s of overhead during the provisioning process, and only requires 150ms to respond to an integrity violation. Furthermore, we have shown that `keylime` can scale to support thousands cloud nodes with reasonable overhead.

## References


[1] GRUB TCG Patch to Support Trusted Boot. `http://trousers.sourceforge.net/grub.html`.

[2] Intel cloud integrity technology. `http://www.intel.com/p/en_US/support/highlights/sftwr-prod/cit`, 2015.

[3] Shane Balfe and Anish Mohammed. Final fantasy – securing online gaming with trusted computing. In Bin Xiao, LaurenceT. Yang, Jianhua Ma, Christian Muller-Schloer, and Yu Hua, editors, *Autonomic and Trusted Computing*, volume 4610 of *Lecture Notes in Computer Science*, pages 123–134. Springer Berlin Heidelberg, 2007.

[4] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vtpm: Virtualizing the trusted platform module. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[5] Sara Bouchenak, Gregory Chockler, Hana Chockler, Gabriela Gheorghe, Nuno Santos, and Alexander Shraer. Verifying cloud services: Present and future. *SIGOPS Oper. Syst. Rev.*, 47(2):6–19, July 2013.

[6] Stefan Brenner, Colin Wulf, and Rüdiger Kapitza. Running zookeeper coordination services in untrusted clouds. In *10th Workshop on Hot Topics in System Dependability (HotDep 14)*, Broomfield, CO, October 2014. USENIX Association.

[7] Mark Burgess. Cfengine: a site configuration engine. In *USENIX Computing systems, Vol 8*, 1995.

[8] Chen Chen, Himanshu Raj, Stefan Saroiu, and Alec Wolman. ctpm: A cloud tpm for cross-device trusted applications. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, Seattle, WA, April 2014. USENIX Association.

[9] Xiaoxin Chen, Tal Garfinkel, E. Christopher Lewis, Pratap Subrahmanyam, Carl A. Waldspurger, Dan Boneh, Jeffrey Dwoskin, and Dan R.K. Ports. Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems. *SIGPLAN Not.*, 43(3):2–13, March 2008.

[10] John Criswell, Nathan Dautenhahn, and Vikram Adve. Virtual ghost: Protecting applications from hostile operating systems. *SIGARCH Comput. Archit. News*, 42(1):81–96, February 2014.

[11] Lucas Davi, Ahmad-Reza Sadeghi, and Marcel Winandy. Dynamic integrity measurement and attestation: towards defense against return-oriented programming attacks. In *Proceedings of the 2009 ACM workshop on Scalable trusted computing*, pages 49–54. ACM, 2009.

[12] Matthew Fioravante and Daniel De Graaf. vTPM. http://xenbits.xen.org/docs/unstable/misc/vtpm.txt, November 2012.

[13] Andreas Haeberlen, Paarijaat Aditya, Rodrigo Rodrigues, and Peter Druschel. Accountable virtual machines. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–16, Berkeley, CA, USA, 2010. USENIX Association.

[14] Jason Hennessey, Chris Hill, Ian Denhardt, Viggnesh Venugopal, George Silvis, Orran Krieger, and Peter Desnoyers. Hardware as a service - enabling dynamic, user-level bare metal provisioning of pools of data center resources. In *2014 IEEE High Performance Extreme Computing Conference*, Waltham, MA, USA, September 2014.

[15] Pieter Hintjens. Curvezmq authentication and encryption protocol. http://rfc.zeromq.org/spec:26, 2013.

[16] Owen S. Hofmann, Sangman Kim, Alan M. Dunn, Michael Z. Lee, and Emmett Witchel. Inktag: Secure applications on an untrusted operating system. *SIGPLAN Not.*, 48(4):265–278, March 2013.

[17] Wei Huang, Afshar Ganjali, Beom Heyn Kim, Sukwon Oh, and David Lie. The state of public infrastructure-as-a-service cloud security. *ACM Comput. Surv.*, 47(4):68:1–68:31, June 2015.

[18] IBM. Software trusted platform module. http://sourceforge.net/projects/ibmswtpm/, April 2014.

[19] Intel. Intel Trusted Boot (tboot). `https://software.intel.com/en-us/articles/intel-trusted-execution-technology`.

[20] Trent Jaeger, Reiner Sailer, and Umesh Shankar. Prima: Policy-reduced integrity measurement architecture. In *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies*, SACMAT '06, pages 19–28, New York, NY, USA, 2006. ACM.

[21] Bernhard Kauer. Oslo: Improving the security of trusted computing. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 16:1–16:9, Berkeley, CA, USA, 2007. USENIX Association.

[22] Ramakrishna Kotla, Tom Rodeheffer, Indrajit Roy, Patrick Stuedi, and Benjamin Wester. Pasture: Secure offline data access using commodity trusted hardware. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 321–334, Hollywood, CA, 2012. USENIX.

[23] P.G.J. Leelipushpam and J. Sharmila. Live vm migration techniques in cloud environment x2014; a survey. In *Information Communication Technologies (ICT), 2013 IEEE Conference on*, pages 408–413, April 2013.

[24] Peter A. Loscocco, Perry W. Wilson, J. Aaron Pendergrass, and C. Durward McDonell. Linux kernel integrity measurement using contextual inspection. In *Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing*, STC '07, pages 21–29, New York, NY, USA, 2007. ACM.

[25] John Manferdelli, Tom Roeder, and Fred Schneider. The cloud-proxy tao for trusted computing. Technical Report UCB/EECS-2013-135, EECS Department, University of California, Berkeley, Jul 2013.

[26] Petros Maniatis, Devdatta Akhawe, Kevin Fall, Elaine Shi, Stephen McCamant, and Dawn Song. Do you know where your data are?: Secure data capsules for deployable data protection. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, HotOS'13, pages 22–22, Berkeley, CA, USA, 2011. USENIX Association.

[27] David A McGrew and John Viega. The galois/counter mode of operation (gcm). *NIST*, 2005.

[28] Microsoft. BitLocker Drive Encryption Overview. `http://windows.microsoft.com/en-us/windows-vista/bitlocker-drive-encryption-overview`.

[29] Thomas Moyer, Kevin Butler, Joshua Schiffman, Patrick McDaniel, and Trent Jaeger. Scalable web content attestation. *IEEE Transactions on Computers*, Mar 2011.

[30] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

[31] Nuno Santos, Rodrigo Rodrigues, Krishna P. Gummadi, and Stefan Saroiu. Policy-sealed data: A new abstraction for building trusted cloud services. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 175–188, Bellevue, WA, 2012. USENIX.

[32] J. Schiffman, T. Moyer, C. Shal, T. Jaeger, and P. McDaniel. Justifying integrity using a virtual machine verifier. In *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, pages 83–92, Dec 2009.

[33] J. Schiffman, Yuqiong Sun, H. Vijayakumar, and T. Jaeger. Cloud verifier: Verifiable auditing service for iaas clouds. In *Services (SERVICES), 2013 IEEE Ninth World Congress on*, pages 239–246, June 2013.

[34] Joshua Schiffman, Thomas Moyer, Hayawardh Vijayakumar, Trent Jaeger, and Patrick McDaniel. Seeding clouds with trust anchors. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*, CCSW '10, pages 43–46, New York, NY, USA, 2010. ACM.

[35] Emin Gün Sirer, Willem de Bruijn, Patrick Reynolds, Alan Shieh, Kevin Walsh, Dan Williams, and Fred B. Schneider. Logical attestation: An authorization architecture for trustworthy computing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 249–264, New York, NY, USA, 2011. ACM.

[36] Andreas Steffen. The linux integrity measurement architecture and tpm-based network endpoint assessment. Technical report, Technical report, HSR University of Applied Sciences, 2012.

[37] Trusted Computing Group. TCG Infrastructure Working Group A CMC Profile for AIK Certificate Enrollment. `http://www.trustedcomputinggroup.org/files/resource_files/738DF0BB-1A4B-B294-D0AF6AF9CC023163/IWG_CMC_Profile_Cert_Enrollment_v1_r7.pdf`. Last Accessed: September 9, 2015.

[38] Trusted Computing Group. Trusted Network Communications. `http://www.trustedcomputinggroup.org/developers/trusted_network_communications`.

[39] Trusted Computing Group. Trusted Platform Module. `http://www.trustedcomputinggroup.org/developers/trusted_platform_module`.

[40] Trusted Computing Group. Virtualized Platform. `http://www.trustedcomputinggroup.org/developers/virtualized_platform`.

[41] Fengzhe Zhang, Jin Chen, Haibo Chen, and Binyu Zang. Cloudvisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 203–216, New York, NY, USA, 2011. ACM.