Keylime Documentation Documentation

Release 7.14.2

Keylime Developers

Jun 03, 2026

CONTENTS:

Warning

This documentation is still under development and not complete. It will be so until this warning is removed.

Welcome to the Keylime Documentation site!

Keylime is a TPM-based highly scalable remote boot attestation and runtime integrity measurement solution. Keylime enables cloud users to monitor remote nodes using a hardware based cryptographic root of trust.

Keylime was originally born out of the security research team in MIT's Lincoln Laboratory and is now developed and maintained by the Keylime community.

This Documentation site contains guides to install, use and administer keylime as well as guides to enable developers to make contributions to keylime or develop services against Keylime's Rest API(s).

We recommend newcomers to read the *design section* to get an understanding what the goals of Keylime are and how they are implemented.

CHAPTER

ONE

INSTALLATION

The current methods for installing Keylime are following: as a package (RHEL 9+, SLE Micro), the Ansible role, the Keylime installer or a manual installation.

1.1 RHEL 9 or later Installation

Please follow the documentation here.

1.2 SLE Micro Installation

Please follow the documetnation here.

1.3 Ansible Keylime Roles

An Ansible role to deploy Keylime , alongside the Keylime Rust agent

This role deploys Keylime for use with a Hardware TPM.

Should you wish to deploy Keylime with a software TPM emulator for development or getting your feet wet, use the Ansible Keylime Soft TPM role instead.

1.3.1 Usage

Download or clone Ansible Keylime from its repository and follow the usage section.

Run the example playbook against your target remote host(s):

```
ansible-playbook -i your_hosts playbook.yml
```

1.3.2 TPM Version Control (Software TPM)

Ansible Keylime Soft TPM provides a role type for 2.0 TPM versions.

TPM 2.0 support can be configured by simply adding the role in the `playbook.yml` file here

For TPM 2.0 use:

```
- ansible-keylime-tpm20
```

This rule uses the TPM 2.0 Emulator (IBM software TPM).

1.3.3 Rust agent

> **Note**
>
> The Rust agent is the official agent for Keylime and replaces the Python implementation. For the rust agent a different configuration file is used (by default `/etc/keylime/agent.conf`) which is **not** interchangeable with the old Python configuration.

Installation instructions can be found in the README.md for the Rust agent.

1.3.4 Push-model agent

> **Note**
>
> The push-model agent (`keylime-push-model-agent`) is a separate binary from the standard Rust agent. It implements the push attestation protocol where the agent initiates connections to the verifier. This feature is currently experimental.
>
> Installation instructions are the same as for the Rust agent. The push-model agent binary is built from the same repository. For configuration and deployment details, see the *Push-Model Attestation* user guide.

1.4 Keylime Bash installer

Keylime requires Python 3.6 or greater.

Installation can be performed via an automated shell script, `installer.sh`. The following command line options are available:

```
Usage: ./installer.sh [option...]
Options:
-k              Download Keylime (stub installer mode)
-m              Use modern TPM 2.0 libraries; this is the default
-s              Install & use a Software TPM emulator (development only)
-p PATH         Use PATH as Keylime path
-h              This help info
```

1.5 Docker - Deployment

The verifier, registrar and tenant can also be deployed using Docker images. Keylime's official images can be found here. Those are automatically generated for every commit and release.

For building those images locally see here.

1.6 Manual

Keylime requires Python 3.6 or greater.

1.6.1 Python-based prerequisites

The following Python packages are required:

- cryptography>=3.3.2
- tornado>=5.0.2
- pyzmq>=14.4
- pyyaml>=3.11
- requests>=2.6
- sqlalchemy>=1.3.12
- alembic>=1.1.0
- packaging>=20.0
- psutil>=5.4.2
- lark>=1.0.0
- pyasn1>=0.4.2
- pyasn1-modules>=0.2.1
- jinja2>=3.0.0
- gpg (Note: the GPG bindings must match the local GPG version and therefore this package should not be installed via PyPI)
- typing-extensions>=3.7.4 (only for Python versions < 3.8)

The current list of required packages can be found here.

All of them should be available as distro packages. See installer.sh for more information if you want to install them this way. You can also let Keylime's `setup.py` install them via PyPI.

1.6.2 TPM 2.0 Support

Keylime uses the Intel TPM2 software set to provide TPM 2.0 support. You will need to install the tpm2-tss software stack (available here) and tpm2-tools utilities available here. See README.md in these projects for detailed instructions on how to build and install.

The brief synopsis of a quick build/install (after installing dependencies) is:

```
# tpm2-tss
git clone https://github.com/tpm2-software/tpm2-tss.git tpm2-tss
pushd tpm2-tss
./bootstrap
./configure --prefix=/usr
make
sudo make install
popd
# tpm2-tools
git clone https://github.com/tpm2-software/tpm2-tools.git tpm2-tools
pushd tpm2-tools
./bootstrap
./configure --prefix=/usr/local
make
```

(continues on next page)

(continued from previous page)

```
sudo make install
popd
```

To ensure that you have the recent version installed ensure that you have the `tpm2_checkquote` utility in your path.

> **Note**
>
> Keylime by default (all versions after 6.2.0) uses the kernel TPM resource manager. For kernel versions older than 4.12 we recommend to use the tpm2-abrmd resource manager (available here).

How the TPM is accessed by tpm2-tools can be set using the `TPM2TOOLS_TCTI` environment variable. Keylime Rust agent understands `TCTI` without the prefix. More information about that can be found here.

Talk to the swtpm emulator directly:

```
export TPM2TOOLS_TCTI="mssim:port=2321"
export TCTI="mssim:port=2321"
```

To talk to the TPM directly (not recommended):

```
export TPM2TOOLS_TCTI="device:/dev/tpm0"
export TCTI="device:/dev/tpm0"
```

1.6.3 Install Keylime

You're finally ready to install Keylime:

```
sudo python setup.py install
```

1.7 Configuring basic (m)TLS setup

Keylime uses mTLS authentication between the different components. By default the verifier creates a CA for this under `/var/lib/keylime/cv_ca/` on first startup. The directory contains files for three different components:

- *Root CA*: `cacert.crt` contains the root CA certificate. **Important:** this certificate needs to be also be deployed on the agent, otherwise the tenant and verifier cannot connect to the agent!
- *Server certificate and key*: `server-cert.crt` and `server-{private,public}.pem` are used by the registrar and verifier for their HTTPS interface.
- *Client certificate and key*: `client-cert.crt` and `client-{private,public}.pem` are used by the tenant to authenticate against the verifier, registrar and agent. The verifier uses this key and certificate to authenticate against the agent.

Keylime allows each component to use their own server and client keys and also a list of trusted certificates for mTLS connections. Please refer to options the the respective configuration files for more details.

1.8 Database support

Keylime supports the following databases:

- SQLite
- PostgreSQL

- MySQL
- MariaDB

SQLite is configured as default (`database_url = sqlite`) where the databases are stored under `/var/lib/keylime`.

Starting with Keylime version 6.4.0 only supports SQLAlchemy's URL format to allow a more flexible configuration. The format for the supported databases can be found in the SQLAlchemy engine configuration documentation.

CHAPTER

TWO

USER GUIDE

2.1 Authentication

Most API interactions are secured using mTLS connections. By default there are two CAs involved, but the components can be configured to accommodate more complex setups.

(The revocation process also uses a CA, but this is different to those CAs)

In push mode, the PoP (Proof of Possession) authentication is mandatory for requests by agents for attestation operations.

Security Note: Never distribute client certificates signed by the verifier's trusted CA to agents. In push-attestation, agents should only authenticate using PoP tokens. In pull-model, agent should use its self-signed server certificate. If an agent had a valid client certificate AND didn't send an Authorization header, they would be identified as an admin by the verifier.

2.1.1 Server Components CA

This CA is created by verifier on startup. It contains the server certificates and keys used by the verifier and registrar for their respective HTTPS interfaces. Then it also contains the client certificates and keys that are used by the tenant to connect to the registrar, verifier and agent. Also the verifier uses that certificate to authenticate itself against the agent.

2.1.2 Agent Keylime CA

The agent runs an HTTPS server and provides its certificate to the registrar (`mtls_cert`).

The server component CA certificate is also required on the agent to authenticate connections from the tenant and verifier. By default `/var/lib/keylime/cv_ca/cacert.crt` is used.

2.2 Authorization Framework

Starting from version 7.14.0 (API version 2.5), Keylime implements a pluggable authorization framework to control access to API operations. The framework separates authentication (proving identity) from authorization (permission to perform actions).

2.2.1 Overview

Keylime's authorization framework is pluggable, allowing different authorization providers to implement various access control policies. Each provider can define its own rules for determining which identities can perform which operations.

2.2.2 Authorization Providers

The authorization provider is configured separately for each component:

Verifier Configuration

```
[verifier]
authorization_provider = simple
```

Registrar Configuration

```
[registrar]
authorization_provider = simple
```

Available Providers

- `simple` (default): Role-based access control with strict separation between agent and admin authentication methods

Future providers may include LDAP, OPA (Open Policy Agent), or custom implementations for enterprise deployments requiring fine-grained RBAC.

2.2.3 SimpleAuthProvider

The `simple` provider is the default authorization provider. It classifies operations into four categories:

- **Public operations**: No authentication required
  - Version information (`GET /versions`)
  - Server information (`GET /`)
  - Identity verification (`GET /verify/identity`)
  - Evidence verification (`POST /verify/evidence`)
  - Session creation for push attestation (`POST /sessions`)
  - Session update/extend (`PATCH /sessions/:session_id`)
- **Agent-only operations**: Requires PoP bearer token authentication
  - Agents can only access their own resources (identity must match resource)
  - Submit attestations (`POST /agents/:agent_id/attestations`)
- **Agent-or-admin operations**: Accessible by both roles with different scopes
  - `GET /agents/:agent_id` - Agents can read own status only, admins can read any agent
- **Admin operations**: Requires mTLS client certificate authentication
  - Full access to all management operations
  - Create/delete/update agents
  - Manage IMA and UEFI policies
  - View attestation results for any agent

The `simple` provider implements strict separation between agent and admin authentication methods:

Authentication Method Separation

Authorization Header	Authentication Path	Identity Type
Present (Bearer token)	Agent path	`agent` or `anonymous`
Absent	Admin path (mTLS)	`admin` or `anonymous`

Critical Security Rule: If an `Authorization` header is present in a request, the request is **always** treated as an agent authentication attempt. There is **no fallback** to mTLS authentication. This prevents privilege escalation attacks where an attacker might send an invalid bearer token while having a valid mTLS certificate.

Authorization Rules

1. **Public actions**: Always allowed regardless of identity
2. **Agent-only actions**: Requires `identity_type == "agent"` AND `identity == resource`
3. **Agent-or-admin actions**: Agent with `identity == resource` OR admin
4. **Admin actions**: Requires `identity_type == "admin"`

2.2.4 Certificate Requirements

The security model relies on proper certificate management:

Role	Authentication	Certificate Requirements
Agent (pull mode)	N/A (agent is server)	Self-signed server cert acceptable. If CA-issued, must have Server Auth EKU only.
Agent (push mode)	PoP bearer token	No client certs from trusted CA. Use PoP tokens only.
Admin/Tenant	mTLS client cert	Signed by verifier's trusted CA with Client Auth EKU.

Security Note: Never distribute client certificates signed by the verifier's trusted CA to agents. Agents should only authenticate using PoP tokens. If an agent had a valid client certificate AND didn't send an Authorization header, they would be identified as an admin.

Pull Mode Agent Certificates

Pull mode agents act as servers (the verifier connects to them). Their certificates have no security relevance for authorization because:

1. Trust is established via TPM quote, not the certificate
2. The agent's certificate is added to the verifier's client-side trust store only for that specific connection
3. The agent never connects to the verifier as a client

Self-signed certificates are acceptable for pull mode agents.

If the pull mode agent certificate is issued by the trusted CA (instead of self-signed), it **must have the Server Authentication EKU** (OID 1.3.6.1.5.5.7.3.1). This prevents the certificate from being used for client authentication, which would grant admin access.

2.2.5 Default Configuration (Development/Testing)

When the verifier is configured with `tls_dir = generate`, it automatically creates:

- A Certificate Authority (CA)
- Server certificates for the verifier
- **Client certificates for admin operations**

The automatically generated client certificate has:

- **Common Name (CN)**: `client`
- **Location**: `/var/lib/keylime/cv_ca/client-cert.crt`
- **Private Key**: `/var/lib/keylime/cv_ca/client-private.pem`

The tenant tool (used for admin operations) is configured to use this certificate by default:

```
[tenant]
tls_dir = default
client_cert = default      # Uses /var/lib/keylime/cv_ca/client-cert.crt
client_key = default       # Uses /var/lib/keylime/cv_ca/client-private.pem
```

What This Means

With default configuration:

- The verifier automatically generates all necessary certificates on startup
- The tenant tool automatically uses the generated client certificate
- Admin operations work **out-of-the-box** without manual certificate management
- This setup is suitable for **development, testing, and single-admin deployments**

2.2.6 Production Configuration (Multi-Admin Deployments)

For production deployments with multiple administrators, you should generate unique certificates for each administrator.

Step 1: Generate Admin Certificates

For each administrator, generate a unique certificate:

```
# Generate certificate for admin1
keylime_ca -d /var/lib/keylime/cv_ca create -n admin1.example.com

# Generate certificate for admin2
keylime_ca -d /var/lib/keylime/cv_ca create -n admin2.example.com
```

Step 2: Package and Distribute Certificates

Create certificate packages for distribution:

```
# Create package for admin1
keylime_ca -d /var/lib/keylime/cv_ca pkg -n admin1.example.com

# This creates: admin1.example.com-pkg.zip
```

Distribute the certificate package securely to each administrator (encrypted email, secure file transfer, etc.).

Step 3: Configure Tenant Tool

Each administrator should extract their certificate package and configure the tenant tool:

```
# Extract certificate package
unzip admin1.example.com-pkg.zip -d ~/keylime-certs/

# Configure tenant tool
cat > ~/.keylime/tenant.conf <<EOF
[tenant]
tls_dir = ~/keylime-certs/
client_cert = admin1.example.com-cert.crt
client_key = admin1.example.com-private.pem
trusted_server_ca = [cacert.crt]
EOF
```

2.2.7 Admin Operations Reference

The following operations require admin authentication (mTLS client certificate signed by the verifier's trusted CA):

Agent Management

- `GET /v3/agents` - List all agents
- `POST /v3/agents` - Create/enroll a new agent
- `GET /v3/agents/:agent_id` - View agent details (admin can view any agent)
- `PATCH /v3/agents/:agent_id` - Update agent configuration
- `DELETE /v3/agents/:agent_id` - Delete an agent
- `PUT /v3/agents/:agent_id/reactivate` - Reactivate an agent
- `PUT /v3/agents/:agent_id/stop` - Stop agent verification

IMA Policy Management

- `GET /v3/policies/ima` - List IMA policies
- `POST /v3/policies/ima` - Create IMA policy
- `GET /v3/policies/ima/:name` - View IMA policy
- `PATCH /v3/policies/ima/:name` - Update IMA policy
- `DELETE /v3/policies/ima/:name` - Delete IMA policy

UEFI/Measured Boot Policy Management

- `GET /v3/refstates/uefi` - List UEFI reference states
- `POST /v3/refstates/uefi` - Create UEFI reference state
- `GET /v3/refstates/uefi/:name` - View UEFI reference state
- `PATCH /v3/refstates/uefi/:name` - Update UEFI reference state
- `DELETE /v3/refstates/uefi/:name` - Delete UEFI reference state

Attestation Viewing (Push Mode)

- `GET /v3/agents/:agent_id/attestations` - View all attestations for an agent
- `GET /v3/agents/:agent_id/attestations/:index` - View specific attestation
- `GET /v3/agents/:agent_id/attestations/latest` - View latest attestation

2.2.8 Agent-Only Operations Reference

The following operations require agent authentication (PoP bearer token) and are exclusively for agents - admins cannot access these endpoints:

Attestation Submission (Push Mode)

- `POST /v3/agents/:agent_id/attestations` - Submit attestation (own agent_id only)
- `PATCH /v3/agents/:agent_id/attestations/:index` - Update attestation
- `PATCH /v3/agents/:agent_id/attestations/latest` - Update latest attestation

Note on Session Management

Session operations (`POST /v3/sessions`, `PATCH /v3/sessions/:session_id`) are public at the authorization layer. The session controller validates the PoP response or existing token internally. This allows agents to complete initial PoP authentication (when they don't yet have a token) and extend existing sessions.

2.2.9 Agent-or-Admin Operations Reference

The following operations are accessible to both agents and admins:

- `GET /v3/agents/:agent_id` - Agent can read own status (agent_id must match token), admin can read any agent

2.2.10 Security Considerations

Authentication Method Separation

The strict separation between bearer token and mTLS authentication is a critical security feature:

1. If an `Authorization` header is present, mTLS is **never** used
2. This prevents an attacker with both a valid mTLS cert and an expired/invalid token from falling back to admin authentication
3. Agents should **never** have client certificates signed by the trusted CA

Certificate Validation

The verifier validates admin mTLS certificates as follows:

1. **Certificate Trust**: The certificate must be signed by a CA trusted by the verifier (configured via `trusted_client_ca`)
2. **Certificate Validity**: The certificate must not be expired or revoked
3. **EKU Check**: The certificate must have Client Authentication EKU

2.2.11 Registrar Authorization

The registrar uses a simpler authorization model than the verifier. Since the registrar is primarily used for agent registration (which must be public) and administrative queries, it has only two categories of operations:

Public Operations (No Authentication Required)

- `POST /v2/agents` - Agent registration
- `POST /v2/agents/:agent_id` - Agent registration (legacy endpoint)
- `POST /v2/agents/:agent_id/activate` - Agent activation
- `PUT /v2/agents/:agent_id/activate` - Agent activation (legacy endpoint)
- `PUT /v2/agents/:agent_id` - Agent activation (legacy endpoint)

- `GET /version` - Version information

Admin Operations (Requires mTLS Client Certificate)

- `GET /v2/agents` - List all registered agents
- `GET /v2/agents/:agent_id` - View agent registration details
- `DELETE /v2/agents/:agent_id` - Delete agent registration

Registrar Security Model

Unlike the verifier, the registrar does not use PoP bearer tokens. The security model is:

1. **Public endpoints**: Agent registration endpoints are accessible without authentication because agents need to register before they have any credentials
2. **Admin endpoints**: All management operations (list, view, delete) require mTLS client certificate authentication

Configuration

The registrar has its own TLS configuration and can use a completely separate CA from the verifier. Admin operations require a client certificate signed by the CA specified in the registrar's `trusted_client_ca` configuration:

```
[registrar]
# Use a dedicated CA for the registrar
tls_dir = /var/lib/keylime/reg_ca
server_key = server-private.pem
server_cert = server-cert.crt
trusted_client_ca = [cacert.crt]

# Or use 'generate' to auto-generate certificates
# tls_dir = generate
```

For convenience during development and testing, the default configuration shares certificates with the verifier:

```
[registrar]
tls_dir = default              # Uses /var/lib/keylime/cv_ca
server_key = default           # Uses server-private.pem
server_cert = default          # Uses server-cert.crt
trusted_client_ca = default    # Uses [cacert.crt]
```

When using the default shared configuration, admin certificates valid for the verifier will also work for registrar operations. In production deployments with separate CAs, administrators need certificates from each component's respective CA.

2.2.12 Troubleshooting

Error: "Action requires admin authentication (mTLS certificate)"

Cause: The request doesn't have a valid mTLS client certificate.

Solution: Verify tenant configuration includes:

```
[tenant]
tls_dir = default  # Or path to certificate directory
client_cert = default  # Or specific certificate file
client_key = default   # Or specific key file
```

Ensure the certificate is signed by a CA in the verifier's `trusted_client_ca` list.

Error: "Action requires agent authentication (PoP token)"

Cause: The PoP token is missing on the request header. Maybe an admin (mTLS) is trying to access an agent-only endpoint.

Solution: Agent endpoints can only be accessed using PoP bearer tokens obtained through the push attestation authentication flow. Admins should use admin endpoints to view agent data.

Error: "Agent cannot access resource (ownership required)"

Cause: An agent is trying to access another agent's resources.

Solution: Agents can only access their own resources. The agent_id in the URL must match the agent_id associated with the bearer token.

Verifying Certificate EKU

To check the Extended Key Usage of a certificate:

```
openssl x509 -in /var/lib/keylime/cv_ca/client-cert.crt -noout -text | grep -A1
↪"Extended Key Usage"
# Should show: TLS Web Client Authentication
```

2.3 Configuration

Keylime is configured by files installed by default in `/etc/keylime`. The files are loaded following a hierarchical order in which the values set for the options can be overridden by the next level.

For each component, a base configuration file (e.g. `/etc/keylime/verifier.conf`) is loaded, setting the base values. Then, the configuration snippets placed in the respective directory (e.g. `/etc/keylime/verifier.conf.d`) are loaded, overriding the previously set values. Finally, options can be overridden via environment variables.

The following components can be configured:

Table 1: Components and configuration

Component	Default base config file	Default snippets directory
agent	`/etc/keylime/agent.conf`	`/etc/keylime/agent.conf.d`
verifier	`/etc/keylime/verifier.conf`	`/etc/keylime/verifier.conf.d`
registrar	`/etc/keylime/registrar.conf`	`/etc/keylime/registrar.conf.d`
tenant	`/etc/keylime/tenant.conf`	`/etc/keylime/tenant.conf.d`
ca	`/etc/keylime/ca.conf`	`/etc/keylime/ca.conf.d`
logging	`/etc/keylime/logging.conf`	`/etc/keylime/logging.conf.d`

Note

For push-model attestation, the verifier must be configured with `mode = push` in the `[verifier]` section. The push-model agent uses the same `/etc/keylime/agent.conf` file (TOML format) but with additional options such as `verifier_url` and `attestation_interval_seconds`. See *Push-Model Attestation* for details.

The next sections contain details of the configuration files

2.3.1 Configuration file processing order

The configurations are loaded in the following order:

1. Default (hardcoded) option values
2. Base configuration options, overriding previously set values
   - By default, located in `/etc/keylime/<component>.conf` for each component
3. Configuration snippets, overriding previously set values
   - By default, located in `/etc/keylime/<component>.conf.d` for each component
   - The configuration snippets are loaded in lexicographic order
4. Environment variables, overriding previously set values
   - The environment variables are in the form `KEYLIME_{COMPONENT}_[SECTION_]{OPTION}`, for example `KEYLIME_VERIFIER_SERVER_KEY`.

2.3.2 Configuration file format

The configuration files for all components are written in INI format, except for the agent which is written in TOML format.

Each component contains a main section named after the component name (for historical reasons). For example, the main section of `verifier.conf` is `[verifier]`.

2.3.3 Override configurations via configuration snippets

To override a configuration option using a configuration snippet, simply create a file in the component's configuration snippets directory (e.g. `/etc/keylime/verifier.conf.d` to override options from `/etc/keylime/verifier.conf`).

Note that the configuration snippets are loaded and processed in lexicographic order, keeping the last value set for each option. It is recommended to use a numeric prefix for the files to control the order in which they should be processed (e.g. `001-custom.conf`).

The configuration snippets need the section to be explicitly provided, meaning that the snippets are required to be valid INI (or TOML in the case of the agent) files.

For example, the following snippet placed in `/etc/keylime/verifier.conf.d/0001-ip.conf` can override the default verifier ip address:

```
[verifier]
ip = 172.30.1.10
```

For the verifier listener, the `ip` option can also be set to a wildcard address to bind to all interfaces. Use `0.0.0.0` for all IPv4 interfaces:

```
[verifier]
ip = 0.0.0.0
```

or `::` for all IPv6 interfaces (which on most dual-stack hosts also accepts incoming IPv4 connections):

```
[verifier]
ip = ::
```

These wildcard addresses only control where the verifier listens for incoming connections. Other components still need a concrete verifier address or DNS name when they connect to the verifier.

2.3.4 Override configurations via environment variables

It is possible to override configuration options by setting the desired value through environment variables. The environment variables are defined as `KEYLIME_{COMPONENT}_[SECTION_]{OPTION}`

The section can be omitted if the option to set is located in the main section (the section named after the component). Otherwise the section is required.

For example, to set the `webhook_url` option from the `[revocations]` section in the `verifier.conf` file, the environment variable to set is `KEYLIME_VERIFIER_REVOCATIONS_WEBHOOK_URL`.

To set an option located in the main section, for example the `server_key` option from the `[verifier]` section in the `verifier.conf`, the environment variable to set is `KEYLIME_VERIFIER_SERVER_KEY` (note that the section can be omitted).

2.3.5 Configuration upgrades

When updating keylime, it is also recommended to upgrade the configuration to make sure that the used configuration is compatible with the keylime version.

The `keylime_upgrade_config` script is installed by default with the keylime components, and is the script that performs the configuration upgrade following the upgrade mappings and templates.

By default, the configuration templates are installed in `/usr/share/keylime/templates`. For each configuration version a respective directory is installed in the templates directory.

The `keylime_upgrade_config` will take the current configuration files as input (by default from `/etc/keylime/`). For each component, the version of the configuration file is checked against the available upgrade template versions to decide if an upgrade is necessary or not. In case an upgrade is not necessary, meaning all the components configuration files are up-to-date, the script does not modify the configuration files.

For each component that needs upgrade, the script will process all the transformations needed by each configuration version. For example, suppose the installed configuration file is in version `1.0` and there are upgrade templates available for the versions `2.0` and `3.0`. The upgrade script will process the transformations defined from the version `1.0` to the version `2.0` and then from the version `2.0` to the version `3.0`.

For each configuration upgrade template version, there are the following files:

- `mapping.json`: This file defines the transformations that should be performed, mapping the configuration option from the older version to the configuration options in the new version. If the mapping has the `update` type, then it describes operations to transform the previous version to the next (adding, removing, or replacing options)
- `<component>.j2`: These are templates for the configuration files. It defines the output format for the configuration file for each component.
- `adjust.py`: This is an optional script that defines special adjustments that cannot be specified through the `mapping.json` file. It is executed after the mapping transformations are applied.

The main goal of the upgrade script is to keep the configuration changes made by the user and keep the configuration files up-to-date. For new options, the default values are used.

2.3.6 The configuration upgrade script `keylime_upgrade_script`

Run `keylime_upgrade_config --help` for the description of the supported options.

When executed by the `root` user, the default output directory for the `keylime_upgrade_config` script is the `/etc/keylime` directory. The existing configuration files are kept intact as backup and renamed with the `.bkp` extension appended to the file names.

In case the `--output` option is provided to the `keylime_upgrade_config` script, the configuration files are written even when they were already up-to-date using the available templates. It can be seen as a way to force the creation of the configuration files, fitting the options read into the new templates.

Passing the `--debug` option to the `keylime_upgrade_config`, the logging level is set to `DEBUG`, making the script more verbose.

The templates directory to be processed can be passed via the `--templates` option. If provided, the script will try to find the configuration upgrade templates in the provided path instead of the default location (`/usr/share/keylime/templates`)

To output files only for a subset of the components, the `--component` can be provided multiple times.

To override input files (by default the `/etc/keylime/<component>.conf` for each component), the `--input` option can be passed multiple times. Unknown components are ignored.

To stop the processing in a target version, set the target version with the `--version` option.

To ignore the input files and use the default value for all options, the `--defaults` option can be provided

Finally, to process a single mapping file, the mapping file path can be passed via the `--mapping` option

2.3.7 Attestation Models: Pull vs Push

Keylime supports two attestation models that determine how the verifier obtains attestation evidence from agents:

Pull Model (Traditional)

In the pull model, the verifier actively polls agents at regular intervals to retrieve attestation evidence. This is the default and traditional mode of operation.

Use Cases:

- Traditional deployments where the verifier can directly connect to agents
- Environments with stable network connectivity
- When you need fine-grained control over attestation frequency

Push Model (Agent-Driven)

In the push model, agents periodically push their attestation evidence to the verifier. This mode is useful when the verifier cannot directly connect to agents (e.g., agents behind firewalls or NAT).

Use Cases:

- Agents deployed behind firewalls or NAT
- Cloud or edge deployments where direct connectivity is limited
- When agents need to control their own attestation schedule

> **Note**
>
> The push model options were introduced in configuration version 2.5 and require the push attestation agent.

2.3.8 Configuration Options Reference

This section provides comprehensive tables of all configuration options for each Keylime component, including default values, environment variable overrides, and applicability to pull/push attestation models.

Verifier Configuration (`/etc/keylime/verifier.conf`)

Common Options (Both Models)

Option	Default	Version	Environment Variable
version	2.5	2.0	KEYLIME_VERIFIER_VERSION
uuid	default	2.0	KEYLIME_VERIFIER_UUID
ip	127.0.0.1	2.0	KEYLIME_VERIFIER_IP
port	8881	2.0	KEYLIME_VERIFIER_PORT
registrar_ip	127.0.0.1	2.0	KEYLIME_VERIFIER_REGISTRAR_IP
registrar_port	8891	2.0	KEYLIME_VERIFIER_REGISTRAR_PORT
enable_agent_mtls	True	2.0	KEYLIME_VERIFIER_ENABLE_AGENT_MTLS
tls_dir	generate	2.0	KEYLIME_VERIFIER_TLS_DIR
server_key	default	2.0	KEYLIME_VERIFIER_SERVER_KEY
server_key_password	(empty)	2.0	KEYLIME_VERIFIER_SERVER_KEY_PASSWORD
server_cert	default	2.0	KEYLIME_VERIFIER_SERVER_CERT
trusted_client_ca	default	2.0	KEYLIME_VERIFIER_TRUSTED_CLIENT_CA
client_key	default	2.0	KEYLIME_VERIFIER_CLIENT_KEY
client_key_password	(empty)	2.0	KEYLIME_VERIFIER_CLIENT_KEY_PASSWORD
client_cert	default	2.0	KEYLIME_VERIFIER_CLIENT_CERT
trusted_server_ca	default	2.0	KEYLIME_VERIFIER_TRUSTED_SERVER_CA
database_url	sqlite	2.0	KEYLIME_VERIFIER_DATABASE_URL
database_pool_sz_ovfl	5,10	2.0	KEYLIME_VERIFIER_DATABASE_POOL_SZ_OVFL
auto_migrate_db	True	2.0	KEYLIME_VERIFIER_AUTO_MIGRATE_DB
num_workers	0	2.0	KEYLIME_VERIFIER_NUM_WORKERS
max_workers	16	2.6	KEYLIME_VERIFIER_MAX_WORKERS
max_upload_size	104857600	2.0	KEYLIME_VERIFIER_MAX_UPLOAD_SIZE
measured_boot_policy_	accept-all	2.0	KEYLIME_VERIFIER_MEASURED_BOOT_POLICY_NAME
measured_boot_imports	[]	2.0	KEYLIME_VERIFIER_MEASURED_BOOT_IMPORTS
measured_boot_evaluat	once	2.0	KEYLIME_VERIFIER_MEASURED_BOOT_EVALUATE
severity_labels	["info", "notice", ...]	2.0	KEYLIME_VERIFIER_SEVERITY_LABELS
severity_policy	[{"event_i ".*", .. .}]	2.0	KEYLIME_VERIFIER_SEVERITY_POLICY
ignore_tomtou_errors	False	2.0	KEYLIME_VERIFIER_IGNORE_TOMTOU_ERRORS
durable_attestation_i	(empty)	2.0	KEYLIME_VERIFIER_DURABLE_ATTESTATION_IMPORT
persistent_store_url	(empty)	2.0	KEYLIME_VERIFIER_PERSISTENT_STORE_URL
transparency_log_url	(empty)	2.0	KEYLIME_VERIFIER_TRANSPARENCY_LOG_URL
time_stamp_authority_	(empty)	2.0	KEYLIME_VERIFIER_TIME_STAMP_AUTHORITY_URL
time_stamp_authority_	(empty)	2.0	KEYLIME_VERIFIER_TIME_STAMP_AUTHORITY_CERTS_PATH
persistent_store_form	json	2.0	KEYLIME_VERIFIER_PERSISTENT_STORE_FORMAT
persistent_store_enco	(empty)	2.0	KEYLIME_VERIFIER_PERSISTENT_STORE_ENCODING
transparency_log_sign	sha256	2.0	KEYLIME_VERIFIER_TRANSPARENCY_LOG_SIGN_ALGO
signed_attributes	(empty)	2.0	KEYLIME_VERIFIER_SIGNED_ATTRIBUTES
require_allow_list_si	False	2.0	KEYLIME_VERIFIER_REQUIRE_ALLOW_LIST_SIGNATURES
authorization_provide	simple	2.5	KEYLIME_VERIFIER_AUTHORIZATION_PROVIDER
cert_subject_alternat	(empty)	2.5	KEYLIME_VERIFIER_CERT_SUBJECT_ALTERNATIVE_NAMES
shutdown_drain_timeou	10	2.6	KEYLIME_VERIFIER_SHUTDOWN_DRAIN_TIMEOUT

Pull Model Specific Options

Option	Default	Version	Environment Variable
quote_interval	2	2.0	KEYLIME_VERIFIER_QUOTE_INTERVAL
retry_interval	2	2.0	KEYLIME_VERIFIER_RETRY_INTERVAL
max_retries	5	2.0	KEYLIME_VERIFIER_MAX_RETRIES
exponential_backoff	True	2.0	KEYLIME_VERIFIER_EXPONENTIAL_BACKOFF
request_timeout	60.0	2.0	KEYLIME_VERIFIER_REQUEST_TIMEOUT

Push Model Specific Options

Option	Default	Version	Environment Variable
mode	pull	2.5	KEYLIME_VERIFIER_MODE
challenge_lifetime	1800	2.5	KEYLIME_VERIFIER_CHALLENGE_LIFETIME
verification_timeout	0	2.5	KEYLIME_VERIFIER_VERIFICATION_TIMEOUT
session_create_rate_l	50	2.5	KEYLIME_VERIFIER_SESSION_CREATE_RATE_LIMIT_PER_IP
session_create_rate_l	60	2.5	KEYLIME_VERIFIER_SESSION_CREATE_RATE_LIMIT_WINDOW_IP
session_create_rate_l	15	2.5	KEYLIME_VERIFIER_SESSION_CREATE_RATE_LIMIT_PER_AGENT
session_create_rate_l	60	2.5	KEYLIME_VERIFIER_SESSION_CREATE_RATE_LIMIT_WINDOW_AGENT
session_lifetime	180	2.5	KEYLIME_VERIFIER_SESSION_LIFETIME
extend_token_on_attes	True	2.5	KEYLIME_VERIFIER_EXTEND_TOKEN_ON_ATTESTATION

Revocations Section

Option	Default	Version	Environment Variable
enabled_revocation_notific	['agent']	2.0	KEYLIME_VERIFIER_REVOCATIONS_ENABLED_REVOCATION_NOT
zmq_ip	127.0.0.1	2.0	KEYLIME_VERIFIER_REVOCATIONS_ZMQ_IP
zmq_port	8992	2.0	KEYLIME_VERIFIER_REVOCATIONS_ZMQ_PORT
webhook_url	(empty)	2.0	KEYLIME_VERIFIER_REVOCATIONS_WEBHOOK_URL

Registrar Configuration (/etc/keylime/registrar.conf)

Option	Default	Version	Environment Variable
version	2.5	2.0	KEYLIME_REGISTRAR_VERSION
ip	127.0.0.1	2.0	KEYLIME_REGISTRAR_IP
port	8890	2.0	KEYLIME_REGISTRAR_PORT
tls_port	8891	2.0	KEYLIME_REGISTRAR_TLS_PORT
tls_dir	default	2.0	KEYLIME_REGISTRAR_TLS_DIR
server_key	default	2.0	KEYLIME_REGISTRAR_SERVER_KEY
server_key_password	(empty)	2.0	KEYLIME_REGISTRAR_SERVER_KEY_PASSWORD
server_cert	default	2.0	KEYLIME_REGISTRAR_SERVER_CERT
trusted_client_ca	default	2.0	KEYLIME_REGISTRAR_TRUSTED_CLIENT_CA
database_url	sqlite	2.0	KEYLIME_REGISTRAR_DATABASE_URL
database_pool_sz_ovfl	5,10	2.0	KEYLIME_REGISTRAR_DATABASE_POOL_SZ_OVFL
max_workers	16	2.6	KEYLIME_REGISTRAR_MAX_WORKERS
auto_migrate_db	True	2.0	KEYLIME_REGISTRAR_AUTO_MIGRATE_DB
durable_attestation_import	(empty)	2.0	KEYLIME_REGISTRAR_DURABLE_ATTESTATION_IMPORT
persistent_store_url	(empty)	2.0	KEYLIME_REGISTRAR_PERSISTENT_STORE_URL
transparency_log_url	(empty)	2.0	KEYLIME_REGISTRAR_TRANSPARENCY_LOG_URL
time_stamp_authority_url	(empty)	2.0	KEYLIME_REGISTRAR_TIME_STAMP_AUTHORITY_URL
time_stamp_authority_certs	(empty)	2.0	KEYLIME_REGISTRAR_TIME_STAMP_AUTHORITY_CERTS_PATH
persistent_store_format	json	2.0	KEYLIME_REGISTRAR_PERSISTENT_STORE_FORMAT
persistent_store_encoding	(empty)	2.0	KEYLIME_REGISTRAR_PERSISTENT_STORE_ENCODING
transparency_log_sign_algo	sha256	2.0	KEYLIME_REGISTRAR_TRANSPARENCY_LOG_SIGN_ALGO
signed_attributes	ek_tpm, aik_tpm, ekcert	2.0	KEYLIME_REGISTRAR_SIGNED_ATTRIBUTES
tpm_identity	default	2.1	KEYLIME_REGISTRAR_TPM_IDENTITY
malformed_cert_action	warn	2.4	KEYLIME_REGISTRAR_MALFORMED_CERT_ACTION
authorization_provider	simple	2.5	KEYLIME_REGISTRAR_AUTHORIZATION_PROVIDER
cert_subject_alternative_r	(empty)	2.5	KEYLIME_REGISTRAR_CERT_SUBJECT_ALTERNATIVE_NAMES

Tenant Configuration (/etc/keylime/tenant.conf)

Option	Default	Version	Environment Variable
version	2.5	2.0	KEYLIME_TENANT_VERSION
verifier_ip	127.0.0.1	2.0	KEYLIME_TENANT_VERIFIER_IP
verifier_port	8881	2.0	KEYLIME_TENANT_VERIFIER_PORT
registrar_ip	127.0.0.1	2.0	KEYLIME_TENANT_REGISTRAR_IP
registrar_port	8891	2.0	KEYLIME_TENANT_REGISTRAR_PORT
tls_dir	default	2.0	KEYLIME_TENANT_TLS_DIR
enable_agent_mtls	True	2.0	KEYLIME_TENANT_ENABLE_AGENT_MTLS
client_key	default	2.0	KEYLIME_TENANT_CLIENT_KEY
client_key_password	(empty)	2.0	KEYLIME_TENANT_CLIENT_KEY_PASSWORD
client_cert	default	2.0	KEYLIME_TENANT_CLIENT_CERT
trusted_server_ca	default	2.0	KEYLIME_TENANT_TRUSTED_SERVER_CA
tpm_cert_store	/var/lib/keylime/tpm_cert_s	2.0	KEYLIME_TENANT_TPM_CERT_STORE
max_payload_size	1048576	2.0	KEYLIME_TENANT_MAX_PAYLOAD_SIZE
accept_tpm_hash_algs	['sha512', 'sha384', 'sha256']	2.0	KEYLIME_TENANT_ACCEPT_TPM_HASH_ALGS
accept_tpm_encryption_algs	['ecc', 'rsa']	2.0	KEYLIME_TENANT_ACCEPT_TPM_ENCRYPTION_ALGS
accept_tpm_signing_algs	['ecschnor 'rsassa']	2.0	KEYLIME_TENANT_ACCEPT_TPM_SIGNING_ALGS
exponential_backoff	True	2.0	KEYLIME_TENANT_EXPONENTIAL_BACKOFF
retry_interval	2	2.0	KEYLIME_TENANT_RETRY_INTERVAL
max_retries	5	2.0	KEYLIME_TENANT_MAX_RETRIES
request_timeout	60	2.0	KEYLIME_TENANT_REQUEST_TIMEOUT
require_ek_cert	True	2.0	KEYLIME_TENANT_REQUIRE_EK_CERT
ek_check_script	(empty)	2.0	KEYLIME_TENANT_EK_CHECK_SCRIPT
mb_refstate	(empty)	2.0	KEYLIME_TENANT_MB_REFSTATE

CA Configuration (/etc/keylime/ca.conf)

Option	Default	Version	Environment Variable
version	2.5	2.0	KEYLIME_CA_VERSION
password	default	2.0	KEYLIME_CA_PASSWORD
cert_country	US	2.0	KEYLIME_CA_CERT_COUNTRY
cert_ca_name	Keylime Certificate Authority	2.0	KEYLIME_CA_CERT_CA_NAME
cert_state	MA	2.0	KEYLIME_CA_CERT_STATE
cert_locality	Lexington	2.0	KEYLIME_CA_CERT_LOCALITY
cert_organization	Keylime	2.0	KEYLIME_CA_CERT_ORGANIZATION
cert_org_unit	53	2.0	KEYLIME_CA_CERT_ORG_UNIT
cert_ca_lifetime	3650	2.0	KEYLIME_CA_CERT_CA_LIFETIME
cert_lifetime	365	2.0	KEYLIME_CA_CERT_LIFETIME
cert_bits	2048	2.0	KEYLIME_CA_CERT_BITS
cert_crl_dist	http://localhost:38 crl	2.0	KEYLIME_CA_CERT_CRL_DIST

Agent Configuration (/etc/keylime/agent.conf)

Warning

The Python agent is deprecated and will be removed in version 7.0.0! Please migrate to the Rust-based agent from https://github.com/keylime/rust-keylime/

Common Options (Both Models)

Option	Default	Version	Environment Variable
version	"2.5"	2.0	KEYLIME_AGENT_VERSION
api_versions	"default"	2.4	KEYLIME_AGENT_API_VERSIONS
uuid	"d432fbb3-	2.0	KEYLIME_AGENT_UUID
ip	"127.0.0.1"	2.0	KEYLIME_AGENT_IP
port	9002	2.0	KEYLIME_AGENT_PORT
contact_ip	"127.0.0.1"	2.0	KEYLIME_AGENT_CONTACT_IP
contact_port	9002	2.0	KEYLIME_AGENT_CONTACT_PORT
registrar_ip	"127.0.0.1"	2.0	KEYLIME_AGENT_REGISTRAR_IP
registrar_port	8890	2.0	KEYLIME_AGENT_REGISTRAR_PORT
enable_agent_mtls	true	2.0	KEYLIME_AGENT_ENABLE_AGENT_MTLS
tls_dir	"default"	2.0	KEYLIME_AGENT_TLS_DIR
server_key	"default"	2.0	KEYLIME_AGENT_SERVER_KEY
server_key_password	""	2.0	KEYLIME_AGENT_SERVER_KEY_PASSWORD
server_cert	"default"	2.0	KEYLIME_AGENT_SERVER_CERT
trusted_client_ca	"default"	2.0	KEYLIME_AGENT_TRUSTED_CLIENT_CA
enc_keyname	"derived_t	2.0	KEYLIME_AGENT_ENC_KEYNAME

continues on next page

Table 3 – continued from previous page

Option	Default	Version	Environment Variable
dec_payload_file	"decrypted	2.0	KEYLIME_AGENT_DEC_PAYLOAD_FILE
secure_size	"1m"	2.0	KEYLIME_AGENT_SECURE_SIZE
tpm_ownerpassword	""	2.0	KEYLIME_AGENT_TPM_OWNERPASSWORD
extract_payload_zip	true	2.0	KEYLIME_AGENT_EXTRACT_PAYLOAD_ZIP
enable_revocation_notifi	true	2.0	KEYLIME_AGENT_ENABLE_REVOCATION_NOTIFICATIONS
revocation_notification_	"127.0.0.1"	2.0	KEYLIME_AGENT_REVOCATION_NOTIFICATION_IP
revocation_notification_	8992	2.0	KEYLIME_AGENT_REVOCATION_NOTIFICATION_PORT
revocation_cert	"default"	2.0	KEYLIME_AGENT_REVOCATION_CERT
revocation_actions	"[]"	2.0	KEYLIME_AGENT_REVOCATION_ACTIONS
payload_script	"autorun.sh"	2.0	KEYLIME_AGENT_PAYLOAD_SCRIPT
enable_insecure_payload	false	2.0	KEYLIME_AGENT_ENABLE_INSECURE_PAYLOAD
measure_payload_pcr	-1	2.0	KEYLIME_AGENT_MEASURE_PAYLOAD_PCR
exponential_backoff	true	2.0	KEYLIME_AGENT_EXPONENTIAL_BACKOFF
retry_interval	2	2.0	KEYLIME_AGENT_RETRY_INTERVAL
max_retries	4	2.0	KEYLIME_AGENT_MAX_RETRIES
tpm_hash_alg	"sha256"	2.0	KEYLIME_AGENT_TPM_HASH_ALG
tpm_encryption_alg	"rsa"	2.0	KEYLIME_AGENT_TPM_ENCRYPTION_ALG
tpm_signing_alg	"rsassa"	2.0	KEYLIME_AGENT_TPM_SIGNING_ALG
ek_handle	"generate'	2.0	KEYLIME_AGENT_EK_HANDLE
enable_iak_idevid	false	2.1	KEYLIME_AGENT_ENABLE_IAK_IDEVID
iak_idevid_template	"detect"	2.1	KEYLIME_AGENT_IAK_IDEVID_TEMPLATE
iak_idevid_asymmetric_al	"rsa"	2.1	KEYLIME_AGENT_IAK_IDEVID_ASYMMETRIC_ALG
iak_idevid_name_alg	"sha256"	2.1	KEYLIME_AGENT_IAK_IDEVID_NAME_ALG
idevid_password	""	2.3	KEYLIME_AGENT_IDEVID_PASSWORD
idevid_handle	""	2.3	KEYLIME_AGENT_IDEVID_HANDLE
iak_password	""	2.3	KEYLIME_AGENT_IAK_PASSWORD
iak_handle	""	2.3	KEYLIME_AGENT_IAK_HANDLE
iak_cert	"default"	2.1	KEYLIME_AGENT_IAK_CERT
idevid_cert	"default"	2.1	KEYLIME_AGENT_IDEVID_CERT
run_as	"keylime:t	2.0	KEYLIME_AGENT_RUN_AS
ima_ml_path	"default"	2.2	KEYLIME_AGENT_IMA_ML_PATH
measuredboot_ml_path	"default"	2.2	KEYLIME_AGENT_MEASUREDBOOT_ML_PATH

Push Model Specific Options

Option	Default	Version	Environment Variable
agent_data_path	"/var/lib/keylime/agent_data.json"	2.5	KEYLIME_AGENT_AGENT_DATA_PATH
verifier_url	"https://localhost:	2.5	KEYLIME_AGENT_VERIFIER_URL
certification_keys_server_iden	"ak"	2.5	KEYLIME_AGENT_CERTIFICATION_KEYS_SERVER_IDENTIFIE
uefi_logs_evidence_version	"1.0"	2.5	KEYLIME_AGENT_UEFI_LOGS_EVIDENCE_VERSION
tls_accept_invalid_certs	false	2.5	KEYLIME_AGENT_TLS_ACCEPT_INVALID_CERTS
tls_accept_invalid_hostnames	false	2.5	KEYLIME_AGENT_TLS_ACCEPT_INVALID_HOSTNAMES
exponential_backoff_max_retrie	5	2.5	KEYLIME_AGENT_EXPONENTIAL_BACKOFF_MAX_RETRIES
exponential_backoff_initial_de	10000	2.5	KEYLIME_AGENT_EXPONENTIAL_BACKOFF_INITIAL_DELAY
exponential_backoff_max_delay	360000	2.5	KEYLIME_AGENT_EXPONENTIAL_BACKOFF_MAX_DELAY

Logging Configuration (`/etc/keylime/logging.conf`)

The logging configuration follows Python's standard logging configuration format. See the Python logging documentation for details on configuring handlers, formatters, and loggers. The version option can be overridden with `KEYLIME_LOGGING_VERSION`.

2.3.9 Configuration Version History

Version	Changes
2.0	Base configuration structure, pull model support
2.1	Added IAK/IDevID support, `tpm_identity` for registrar
2.2	Added `ima_ml_path` and `measuredboot_ml_path` configuration
2.3	Added persisted key handles for IAK/IDevID (`iak_handle`, `idevid_handle`)
2.4	Added `api_versions` for agent, `malformed_cert_action` for registrar
2.5	**Push model support**: Added `mode`, `challenge_lifetime`, `verification_timeout`, session rate limiting and lifetime options for verifier; `verifier_url`, `agent_data_path`, TLS validation, exponential backoff options for agent. Added `authorization_provider` and `cert_subject_alternative_names` for verifier and registrar
2.6	Added `shutdown_drain_timeout` for verifier graceful shutdown

For detailed information on all configuration options for each component, refer to the configuration files in `/etc/keylime/` and their inline documentation.

2.4 Push-Model Attestation

> **Warning**
>
> Push-model attestation is currently experimental. The feature is functional but the API and configuration options may change in future releases.

2.4.1 Introduction

In the default pull model, the Keylime verifier continuously polls agents for attestation data. This requires the verifier to reach the agent over the network.

The push model reverses this: the agent initiates connections to the verifier and proactively sends attestation evidence. This is useful when the verifier cannot directly reach the agent, for example behind firewalls, NAT, or in edge/IoT deployments.

For a detailed description of how push-model attestation works, see *Push-Model Attestation*.

2.4.2 Prerequisites

- Keylime verifier and registrar installed and running
- The `keylime-push-model-agent` binary installed on the target machine
- A TPM 2.0 device (hardware or emulated for development)
- Network connectivity **from the agent to the verifier and registrar** (the reverse is not required)
- The verifier's CA certificate available on the agent machine

2.4.3 Configuring the Verifier for Push Mode

Set the verifier's attestation mode to `push` in `/etc/keylime/verifier.conf`:

```
[verifier]
mode = push
```

Or use a configuration snippet in `/etc/keylime/verifier.conf.d/`:

```
# /etc/keylime/verifier.conf.d/001-push-mode.conf
[verifier]
mode = push
```

The verifier can also be configured via environment variable:

```
export KEYLIME_VERIFIER_MODE=push
```

Note

The `mode` setting affects all agents on this verifier. A verifier in push mode expects agents to submit attestation data; it does not poll agents. A single verifier cannot operate in both modes simultaneously.

Additional verifier settings relevant to push mode:

- `quote_interval`: Used to calculate the agent timeout threshold (`quote_interval * 5`). Default: `2` seconds.
- `challenge_lifetime`: How long a challenge nonce remains valid for evidence submission.
- `verification_timeout`: Maximum time allowed for evidence verification.

After changing the configuration, restart the verifier:

```
sudo systemctl restart keylime_verifier
```

2.4.4 Configuring the Push-Model Agent

The push-model agent is a separate binary from the standard Keylime agent. It is installed as `keylime_push_model_agent` (or `keylime-push-model-agent`).

The agent is configured through `/etc/keylime/agent.conf` (TOML format), command-line arguments, or environment variables.

Key Configuration Options

The following options are specific to or particularly important for push-model operation:

Option	Default	Description
`verifier_url`	`https://localhost:88`	URL of the verifier. Must use HTTPS.
`verifier_tls_ca_cert`	`cv_ca/cacert.crt`	Path to the CA certificate for verifying the verifier's TLS certificate. Relative paths are resolved from `keylime_dir`.
`attestation_interval_secor`	`60`	Interval in seconds between attestation cycles.
`registrar_ip`	`127.0.0.1`	IP address of the registrar.
`registrar_port`	`8890`	Port of the registrar.
`registrar_tls_enabled`	`false`	Enable TLS for registrar communication.
`registrar_tls_ca_cert`	`cv_ca/cacert.crt`	CA certificate for registrar TLS verification.
`uuid`	(generated)	Agent UUID. Can be a specific UUID, `generate` (random), or `hash_ek` (derived from the EK).
`api_versions`	`3.0`	API versions supported by the agent. Defaults to `3.0` for push model.
`tpm_hash_alg`	`sha256`	TPM hash algorithm (`sha256`, `sha384`, `sha512`).
`tpm_signing_alg`	`rsassa`	TPM signing algorithm (`rsassa`, `ecdsa`).
`keylime_dir`	`/var/lib/keylime`	Working directory for certificates and data files.

Example Minimal Configuration

```
# /etc/keylime/agent.conf (push-model agent)
[agent]
uuid = "d432fbb3-d2f1-4a97-9ef7-75bd81c00000"
verifier_url = "https://verifier.example.com:8881"
verifier_tls_ca_cert = "/var/lib/keylime/cv_ca/cacert.crt"
attestation_interval_seconds = 60
registrar_ip = "registrar.example.com"
registrar_port = 8890
tpm_hash_alg = "sha256"
tpm_signing_alg = "rsassa"
```

Command-Line Arguments

The push-model agent accepts the following command-line arguments, which override configuration file values:

```
--verifier-url <URL>                    Verifier URL (required)
--registrar-url <URL>                   Registrar URL (default: http://127.0.0.1:8888)
--agent-identifier <ID>                 Agent UUID
--attestation-interval-seconds <SECS>   Attestation interval (default: 60)
```

(continues on next page)

(continued from previous page)

```
--ca-certificate <PATH>                CA certificate for TLS verification
--api-version <VERSION>                API version (default: v3.0)
--timeout <MS>                         Request timeout in milliseconds (default: 5000)
--insecure                             Accept invalid TLS certificates (testing only)
--avoid-tpm                            Use mock TPM (testing only)
```

Exponential Backoff

When the agent encounters errors (network failures, verifier unavailable), it uses exponential backoff for retries:

Option	Default	Description
exponential_backoff_initial_de	10000	Initial delay in milliseconds (10 seconds)
exponential_backoff_max_retrie	5	Maximum number of retry attempts
exponential_backoff_max_delay	300000	Maximum delay in milliseconds (5 minutes)

2.4.5 Systemd Service Management

The push-model agent is managed as a systemd service:

```
# Enable the service to start on boot
sudo systemctl enable keylime_push_model_agent

# Start the service
sudo systemctl start keylime_push_model_agent

# Check service status
sudo systemctl status keylime_push_model_agent

# View logs
sudo journalctl -u keylime_push_model_agent -f
```

> **Warning**
>
> The push-model agent service (`keylime_push_model_agent.service`) conflicts with the standard pull-model agent service (`keylime_agent.service`). Only one can run at a time on the same machine. Starting one will stop the other.

The service is configured to restart on failure with a 120-second delay between restart attempts.

2.4.6 Enrolling an Agent for Push-Model Attestation

Use the `keylime_tenant` tool with the `--push-model` flag to enroll an agent for push-model attestation:

```
# Add an agent in push mode
sudo keylime_tenant -c add --push-model -u <agent-uuid>

# Add with a runtime IMA policy
sudo keylime_tenant -c add --push-model -u <agent-uuid> \
    --runtime-policy-name <policy-name>
```

(continues on next page)

(continued from previous page)

```
# Add with a measured boot policy
sudo keylime_tenant -c add --push-model -u <agent-uuid> \
    --mb-policy-name <policy-name>
```

Note

In push mode, the `-t` / `--targethost` option is not required because the verifier does not need to connect to the agent. The agent's IP and port are set to `None` in the verifier's database.

To check the status of a push-model agent:

```
sudo keylime_tenant -c cvstatus -u <agent-uuid>
```

To remove an agent:

```
sudo keylime_tenant -c delete -u <agent-uuid>
```

2.4.7 TLS Configuration for Push Model

The push model uses TLS differently from the pull model:

Agent-to-verifier connection:

- The agent connects to the verifier over HTTPS
- The agent verifies the verifier's server certificate using the configured CA certificate (`verifier_tls_ca_cert`)
- The agent does **not** present a client certificate (no mTLS)
- Authentication is done via PoP bearer tokens (see *Authentication*)

Agent-to-registrar connection:

- The agent connects to the registrar to register itself
- TLS can be enabled with `registrar_tls_enabled = true`
- The registrar CA certificate is configured with `registrar_tls_ca_cert`

Firewall considerations:

- No inbound ports need to be opened on the agent machine
- The agent needs outbound access to the verifier port (default: 8881)
- The agent needs outbound access to the registrar port (default: 8890)

To set up TLS, copy the verifier's CA certificate to the agent machine:

```
# On the verifier machine, the CA cert is typically at:
# /var/lib/keylime/cv_ca/cacert.crt

# Copy to the agent machine:
scp verifier:/var/lib/keylime/cv_ca/cacert.crt /var/lib/keylime/cv_ca/cacert.crt
```

2.4.8 Verifying the Deployment

After starting both the verifier (in push mode) and the push-model agent:

1. **Check agent registration** in the registrar:

   ```
   sudo keylime_tenant -c regstatus -u <agent-uuid>
   ```

2. **Check attestation status** in the verifier:

   ```
   sudo keylime_tenant -c cvstatus -u <agent-uuid>
   ```

3. **View verifier logs** for attestation activity:

   ```
   sudo journalctl -u keylime_verifier -f
   ```

   Successful attestations will show evidence receipt and verification completion messages.

4. **View agent logs** for attestation cycles:

   ```
   sudo journalctl -u keylime_push_model_agent -f
   ```

   The agent logs will show transitions through the state machine: registration, negotiation, and attestation phases.

2.4.9 Troubleshooting

Agent cannot connect to verifier

- Verify the `verifier_url` is correct and uses HTTPS
- Check that the verifier is running and listening on the configured port
- Verify network connectivity from the agent to the verifier
- Check that the CA certificate (`verifier_tls_ca_cert`) matches the verifier's server certificate

Agent shows timeout failures

The verifier marks an agent as failed if it does not receive an attestation within `quote_interval * 5` seconds.

- Verify the `attestation_interval_seconds` on the agent is less than the verifier's timeout threshold
- Check for network instability between agent and verifier
- Review agent logs for errors during attestation cycles

PoP authentication errors

- Ensure the agent is properly registered in the registrar (the AK must be known)
- Check that the TPM is accessible and functioning
- Verify the agent UUID matches between agent configuration and verifier enrollment

Agent state stuck in Negotiating

- The verifier may be rejecting capabilities. Check verifier logs for error details
- Ensure the TPM algorithms configured on the agent are accepted by the verifier
- Check that the `api_versions` setting includes `3.0`

Service fails to start

- Check that the pull-model agent service is not running (`systemctl status keylime_agent`)
- Verify the configuration file syntax (TOML format)
- Check file permissions on TLS certificates and TPM device

2.5 Runtime Integrity Monitoring

Keylime's runtime integrity monitoring requires the set up of Linux IMA. More information about IMA in general can be found in the openSUSE Wiki.

You should refer to your Linux Distributions documentation to enable IMA, but as a general guide most recent versions already have `CONFIG_IMA` toggled to `Y` as a value during Kernel compile.

It is then just a case of deploying an `ima-policy` file. On a Fedora or Debian system, the file is located in `/etc/ima/ima-policy`.

For configuration of your IMA policy, please refer to the IMA Documentation.

Within Keylime we use the following for demonstration (found in `demo/ima-policies/ima-policy-keylime`):

```
# PROC_SUPER_MAGIC
dont_measure fsmagic=0x9fa0
# SYSFS_MAGIC
dont_measure fsmagic=0x62656572
# DEBUGFS_MAGIC
dont_measure fsmagic=0x64626720
# TMPFS_MAGIC
dont_measure fsmagic=0x01021994
# RAMFS_MAGIC
dont_measure fsmagic=0x858458f6
# SECURITYFS_MAGIC
dont_measure fsmagic=0x73636673
# SELINUX_MAGIC
dont_measure fsmagic=0xf97cff8c
# CGROUP_SUPER_MAGIC
dont_measure fsmagic=0x27e0eb
# OVERLAYFS_MAGIC
# when containers are used we almost always want to ignore them
dont_measure fsmagic=0x794c7630
# Don't measure log, audit or tmp files
dont_measure obj_type=var_log_t
dont_measure obj_type=auditd_log_t
dont_measure obj_type=tmp_t
# MEASUREMENTS
measure func=BPRM_CHECK
measure func=FILE_MMAP mask=MAY_EXEC
measure func=MODULE_CHECK uid=0
```

This default policy measures all executables in `bprm_check` and all files `mmapped` executable in `file_mmap` and module checks and skips several irrelevant files (logs, audit, tmp, etc).

Once your `ima-policy` is in place, reboot your machine (or even better have it present in your image for first boot).

You can then verify IMA is measuring your system:

```
# head -5 /sys/kernel/security/ima/ascii_runtime_measurements
PCR                                template-hash filedata-hash                              ␣
↪         filename-hint
10 3c93cea361cd6892bc8b9e3458e22ce60ef2e632 ima-ng␣
↪sha1:ac7dd11bf0e3bec9a7eb2c01e495072962fb9dfa boot_aggregate
10 3d1452eb1fcbe51ad137f3fc21d3cf4a7c2e625b ima-ng␣
↪sha1:a212d835ca43d7deedd4ee806898e77eab53dafa /usr/lib/systemd/systemd
10 e213099a2bf6d88333446c5da617e327696f9eb4 ima-ng␣
↪sha1:6da34b1b7d2ca0d5ca19e68119c262556a15171d /usr/lib64/ld-2.28.so
10 7efd8e2a3da367f2de74b26b84f20b37c692b9f9 ima-ng␣
↪sha1:af78ea0b455f654e9237e2086971f367b6bebc5f /usr/lib/systemd/libsystemd-shared-239.so
10 784fbf69b54c99d4ae82c0be5fca365a8272414e ima-ng␣
↪sha1:b0c601bf82d32ff9afa34bccbb7e8f052c48d64e /etc/ld.so.cache
```

2.5.1 Keylime Runtime Policies

A runtime policy in its most basic form is a set of "golden" cryptographic hashes of files' un-tampered state or of keys that may be loaded onto keyrings for IMA verification.

Keylime will load the runtime policy into the Keylime Verifier. Keylime will then poll tpm quotes to *PCR 10* on the agents TPM and validate the agents file(s) state against the policy. If the object has been tampered with or an unexpected key was loaded onto a keyring, the hashes will not match and Keylime will place the agent into a failed state. Likewise, if any files invoke the actions stated in `ima-policy` that are not matched in the allowlist, keylime will place the agent into a failed state.

Allowlists are contained in Keylime runtime policies - see below for more details.

Generate a Runtime Policy

Runtime policies heavily depend on the IMA configuration and used files by the operating system. Keylime provides two helper scripts for getting started.

> **Note**
>
> Those scripts only provide a reference point to get started and **not** a complete solution. We encourage developers / users of Keylime to be creative and come up with their own process for securely creating and maintaining runtime policies.

Create Runtime Policy from a Running System

The first script generates a runtime policy from the `initramfs`, IMA log (just for the `boot aggregate`) and files located on the root filesystem of a running system.

The `create_runtime_policy.sh` script is available here

Run the script as follows:

```
# create_runtime_policy.sh -o runtime_policy_keylime.json
```

For more options see the help page `create_runtime_policy.sh`:

```
Usage: $0 -o/--output_file FILENAME [-a/--algo ALGO] [-x/--ramdisk-location PATH] [-y/--
↪boot_aggregate-location PATH] [-z/--rootfs-location PATH] [-e/--exclude_list FILENAME]␣
↪[-s/--skip-path PATH]"
```

(continues on next page)

(continued from previous page)

```
optional arguments:
-a/--algo                    (checksum algorithmi to be used, default: sha1sum)
-x/--ramdisk-location        (path to initramdisk, default: /boot/, set to "none" to␣
↪skip)
-y/--boot_aggregate-location (path for IMA log, used for boot aggregate extraction,␣
↪default: /sys/kernel/security/ima/ascii_runtime_measurements, set to "none" to skip)
-z/--rootfs-location         (path to root filesystem, default: /, cannot be skipped)
-e/--exclude_list            (filename containing a list of paths to be excluded (i.e.,␣
↪verifier will not try to match checksums), default: none)
-s/--skip-path               (comma-separated path list, files found there will not have␣
↪checksums calculated, default: none)
-h/--help                    show this message and exit
```

Note: note, you need the OpenSSL installed to have the sha*sum CLI executables available

The resulting *runtime_policy_keylime.json* file can be directly used by `keylime_tenant` (option `--runtime-policy`)

Warning

It's best practice to create the runtime policy in a secure environment. Ideally, this should be on a fully encrypted, air gapped computer that is permanently isolated from the Internet. Disable all network cards and sign the runtime policy hash to ensure no tampering occurs when transferring to other machines.

Creating more Complex Policies

The second script allows the user to build more complex policies by providing options to include: keyring verification, IMA verification keys, generating allowlist from IMA measurement log and extending existing policies.

A basic policy can be easily created by using a IMA measurement log from system:

```
keylime_create_policy -m /path/to/ascii_runtime_measurements -o runtime_policy.json
```

For more options see the help page `keylime_create_policy -h`:

```
usage: keylime_create_policy [-h] [-B BASE_POLICY] [-k] [-b] [-a ALLOWLIST] [-m IMA_
↪MEASUREMENT_LIST] [-i IGNORED_KEYRINGS] [-o OUTPUT] [--no-hashes] [-A IMA_SIGNATURE_
↪KEYS]

This is an experimental tool for adding items to a Keylime's IMA runtime policy

options:
  -h, --help            show this help message and exit
  -B BASE_POLICY, --base-policy BASE_POLICY
                        Merge new data into the given JSON runtime policy
  -k, --keyrings        Create keyrings policy entries
  -b, --ima-buf         Process ima-buf entries other than those related to keyrings
  -a ALLOWLIST, --allowlist ALLOWLIST
                        Use given plain-text allowlist
  -m IMA_MEASUREMENT_LIST, --ima-measurement-list IMA_MEASUREMENT_LIST
                        Use given IMA measurement list for keyrings and critical data␣
↪extraction rather than /sys/kernel/security/ima/ascii_runtime_measurements
```

(continues on next page)

(continued from previous page)

```
  -i IGNORED_KEYRINGS, --ignored-keyrings IGNORED_KEYRINGS
                        Ignored the given keyring; this option may be passed multiple␣
↪times
  -o OUTPUT, --output OUTPUT
                        File to write JSON policy into; default is to print to stdout
  --no-hashes           Do not add any hashes to the policy
  -A IMA_SIGNATURE_KEYS, --add-ima-signature-verification-key IMA_SIGNATURE_KEYS
                        Add the given IMA signature verification key to the Keylime-
↪internal 'tenant_keyring'; the key should be an x509 certificate in DER or PEM format␣
↪but may also be a public or private key
                        file; this option may be passed multiple times
```

Runtime Policy Entries for Keys

IMA can measure which keys are loaded onto different keyrings. Keylime has the option to verify those keys and automatically use them for signature verification.

The hash of the an key can be generated for example with:

```
sha256sum /etc/keys/ima/rsakey-rsa.crt.der
```

As seen the the JSON schema below, the hash (sha1 or sha256) depending on the IMA configuration can be added as the following where in `.ima` is the keyring the key gets loaded onto and <SHA256_HASH> is the hash of that key:

```
jq '.keyrings += {".ima" : ["<SHA256_HASH>"]}'  runtime_policy.json  > runtime_policy_
↪with_keyring.json
```

The following rule should be added to the IMA policy so that IMA reports keys loaded onto keyrings .ima and .evm (since Linux 5.6):

```
measure func=KEY_CHECK keyrings=.ima|.evm
```

If the key should only be verified and not be used for IMA signature verification, then it can be added to the ignore list:

```
jq '.ima.ignored_keyrings += [".ima"]' runtime_policy.json > runtime_policy_ignore_ima.
↪json
```

If * is added no verified keyring is used for IMA signature verification.

Runtime Policy JSON Schema

The tenant parses the allow and exclude list into a JSON object that is then sent to the verifier. Depending of the use case the object can also be constructed manually instead of using the tenant.

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "title": "Keylime Runtime policy",
    "type": "object",
    "properties": {
        "meta": {
            "type": "object",
            "properties": {
                "generator": {
                    "type": "integer",
```

(continues on next page)

(continued from previous page)

```
                    "description": "Identifier of the origin of the policy."
                },
                "version": {
                    "type": "integer",
                    "description": "Version number of the IMA policy schema"
                }
            },
            "required": ["version", "generator"],
            "additionalProperties": false
        },
        "release": {
            "type": "number",
            "title": "Release version",
            "description": "Version of the IMA policy (arbitrarily chosen by the user)"
        },
        "digests": {
            "type": "object",
            "title": "File paths and their digests",
            "patternProperties": {
                ".*": {
                    "type": "array",
                    "title": "Path of a valid file",
                    "items": {
                        "type": "string",
                        "title": "Hash of an valid file"
                    }
                }
            }
        },
        "excludes": {
            "type": "array",
            "title": "Regular expression strings to match file paths to exclude",
            "items": {
                "type": "string",
                "format": "regex"
            }
        },
        "keyrings": {
            "type": "object",
            "patternProperties": {
                ".*": {
                    "type": "string",
                    "title": "Hash of the content in the keyring"
                }
            }
        },
        "ima-buf": {
            "type": "object",
            "title": "Validation of ima-buf entries",
            "patternProperties": {
                ".*": {
                    "type": "string",
```

(continues on next page)

(continued from previous page)

```
                    "title": "Hash of the ima-buf entry"
                }
            }
        },
        "verification-keys": {
            "type": "string",
            "title": "A JSON encoded IMA Keyring object",
            "description": "A JSON encoded IMA Keyring object. It contains public keys␣
↪used to verify IMA attached signatures",
        },
        "ima": {
            "type": "object",
            "title": "IMA validation configuration",
            "properties": {
                "ignored_keyrings": {
                    "type": "array",
                    "title": "Ignored keyrings for key learning",
                    "description": "The IMA validation can learn the used keyrings␣
↪embedded in the kernel. Use '*' to never learn any key from the IMA keyring␣
↪measurements",
                    "items": {
                        "type": "string",
                        "title": "Keyring name"
                    }
                },
                "log_hash_alg": {
                    "type": "string",
                    "title": "IMA entry running hash algorithm",
                    "description": "The hash algorithm used for the running hash in IMA␣
↪entries (second value). The kernel currently hardcodes it to sha1.",
                    "const": "sha1"
                }
            },
            "required": ["ignored_keyrings", "log_hash_alg"],
            "additionalProperties": false
        }
    },
    "required": ["meta", "release", "digests", "excludes", "keyrings", "ima", "ima-buf",
↪"verification-keys"],
    "additionalProperties": false
}
```

IMA Keyring JSON Schema

The *verification-keys* field of the Runtime Policy is a JSON encoded IMA Keyring object. The following schema can be used to validate it:

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "title": "IMA Keyring JSON schema",
    "type": "object",
    "properties": {
```

(continues on next page)

(continued from previous page)

```
        "meta": {
            "type": "object",
            "required": ["version"],
            "properties": {
                "version": {"type": "integer", "minimum": 1},
            },
        },
        "keyids": {
            "type": "array",
            "description": "Lowest 4 bytes of the SHA-1 hash over the DER encoded public␣
↪key",
            "title": "Identifier of the public keys.",
            "items": {"type": "integer"}
        },
        "pubkeys": {
            "type": "array",
            "description": "An array of Base64 encoded public keys in DER format"
            "title": "The keyring public keys"
            "items": {"type": "string"}
        },
    },
    "required": ["keyids", "pubkeys"],
    "additionalProperties": false
}
```

2.5.2 Remotely Provision Agents

Now that we have our runtime policy available, we can send it to the verifier.

> **Note**
>
> If you're using a TPM Emulator (for example with the ansible-keylime-tpm-emulator, you will also need to run the keylime ima emulator. To do this, open a terminal and run `keylime_ima_emulator`

Using the `keylime_tenant` we can send the runtime policy as follows:

```
keylime_tenant -c add --uuid <agent-uuid> --runtime-policy /path/to/policy.json
```

> **Note**
>
> If your agent is already registered, you can use `-c update`

2.5.3 How can I test this?

Create a script that does anything (for example `echo "hello world"`) that is not present in your runtime policy. Run the script as root on the agent machine. You will then see the following output on the verifier showing the agent status change to failed:

```
keylime.tpm - INFO - Checking IMA measurement list...
keylime.ima - WARNING - File not found in allowlist: /root/evil_script.sh
```

(continues on next page)

(continued from previous page)

```
keylime.ima - ERROR - IMA ERRORS: template-hash 0 fnf 1 hash 0 good 781
keylime.cloudverifier - WARNING - agent D432FBB3-D2F1-4A97-9EF7-75BD81C00000 failed,␣
↪stopping polling
```

2.5.4 IMA File Signature Verification

Keylime supports the verification of IMA file signatures, which also helps to detect modifications on immutable files and can be used to complement or even replace the allowlist of hashes in the runtime policy if all relevant executables and libraries are signed. However, the set up of a system that has *all* files signed is beyond the scope of this documentation.

In the following we will show how files can be signed and how a system with signed files must be registered. We assume that the system has already been set up for runtime-integrity monitoring following the above steps and that the system would not show any errors on the Keylime Verifier side. It should not be registered with the keylime verifier at this point. If it is, we now deregister it:

```
keylime_tenant -c delete -u <agent-uuid>
```

Our first step is to enable IMA Appraisal in Linux. Recent Fedora kernels for example have IMA Appraisal support built-in but not activated. To enable it, we need to add the following Linux kernel parameters to the Linux boot command line:

```
ima_appraise=fix ima_template=ima-sig ima_policy=tcb
```

For this we edit */etc/default/grub* and append the above parameters to the *GRUB_CMDLINE_LINUX* line and then recreate the system's grub configuration file with the following command:

```
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
```

IMA will be in IMA Appraisal fix-mode when the system is started up the next time. Fix-mode, unlike enforcement mode, does not require that all files be signed but will give us the benefit that the verifier receives all file signatures of signed executables.

For IMA Appraisal to append the file signatures to the IMA log, we need to append the following line to the above IMA policy:

```
appraise func=BPRM_CHECK fowner=0 appraise_type=imasig
```

We now create our IMA file signing key using the following commands:

```
openssl genrsa -out ima-filesigning.pem 2048
openssl rsa -in ima-filesigning.pem -pubout -out ima-pub.pem
```

Next, we determine the hash (sha1 or sha256) that IMA is using for file measurements by looking at the IMA measurement log and then use evmctl to sign a demo executable that we derive from the echo tool:

```
sudo dnf -y install ima-evm-utils
cp /bin/echo ./myecho
sudo evmctl ima_sign --key ima-filesigning.pem -a <hash> myecho
```

Note

It is important that we use the same hash for signing the file that IMA also uses for file measurements. In the case we use 'sha1' since the IMA measurement log further above shows sha1 filedata-hashes in the 4th column. On

more recent systems we would likely use 'sha256'.

Note

If the IMA measurement log contains invalid signatures, the system will have to be rebooted to start over with a clean log that the Keylime Verifier can successfully verify.

Invalid signatures may for example be in the log if executables were accidentally signed with the wrong hash, such as sha1 instead of sha256. In this case they all need to be re-signed to match the hash that IMA is using for file signatures.

Another reason for an invalid signature may be that a file was modified after it was signed. Any file modification will invalidate the signature. Similarly, a malformatted or altered *security.ima* extended attribute will lead to a signature verification failure.

Yet another reason may be that an unknown key was used for signing files. In this case the system should be re-registered with that additional key using the Keylime tenant tool.

To verify that the file has been properly signed, we can use the following command, which will show the security.ima extended attribute's value:

```
getfattr -m ^security.ima --dump myecho
```

We now reboot the machine:

```
reboot
```

After the reboot the IMA measurement log should not have any measurement of the *myecho* tool. The following command should not return anything:

```
grep myecho /sys/kernel/security/ima/ascii_runtime_measurements
```

We now create a new policy that includes the signing key using the `keylime_create_policy` tool:

```
keylime_create_policy -B /path/to/runtime_policy.json -A /path/to/ima-pub.pem  -o /
↪output/path/runtime_policy_with_key.json
```

After that we register the agent with the new policy:

```
keylime_tenant -c add --uuid <agent-uuid> -f payload --runtime-policy /path/to/runtime_
↪policy_with_key.json
```

We can now execute the `myecho` tool as root:

```
sudo ./myecho
```

At this point we should not see any errors on the verifier side and there should be one entry of 'myecho' in the IMA measurement log that contains a column after the file path containing the file signature:

```
grep myecho /sys/kernel/security/ima/ascii_runtime_measurements
```

To test that signature verification works, we can now invalidate the signature by *appending* a byte to the file and executing it again:

```
echo >> ./myecho
sudo ./myecho
```

We should now see two entries in the IMA measurement log. Each one should have a different measurement:

```
grep myecho /sys/kernel/security/ima/ascii_runtime_measurements
```

The verifier log should now indicating a bad file signature:

```
keylime.tpm - INFO - Checking IMA measurement list on agent: D432FBB3-D2F1-4A97-9EF7-
↪75BD81C00000
keylime.ima - WARNING - signature for file /home/test/myecho is not valid
keylime.ima - ERROR - IMA ERRORS: template-hash 0 fnf 0 hash 0 bad-sig 1 good 3042
keylime.cloudverifier - WARNING - agent D432FBB3-D2F1-4A97-9EF7-75BD81C00000 failed,␣
↪stopping polling
```

2.5.5 Using Key Learning to Verify Files

Note: The following has been tested with RHEL 9.3 and keylime 7.3. It is work-in-progress on CentOS and Fedora.

Using key learning to verify files requires that files logged by IMA are appropriately signed. If files are not signed or have a bad signature then they must be either in the exclude list of the runtime policy or their hashes must be part of the runtime policy. It should also be noted that IMA signature verification provides lock-down of a system and ensures the provenance of files from a trusted source but, unlike file hashes, does not provide protection for file renaming or replacing files and signatures with other versions (downgrading).

For the following setup we use RHEL 9.3 since this distribution carries file signatures in its rpm packages and the Dracut scripts have been added to load the IMA signature verification keys onto the `.ima` keyring.

All below steps are run as *root*.

To ensure that file signatures are installed when packages are installed, run the following command:

```
dnf -y install rpm-plugin-ima
```

Since some packages did not carry file signatures until recently, update all packages to ensure that the signatures are installed:

```
dnf -y update
```

In case the system was previously not installed with file signatures, run the following command to reinstall all packages with file signatures:

```
dnf -y reinstall \*
```

To verify whether a particular file has its file signature installed use the following command to display the contents of `security.ima`. If nothing is displayed then this file misses its file signature:

```
getfattr -m ^security.ima -e hex --dump /usr/bin/bash
```

We must setup the system with the kernel command line option `ima_template=ima-sig` so that IMA signatures become part of the measurement log. It is not necessary to enable signature enforcement on the system, measuring executed applications is sufficient for the purpose of 'key learning'. For this we edit `/etc/default/grub` and adjust the following line:

```
GRUB_CMDLINE_LINUX="rhgb quiet ima_template=ima-sig"
```

Then run the following command to update the kernel command line options:

```
grub2-mkconfig -o /boot/grub2/grub.conf   # grub.cfg on CentOS/RHEL
```

Set the following IMA policy in `/etc/ima/ima-policy` when systemd will load the policy:

```
# PROC_SUPER_MAGIC
dont_measure fsmagic=0x9fa0
# SYSFS_MAGIC
dont_measure fsmagic=0x62656572
# DEBUGFS_MAGIC
dont_measure fsmagic=0x64626720
# TMPFS_MAGIC
dont_measure fsmagic=0x01021994
# RAMFS_MAGIC
dont_measure fsmagic=0x858458f6
# SECURITYFS_MAGIC
dont_measure fsmagic=0x73636673
# SELINUX_MAGIC
dont_measure fsmagic=0xf97cff8c
# CGROUP_SUPER_MAGIC
dont_measure fsmagic=0x27e0eb
# OVERLAYFS_MAGIC
# when containers are used we almost always want to ignore them
dont_measure fsmagic=0x794c7630

# Measure and log keys loaded onto the .ima keyring
measure func=KEY_CHECK keyrings=.ima
# Measure and log executables
measure func=BPRM_CHECK
# Measure and log shared libraries
measure func=FILE_MMAP mask=MAY_EXEC
```

Copy IMA signature verification key(s) so that Dracut scripts can load the keys onto the `.ima` keyring early during system startup:

```
mkdir -p /etc/keys/ima
cp /usr/share/doc/kernel-keys/$(uname -r)/ima.cer /etc/keys/ima # RHEL/CentOS
```

Enable the IMA Dracut scripts in the initramfs:

```
dracut --kver $(uname -r) --force --add integrity
```

Then reboot the system:

```
reboot
```

Once the system has been rebooted it must show at least two entries in the IMA log where keys were loaded onto the .ima keyring:

grep -E “ .ima “ /sys/kernel/security/ima/ascii_runtime_measurements

The first entry represents the Linux kernel signing key and the second entry is the IMA file signing key.

We now create the policy:

```
grep \
  -E "(boot_aggregate| ima-buf )" \
  /sys/kernel/security/ima/ascii_runtime_measurements > trimmed_ima_log

keylime_create_policy -k -m ./trimmed_ima_log -o mypolicy.json
```

The 1st command creates a trimmed-down IMA measurement log that only contains the boot_aggregate and ima-buf entries. The latter show the key(s) that were loaded onto the `.ima` keyring.

The 2nd command creates the runtime policy that holds the boot_aggregate entry and a hash over keys that were loaded onto the .ima keyring. This hash is used to verify that only trusted keys are learned.

We can now start to monitor this system:

```
touch payload  # create empty payload for example purposes
keylime_tenant -c update --uuid <agent-uuid> -f payload --runtime-policy ./mypolicy.json
```

In case the verification of the system fails we need to inspect the verifier log and add those files to the `trimmed_ima_log` that failed verification. Assuming files with the filename pattern `livesys` failed verification we repeat the steps above as follows by adding files with the file pattern `livesys` to the trimmed log. These files will then be verified using their hashes rather than signatures. Another possibility would be to add these files to the list of excluded files. We may need to repeat the following steps until the system passes verification:

```
grep \
  -E "(boot_aggregate| ima-buf |livesys)" \
  /sys/kernel/security/ima/ascii_runtime_measurements > trimmed_ima_log

keylime_create_policy -k -m ./trimmed_ima_log -o mypolicy.json

keylime_tenant -c update --uuid <agent-uuid> -f payload --runtime-policy ./mypolicy.json
```

To trigger a verification failure an unsigned application can be started:

```
cat <<EOF > test.sh
#!/usr/bin/env bash
echo Test
EOF

chmod 0755 test.sh

./test.sh
```

To re-enable the verification of the system the policy needs to be updated to contain `test.sh` and possibly all other applications that are not signed:

grep
: -E “(boot_aggregate| ima-buf |test.sh)” /sys/kernel/security/ima/ascii_runtime_measurements > trimmed_ima_log

keylime_create_policy -k -m ./trimmed_ima_log -o mypolicy.json

keylime_tenant -c update –uuid <agent-uuid> -f payload –runtime-policy ./mypolicy.json

2.5.6 Legacy allowlist and excludelist Format

Since Keylime 6.6.0 the old JSON and flat file formats for runtime policies are deprecated. Keylime provides with `keylime_convert_runtime_policy` a utility to convert those into the new format.

2.6 User Selected PCR Monitoring

> **Warning**
>
> This page is still under development and not complete. It will be so until this warning is removed.

Using use the *tpm_policy* feature in Keylime, it is possible to monitor a remote machine for any given PCR.

This can be used for Trusted Boot checks for both the *rhboot* shim loader and Trusted Grub 2.

> **Note**
>
> On larger deployments the PCR values might be insufficient. In this case use the UEFI event log for measured boot: *Use Measured Boot*.

2.6.1 How to use

Select which PCRs you would like Keylime to measure, by using the *tpm2_pcrread* from the tpm2-tools tool.

You can add a node to using *keylime_tenant*:

```
# Now actually add the node
keylime_tenant -c add \
--uuid d432fbb3-d2f1-4a97-9ef7-75bd81c00000 \
--tpm_policy '{"22":["0000000000000000000000000000000000000001",
↪"0000000000000000000000000000000000000000000000000000000000000001",
↪"00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪","ffffffffffffffffffffffffffffffffffffffff",
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff",
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"],"15":["0000000000000000000000000000000000000000",
↪"0000000000000000000000000000000000000000000000000000000000000000",
↪"00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"]}'
```

2.6.2 rhboot shim-loader

The following is sourced from the rhboot shim repository please visit the upstream README to ensure information is still accurate

The following PCRs are extended by shim:

PCR4:

- the Authenticode hash of the binary being loaded will be extended into PCR4 before SB verification.
- the hash of any binary for which Verify is called through the shim_lock protocol

PCR7:

- Any certificate in one of our certificate databases that matches a binary we try to load will be extended into PCR7. That includes:
  - DBX - the system denylist, logged as “dbx”
  - MokListX - the Mok denylist, logged as “MokListX”
  - vendor_dbx - shim’s built-in vendor denylist, logged as “dbx”
  - DB - the system allowlist, logged as “db”
  - MokList the Mok allowlist, logged as “MokList”
  - vendor_cert - shim’s built-in vendor allowlist, logged as “Shim”
  - shim_cert - shim’s build-time generated allowlist, logged as “Shim”
- MokSBState will be extended into PCR7 if it is set, logged as “MokSBState”.

PCR8:

- If you’re using the grub2 TPM patchset we cary in Fedora, the kernel command line and all grub commands (including all of grub.cfg that gets run) are measured into PCR8.

PCR9:

- If you’re using the grub2 TPM patchset we cary in Fedora, the kernel, initramfs, and any multiboot modules loaded are measured into PCR9.

PCR14:

- MokList, MokListX, and MokSBState will be extended into PCR14 if they are set.

2.7 Use Measured Boot

> **Warning**
>
> This page is still under development and not complete. It will be so until this warning is removed.

2.7.1 Introduction

In any real-world large-scale production environment, a large number of different types of nodes will typically be found. The TPM 2.0 defines a specific meaning - measurement of UEFI bios, measurement of boot device firmware - for each of the lower-numbered PCRs (e.g., PCRs 0-9), as these are extended during the multiple events of a measured boot log. However, simply comparing the contents of these PCRs against a well-known “golden value” becomes unfeasible. The reason for this is, in addition to the potentially hundreds of variations due to node type, it can be experimentally demonstrated that some PCRs (e.g, PCR 1) vary for each physical machine, if such machine is netbooted (as it encodes the MAC address of the NIC used during boot.)

Fortunately, the UEFI firmware is now exposing the event log through an ACPI table and a “recent enough” Linux kernel (e.g., 5.4 or later) is now consuming this table and exposing this boot event log through the securityfs, typically at the path */sys/kernel/security/tpm0/binary_bios_measurements*. When combined with *secure boot* and a “recent enough” version of grub (2.06 or later), the aforementioned PCR set can be fully populated, including measurements of all components, up to the *kernel* and *initrd*.

In addition to these sources of (boot log) data, a “recent enough” version of *tpm2-tools* (5.0 or later) can be used to consume the contents of such logs and thus rebuild the contents of PCRs [0-9] (and potentially PCRs [11-14]).

2.7.2 Implementation

Keylime can make use of this new capability in a very flexible manner. It can allow the keylime operator (i.e. tenant) to provide a measured boot policy for a keylime agent node and let the keylime verifier evaluate the policy against the boot event log of that node. In theory, the measured boot policy could contain both the reference state (i.e. reference data about the various boot events) of the node and the rule/policy to evaluate the boot event log against that reference state.

Keylime, at present, allows the operator to provide the reference state (a.k.a. measured boot reference state) which is used by the policy engine 'elchecking' (part of the verifier) to evaluate the boot event log according to the rule/policy specified. The rule/policy to evaluate the boot event log against the measured boot reference state is specified in *verifier.conf*, with parameter named *measured_boot_policy_name*. The default value for it is *accept-all*, meaning "just don't try to match the contents, just replay the log and make sure the values of PCRs [0-9] and [11-14] match".

When the measured boot policy is provided by the operator while registering an agent node, the following actions will be taken.

1. PCRs [0-9] and [11-14] will be included in the quote sent by *keylime_agent*
2. The *keylime_agent* will also send the contents of`/sys/kernel/security/tpm0/binary_bios_measurements`
3. The *keylime_verifier* will replay the boot log from step 2, ensuring the correct values for PCRs collected in step 1. Again, this is very similar to what it is done with "IMA logs" and PCR 10.
4. The very same *keylime_verifier* will take the boot log, now deemed "attested" and evaluate it against the measured boot policy, causing the attestation to fail if it does not conform.

Note

The PCR replay in step 3 is restricted to the intersection of `MEASUREDBOOT_PCRS` (the PCRs Keylime collects) and the set returned by the active policy's `get_relevant_pcrs()` method. PCRs outside that set are excluded from the replay check. This allows policies to declare that certain PCRs (e.g. PCR 11, which `systemd-pcrphase` extends at runtime) are not part of their trust model, preventing spurious verification failures. Policies that return an empty set from `get_relevant_pcrs()` (such as the built-in `accept-all` and `reject-all`) apply no restriction, so all `MEASUREDBOOT_PCRS` are replayed.

2.7.3 How to use

The operator can provide the measured boot policy / measured boot reference state by using the option '–mb-policy' with the command *keylime_tenant*.

The simplest way to test this functionality is by providing an empty measured boot policy with the *accept-all* measured_boot_policy_name specified in *verifier.conf*, which will cause the *keylime_verifier* to skip the aforementioned steps 3 and 4 (no PCR replay and no policy evaluation).

An example follows:

```
echo "{}" > mb_policy.txt

keylime_tenant -c add -t <AGENT IP> -v <VERIFIER IP> -u <AGENT UUID> --mb-policy ./mb_
↪policy.txt
```

Note: please keep in mind that the IMA-specific options can be combined with the above options in the example, resulting in a configuration where a *keylime_agent* sent a quote with PCRs [0-15] and both logs (boot and IMA)

Evidently, to be fully used in a meaningful manner, keylime operators need to provide their own measured boot policies and custom python module supporting custom values of measured_boot_policy_name. The python module needs to be specified with *measured_boot_imports* in the *verifier.conf*.

The most convenient way to create a measured boot policy is starting from the contents of an UEFI boot log from a given node, and then tweak and customize it to make more generic. Keylime includes a tool (under *scripts* directory) - *create_mb_refstate* - which will consume a boot log and output a JSON file for measured boot policy. An example follows:

```
create_mb_refstate /sys/kernel/security/tpm0/binary_bios_measurements measured_boot_
↪reference_state.json

keylime_tenant -c add -t <AGENT IP> -v <VERIFIER IP> -u <AGENT UUID> --mb-policy ./
↪measured_boot_reference_state.json
```

This measured boot reference state can be (as in the example above) consumed "as is", or it can be tweaked to be made more generic (or even more strict, if the keylime operator chooses so).

Keylime facilitates a framework (a.k.a. elchecking policy engine) specified in *policies.py* under *keylime/mta/elchecking* directory, where some "trivial" policies such as *accept-all* and *reject-all* are pre-defined. The Domain-Specific Language (DSL) used by the framework are defined in *tests.py* and an illustrative use of it can be seen in the policy *example.py*, all under the same directory. This example policy was crafted to be meaningful (i.e., with a relevant number of parameters tests) and yet applicable to a large set of nodes. It consumes a measured boot policy such as the one generated by the aforementioned tool or the *example_reference_state.json*, located under the same directory.

Just to quickly exemplify what this policy does, it for instance tests if a node has *SecureBoot* enabled (*tests.FieldTest("Enabled", tests.StringEqual("Yes"))*) and if a node has a well-formed kernel command line boot parameters (e.g., *tests.FieldTest("String", tests.RegExp(r".*/grub.*"))*). The policy is well documented, and operators are encouraged to just read through the comments in order to understand how the tests are implemented.

While an operator can attempt to write its own policy from scratch, it is recommended that one just copies *example.py* into *mypolicy.py*, change it as required and then just points to this policy for *measured_boot_policy_name* in *verifier.conf* for its own use.

2.7.4 Named Measured Boot Policy

Keylime allows the operator to store measured boot policies with names and use the names to associate measured boot policies with various nodes. The policies are stored in a database maintained by the keylime verifier. The operator can also list, view, update and delete the policies stored.

Examples to add, update, show, delete and list named measuredboot policies:

```
keylime_tenant -c addmbpolicy     -v <VERIFIER_IP>  --mb-policy-name <policy1_name> --mb-
↪policy <policy1_file>

keylime_tenant -c updatembpolicy  -v <VERIFIER_IP>  --mb-policy-name <policy1_name> --mb-
↪policy <policy1_file2>

keylime_tenant -c showmbpolicy    -v <VERIFIER_IP>  --mb-policy-name <policy1_name>

keylime_tenant -c deletembpolicy  -v <VERIFIER_IP>  --mb-policy-name <policy1_name>

keylime_tenant -c listmbpolicy    -v <VERIFIER_IP>
```

Operator can provide the name of an stored measured boot policy to use the policy while registering a node as follows:

```
keylime_tenant -c add -t <AGENT IP> -v <VERIFIER IP> -u <AGENT UUID> --mb-policy-name
↪<policy1_name>
```

If the poliy is not already stored, the following command to register the node will also store the policy into the database:

```
keylime_tenant -c add -t <AGENT IP> -v <VERIFIER IP> -u <AGENT UUID> --mb-policy-name
↪<policy2_name> --mb-policy <policy2_file>
```

The following command to register the node will store the UUID of the node as the name of the policy into the database:

```
keylime_tenant -c add -t <AGENT IP> -v <VERIFIER IP> -u <AGENT UUID> --mb-policy
↪<policy3_file>
```

2.8 The `keylime-policy` tool

`keylime-policy` is a tool that is able to create both runtime and measured boot policies, as well as sign runtime policies. It emerged by consolidating this existing functionality that was available from a few scripts scattered across the Keylime source tree.

2.8.1 Creating runtime policies

A runtime policy in its most basic form is a set of "golden" cryptographic hashes of files' un-tampered state or of keys that may be loaded onto keyrings for IMA verification (*see more on Keylime Runtime Policies*).

To create runtime policies, we use the `keylime-policy create runtime` subcommand. This subcommand supports multiple options, which can be seen by issuing the command `keylime-policy create runtime --help`. The next table list some of the common options:

Table 4: Common options for creating runtime policies

Option	Description
`-o` or `--output`	This option specifies where the tool will output the data. If not specified, it will print the data to the standard output
`-m` or `--ima-meas`	This option specifies an IMA measurement list to use for obtaining data such as hashes and keyring. If not specified, it will attempt to use `/sys/kernel/security/ima/ascii_runtime_measurements` by default. If you want an empty list, you can use `/dev/null` instead
`-a` or `--alowlist`	Indicates a plain text allow list to read checksums from
`-e` or `--excludel`	This option specifies an exclude list file whose contents will be added to the policy. Files and directories matching the entries in this list – which also supports Python regular expressions with one regular expression per line – will be *ignored* by Keylime in the sense that it will not fail validation if those files change
`--ramdisk-`	This option specifies a path where the initial ramdisks (e.g. initrds or initramfs') are located, e.g.: `/boot`
`--rootfs`	This option specifies a path to a root file system, e.g.: `/`
`--local-rp`	This option specifies a local RPM repository directory; `keylime-policy` will create a runtime policy based on the files that are part of the RPMs of this repo
`--remote-r`	This option specifies a remote RPM repository; similar to `--local-rpm-repo`, `keylime-policy` will create a runtime policy based on the files that are part of the RPMs of this repo

Let us look at some examples next:

Converting a legacy allow list into a runtime policy

Legacy allow lists are files in which each line is of the form `<digest> <file>`. To convert such a file – *allowlist.txt*, in this example – into a runtime policy, you can do the following:

```
keylime-policy create runtime --allowlist allowlist.txt
```

Converting an exclude list into a runtime policy

An exclude list file contains a list of files or directories that are to be excluded from Keylime measurements. To convert an exclude list named **excludelist.txt** into a runtime policy, issue the following command:

```
keylime-policy create runtime --excludelist excludelist.txt
```

Creating a runtime policy with files from a rootfs

To create a runtime policy including files from a given root file system, specify the `--rootfs <directory>` argument, as in the next example:

```
keylime-policy create runtime --rootfs /mnt/rootfs
```

Creating a runtime policy with files from the initial ramdisks

To have a runtime policy including files from the initial ramdisks (initrd, initramfs), specify the direcory where they are located with `--ramdisk-dir`:

```
keylime-policy create runtime --ramdisk-dir /boot
```

Creating a runtime policy from RPM repositories

The `keylime-policy` tool is able to create runtime policies from RPM repositories, both local and remote:

Local repository

To create a policy from a local RPM repository, we can use the `--local-rpm-repo` switch:

```
keylime-policy create runtime --local-rpm-repo /tmp/local-rpm-repo
```

Note that, in this case, `/tmp/local-rpm-repo` should be a valid RPM repository, i.e., it should contain the `repodata` subdirectory with the relevant metadata files, such as `repomd.xml`.

For reference, createrepo_c is a tool capable of creating such repositories.

Remote repository

We can also create policies from remote RPM repositories, and in this case, the relevant `keylime-policy` switch is `--remote-rpm-repo`:

```
keylime-policy create runtime --remote-rpm-repo https://composes.stream.centos.org/
↪stream-10/production/latest-CentOS-Stream/compose/BaseOS/x86_64/os/
```

Similar to when we created a policy from a local repository, we need to make sure to give the address of a valid RPM repository to `keylime-policy`.

Also note that *this operation may take a long time*, especially in the case the `filelists-ext` metadata is not available from the repository.

Creating a runtime policy from multiple sources

The previous examples show how to generate a runtime policy based on a *single source of data*, e.g., from a local or remote RPM repository, from a legacy allowlist, from a root file system, etc. The `keylime-policy` tool is able to combine multiple sources while generating the runtime policy, as we can see in the next example:

```
keylime-policy create runtime --rootfs /mnt/rootfs --ramdisk-dir /boot --allowlist␣
↪allowlist.txt --excludelist excludelist.txt --local-rpm-repo /tmp/local-rpm-repo --
↪remote-rpm-repo https://composes.stream.centos.org/stream-10/production/latest-CentOS-
↪Stream/compose/BaseOS/x86_64/os/
```

Have in mind that, depending on the options used, the operation may take a long time.

2.8.2 Creating measured boot policies

`keylime-policy` supports consuming the UEFI event log file to generate a JSON file for a measured boot policy that can be later tweaked and customized to make it more generic, through the `keylime-policy create measured-boot` subcommand. The following table list the available options for it:

Table 5: Options for creating measured boot policies

Option	Description
`-o` or `--output`	This option specifies where the tool will output the data. If not specified, it will print the data to the standard output
`-e` or `--eventlog-file`	This option specifies the binary UEFI event log file, which is normally `/sys/kernel/security/tpm0/binary_bios_measurements`. This option is **required**
`-i` or `--without-secur`	This option indicates you want to create a measured boot reference policy without SecureBoot (only measured boot)

Create a measured boot policy

To create a measured boot policy with `keylime-tool` using the `/sys/kernel/security/tpm0/binary_bios_measurements` event log file, you can issue the following command:

```
keylime-policy create measured-boot -e /sys/kernel/security/tpm0/binary_bios_measurements
```

It may be required to add the `-i` switch to the above command, if the provided event log file has Secure Boot disabled; in this case, you should see a message like this, after running the previous command: *Provided eventlog has SecureBoot disabled, but -i flag was not set*.

2.8.3 Signing runtime policies

`keylime-policy` also supports signing Keylime runtime policies with DSSE (Dead Simple Signing Envelope) through the `keylime-policy sign runtime` subcommand. The available options for this subcommand are listed next:

Table 6: Options for signing runtime policies

Option	Description
`-o` or `--output`	This option specifies where the tool will output the data. If not specified, it will print the data to the standard output
`-r` or `--runtime-policy`	This option specifies the location of the runtime policy file. This option is **required**
`-k` or `--keyfile`	This option specifies the Elliptic-curve private key to sign the policy with
`-p` or `keypath`	This option specifies where the private key will be written to, if one is not specified via the `--keyfile` argument
`-b` or `--backend`	This option specifies the DSSE backend to use, which can be either `ecdsa` or `x509`. The default backend is *ecdsa*
`-c` or `--cert-outfile`	This option specifies the output file for the x509 certificate, when using the `x509` DSSE backend

When signing runtime policies, we need to select a DSSE backend, which can be either *ecdsa* or *x509*; if we don't explicitly select one of them, `keylime-policy` will use `ecdsa` as the default option.

The only strictly required option is the runtime policy to be signed, which can be provided via the `-r` switch. If you select the `x509` DSSE backend, you will also need to provide the output file for the x509 certificate with the `-c` option.

As for the private Elliptic-curve key to be used for the signing, you can either specify one with the `-k` switch, or `keylime-policy` will generate one for you. If it does generate one automatically, it will save this key with the name `keylime-ecdsa-key.pem`, in the current directory; if you want the generated key to have a different file name, you can specify the desired file name with the `-p` switch.

For the next examples, we will sign the `policy.json` runtime policy file and will **not** specify an output file with `-o`, so `keylime-policy` should output its result to *stdout*.

Signing a runtime policy with a provided private key

In this example, we have an EC private key `ec-p521-private.pem` and want to use it to sign our policy:

```
keylime-policy sign runtime -r policy.json -k ec-p521-private.pem
```

Signing a runtime policy without providing a private key

In this other example, we will not provida en EC private key, so `keylime-policy` will generate one for us and save it as `keylime-ecdsa-key.pem`:

```
keylime-policy sign runtime -r policy.json
```

You can verify that you now have a `keylime-ecdsa-key.pem` file with a content that looks like this:

```
-----BEGIN EC PRIVATE KEY-----
<key content here>
-----END EC PRIVATE KEY-----
```

Not providing a private key but specifying a custom name for the autogenerated key

Here we will not specify a private EC key for the signing, but we want the autogenerated key to have the name `autogen-ec-key.pem`:

```
keylime-policy sign runtime -r policy.json -p autogen-ec-key.pem
```

You can verify that you now have a `autogen-ec-key.pem` private key file.

Signing a policy using the x509 DSSE backend

In this example we will use the x509 DSSE backend. To do that, we need to specify the backend with *-b x509* and we also need to specify the output file for the x509 certificate with the *-c* switch:

```
keylime-policy sign runtime -r policy.json -k ec-p521-private.pem -b x509 -c x509.pem
```

You can verify that you now have a `x509.pem` file with the contents that look like this:

```
-----BEGIN CERTIFICATE-----
<certificate content here>
-----END CERTIFICATE-----
```

2.9 IDevID and IAK

Warning

This page is still under development and not complete. It will be so until this warning is removed.

2.9.1 Introduction

The IEEE 802.1AR Standard[1] defines a secure device identifier (DevID) as "a cryptographic identity that is bound to a device and used to assert the device's identity". The initial identity is defined as an IDevID/IAK ("I" for initial) and is first created in the factory where the server, switch, or other device is built before shipping. The IDevID credential is intended to be usable for the life of the product.

The IEEE 802.1AR Standard can be used together with TPM-based keys and certificates as described in[2]. This standard from the Trusted Computing Group (TCG) applies the IEEE Standard 802.1AR device identity module definition and formatting to keys protected by a TPM (conforming to version 2.0 of the TPM specification). The security of IDevIDs/IAKs are anchored in the TPM and its Endorsement Key (EK) as well as in the correct provisioning of the IDevID/IAK certificates by the device manufacturer.

The IDevID and IAK keys are generated by the TPM at the OEM factory and certified by the device manufacturer's certificate authority (CA). The CA enforces the TCG-defined provisioning protocol which ensures that the IDevID/IAK keys are resident in the TPM, similar to the standard process used to verify that an attestation key (AK) and endorsement key (EK) are colocated within a TPM. The IDevID certificates include information about the identity of the device, for example, the device's serial number. This can be used to bootstrap onboarding of devices for verification by Keylime. Using IDevID provides important security guarantees by ensuring that a device is identifiable, genuine and authorised. This allows for secure, automatic enrolment of devices even when they are geographically distributed (in remote or branch offices, for example).

2.9.2 How to use

To allow IDevID and IAK registration, the registrar needs to be, at minimum, loaded with the manufacturer certificates used to provision the IDevID and IAK and configured to accept any TPM identity for registration (the default behaviour). Optionally, the registrar can be configured to **require** an IDevID and IAK in order to register new devices. The agent needs to be configured to turn on IDevID and IAK regeneration, with the correct template selected, and also needs to know where it's IDevID and IAK certificates are stored.

[1] IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity, https://standards.ieee.org/standard/802_1AR-2018.html
[2] TPM 2.0 Keys for Device Identity and Attestation, https://trustedcomputinggroup.org/wp-content/uploads/TPM-2p0-Keys-for-Device-Identity-and-Attestation_v1_r12_pub10082021.pdf

Registrar

The certificates for the authority that signed the registering device's IAK and IDevID need to be added to the `TPM_CERT_STORE`, the default location being `/var/lib/keylime/tpm_cert_store`. The cert store can hold certificates fom multiple authorities if you have devices from different manufacturers. Keylime will look through the store until a match is detected.

With the manufacturer certificates added, the Keylime registrar will accept IDevIDs and IAKs received from agents during registration. In the default configuration, the registrar will not prevent an agent from registering with just an EK instead (and no IDevID and IAK). To disable this behaviour and make the use of IDevID and IAK a requirement for registering devices, the configuration option `tpm_identity` needs to be changed, either by modifying `registrar.conf`:

```
tpm_identity = iak_idevid
```

or alternatively by overriding the configuration option by setting the `KEYLIME_REGISTRAR_TPM_IDENTITY` environment variable.

Agent

For the agent to use its IDevID and IAK to register, the IAK and IDevID certificates need to be added to the `keylime_dir` and named `iak-cert.crt` and `idevid-cert.crt` respectively. Alternatively, the `iak_cert` and `idevid_cert` configuration options can be set to absolute paths or filenames relative to the `keylime_dir`.

As with the registrar, the configuration options can either be set in the config file, `agent.conf` for the agent, or overridden using environment variables.

Next, the functionality should be enabled by changing `enable_iak_idevid` to `true`. Finally, if using a standard template, the `iak_idevid_template` configuration option should be left with the default value `detect`. This will cause the Keylime agent to detect what template was used from the imported certificates. With all this complete your agent should be ready to register using its IDevID and IAK.

> **Note**
>
> The following information can be used to manually set either the template or the individual algorithms and should not be required for the majority of users. The detect function mentioned above should be used instead.

The template used by the certificates can also be manually specified. If the wrong template is chosen, the agent will fail to match the regenerated keys against the imported certificates and registration will fail.

The template should be set using the `iak_idevid_template` configuration option. As an alternative, the `iak_idevid_template` option can be set to `manual` and the `iak_idevid_asymmetric_alg` and `iak_idevid_name_alg` options can be used to manually set the algorithms, but this is not recommended, and **these options are ignored** if `iak_idevid_template` is not set to `manual`.

Template definitions can be found in section 7.3.4 of[Page 52, 2]. If you don't know what template your IAK and IDevID use, the following table can be used to match your algorithms to a template:

Template	asymmetricAlg	nameAlg
H-1	RSA 2048	SHA256
H-2	ECC NIST P256	SHA256
H-3	ECC NIST P384	SHA384
H-4	ECC NIST P521	SHA512
H-5	ECC SM2 P256	SM3_256

Qualifying Data in AK Certification (API v2.6+)

Starting with API version 2.6, the agent hashes the agent ID using SHA-256 before passing it as `qualifyingData` to `TPM2_Certify` when certifying the AK with the IAK. This ensures the qualifying data fits within the `TPM2B_DATA` size limit on TPMs that only support SHA-256 (where the limit is 34 bytes), which is smaller than a typical UUID string (36 bytes).

The registrar accepts both the hashed format (from agents using API v2.6+) and the raw agent ID format (from older agents) to maintain backward compatibility.

2.10 Secure Payloads

> **Warning**
>
> This page is still under development and not complete. It will be so until this warning is removed.

Secure payloads offer the ability to provision encrypted data to an enrolled node. This encrypted data can be used to deliver secrets needed by the node such as keys, passwords, certificate roots of trust, etc.

Secure payloads are for anything which requires strong confidentiality and integrity to bootstrap your system.

The payload itself is encrypted and sent via the Keylime Tenant CLI (or rest API) to the Keylime Agent. The Agent also sends part of the key needed to decrypt the payload, a key share, called the *u_key* or user key. Only when the Agent has passed its enrolment criteria (including any *tpm_policy* or IMA allowlist), will the other key share of the decryption key, called the *v_key* or verification key, be passed to the Agent by the Keylime Verifier to decrypt the payload.

> **Note**
>
> An alternative to secure payloads is to deliver the encrypted data to the node through some other mechanism like *cloud-init* or pre-embedded in a disk image. The Keylime protocol described above will still run to derive the decryption key for this data, but the data itself will never been seen or transported by Keylime. This guide does not discuss this method.

Keylime offers two modes for sending secure payloads: single file encryption and certificate package mode. In the following sections we describe each. If you're interested in using the more advanced certificate package mode, we recommend you also read the Single File Encryption section as it contains configuration options and other information that both modes share.

2.10.1 Single File Encryption

In this mode, a file you specify to the *keylime_tenant* application with the *-f* option will be encrypted by the Tenant using the bootstrap key and securely delivered to the Agent. Once the Keylime protocol with the Tenant and Verifier has completed, the Keylime Agent will decrypt this file and place it in */var/lib/keylime/secure/decrypted_payload* This is the default file name, but you can adjust the name of this file using the *dec_payload_file* option in *agent.conf*. You can also optionally specify a zip file as the file to be securely delivered. If the *extract_payload_zip* option in *agent.conf* is set (which it is by default), then Keylime will automatically extract the zip file to */var/lib/keylime/secure/unzipped*. Finally, Keylime can also execute a script contained in the zip file once it has been unzipped. You can think of this as a very simple form of cloud-init. By default this script is called *autorun.sh*. You can override this default with a different script name by adjusting the *payload_script* option in *agent.conf*. Note also that this script must be contained in the encrypted zip file, from which it will be extraced and then placed in */var/lib/keylime/secure/unzipped*.

Because the keys that Keylime uses to decrypt the data and the decrypted data itself are very sensitive, Keylime will only write those files to the memory-backed (and therefore non-persistent) */var/lib/keylime/secure* directory. This is a

bind-mounted tmpfs partition. As such, depending on how large your payload is, you may need to increase the size of this mounted partition by adjusting the *secure_size* option in *agent.conf*.

This simple mode of operation is suitable for many situations where the secrets and other bootstrapping information are basic. However, there are several features that Keylime supports like revocation and certificate management that do not work in this mode. For those, you'll need the next mode: Certificate Package Mode.

2.10.2 Certificate Package Mode

This mode of Keylime automates many common actions that tenants will want to take when provisioning their Agents. First, Keylime can create an X509 certificate authority (CA) using *keylime_ca -d listen* and then issue certificates and the corresponding private keys to each provisioned node. This CA lives on the same host where the tenant issues the *keylime_ca* command and can be used to bootstrap many other security solutions like mutual TLS or SSH. To use this mode, pass the *–cert* option and a directory where the CA is located as the parameter to this option. Keylime will then create a certificate for the node (with the common name set to the Agent's UUID) and then create a zip file containing the newly generated X509 certificates, trust roots, and private keys. It then uses the same process for single file encryption as described above to securely deliver all the keys to the Agent. Optionally, the user can specify with the *–include* option a directory of additional files to be put into the certification package zip and securely delivered to the Agent.

This mode of operation also natively supports certificate revocation. If the Keylime Verifier detects an Agent that no longer satisfies its integrity policy (e.g., it booted an unauthorized kernel or ran an unauthorized binary not on the IMA allowlist), it will create a signed revocation notification. These revocation notifications are signed by a special certificate/private key called the RevocationNotifier. Keylime will automatically create this certificate and pass it to the verifier when you add a new Agent to the verifier. Keylime will also include the public certificate for this key in the zip it sends to the Agent. This way Agents can validate the revocation notifications they receive from the verifier.

By default all Keylime Agents listen for these revocation notifications (see the *listen_notifications* option in *agent.conf*). Using the keys in the unzipped certificate package, Agents can check that the revocations are valid. Keylime Agents can also take actions in response to a valid revocation. You can configure these actions by putting additional files into the delivered zip file using *–include*.

Revocation actions are small Python scripts that will run on an Agent when a valid revocation is received. They should contain an *execute* function that takes one argument. This argument is a Python dictionary of metadata that can be used to tailor what the revocation action does. In the cert package mode, Keylime will specify the certificate serial number and common name (aka UUID) of the node that has failed its integrity check inside this metadata passed to the revocation action. For example, you can use this info to revoke the the offending X509 certificate.

One subtlety to revocation actions is that they are not intended for the Agent that has been revoked. If an Agent has failed its integrity check, then we really can't trust that it won't ignore the revocations and do arbitrarily malicious things. So, revocation actions are for other well-behaving Agents in the system to take action against the revoked Agent. For example, by revoking its certificate as described above or firewalling it from the network, etc.

There are some conventions to specifying revocation actions. As described above, their names must start with *local_action* to be executed. They also must be listed (without *.py* extensions) in a comma separated list in a file called *action_list* in the zip file. For example to run *local_action_a.py* and *local_action_b.py* the *action_list* file should contain *local_action_a,local_action_b*.

So far we've described all the details of this in fine detail, but much of this automation will happen by default.

2.10.3 Certificate Package Example

Let's put all of the above together with an example.

For the following example, we will provision some SSH keys onto the Agent.

1. Create a directory to host the files and *autorun.sh* script. For this example, we will use the directory *payload*
2. Create an *autorun.sh* script in the *payload* directory:

```
#!/bin/bash

# this will make it easier for us to find our own cert
ln -s `ls *-cert.crt | grep -v Revocation` mycert.crt

mkdir -p /root/.ssh/
cp payload_id_rsa* /root/.ssh/
chmod 600 /root/.ssh/payload_id_rsa*
```

3. Copy the files you wish to provision into the *payload* directory.

```
$ ls payload/
autorun.sh
payload_id_rsa.pub
payload_id_rsa
```

Send the files using the Keylime Tenant tool:

```
keylime_tenant -t <agent-ip> --cert myca --include payload
```

Recall that the *–cert* option tells Keylime to create a certificate authority at the default location */var/lib/keylime/ca* and give this machine an X509 identity with its UUID. Keylime will also create a revocation notifier certificate for this CA and make it available to the verifier. Finally, the *–include* option tells Keylime to securely deliver the files in the specified directory (*payload* in our case) along with the X509 certs to the targeted Agent machine.

If the enrolment was been successful, you will be able to see the contents of the *payload* directory in */var/lib/keylime/secure/unzipped* along with the certs and included files. You should also see the SSH keys we included made in */root/.ssh* directory from where the autorun.sh script was ran.

Now, let's extend this example with revocation. In this example, we're going to execute a simple revocation action on the node that was revoked.

It is also possible to configure scripts for execution should a node fail any given criteria (IMA measurements, for example).

To configure this, we will use our *payload* directory again.

First create a Python script with the preface of *local_action*

For example *local_action_rm_ssh.py*

Within this script create an *execute* function:

```
import os
import json
import keylime.ca_util as ca_util
import keylime.secure_mount as secure_mount

async def execute(event):
    if event['type'] != 'revocation':
        return

    json_meta = json.loads(event['meta_data'])
    serial = json_meta['cert_serial']

    # load up my own cert
    secdir = secure_mount.mount()
```

(continues on next page)

(continued from previous page)

```
    mycert = ca_util.load_cert_by_path(f'{secdir}/unzipped/mycert.crt')

    # is this revocation meant for me?
    if serial == mycert.serial_number:
        os.remove("/root/.ssh/payload_id_rsa")
        os.remove("/root/.ssh/payload_id_rsa.pub")
```

Next, in the *payload* directory create the *action_list* file containing *local_action_rm_ssh* (remember not to put the *.py* extension).

Warning

In the above example, the node that fails its integrity check is the same one that we're expecting to run the revocation action to delete the key. Since the node is potentially compromised, we really can't expect that it will actually do this and not just ignore the revocation. A more realistic scenario for SSH keys is to provision one node with the SSH key generated as above, then provision a second server and add *payload_id_rsa.pub* to *.ssh/authorized_keys* using an autorun script. At this point, you can SSH from the first server to the second one. Should the first machine fail its integrity, then an revocation action on the second server can remove the compromised first machine from its list of Secure machines in *.ssh/authorized_keys*

Many actions can be executed based on CA revocation. For more details and examples, please refer to the *Agent Revocation* page.

2.11 Agent Revocation

Warning

This page is still under development and not complete. It will be so until this warning is removed.

CHAPTER

THREE

DESIGN OF KEYLIME

3.1 Overview of Keylime

Keylime mainly consists of an agent, two server components (verifier and registrar) and a commandline tool the tenant.

3.1.1 Agent

The agent is a service that runs on the operating system that should be attested. It communicates with the TPM to enroll the AK and to generate quotes and collects the necessary data like the UEFI and IMA event logs to make state attestation possible.

The agent provides an interface to provision the device further once it was attested successfully for the first time using the secure payload mechanism. For more details see: *Secure Payloads*.

It is possible for the agent to listen to revocation events that are sent by the verifier if an agent attestation failed. This is useful for environments where attested systems directly communicate with each other and require that the other systems are trusted. In this case a revocation message might change local policies so that the compromised system cannot access any resources from other systems.

3.1.2 Registrar

The agent registers itself in the registrar. The registrar manages the agent enrollment process which includes getting an UUID for the agent, collecting the EK_{pub}, EK certificate and AK_{pub} from an agent and verifying that the AK belongs to the EK (using *MakeCredential* and *ActivateCredential*).

Once an agent has been registered in the registrar, it is ready to be enrolled for attestation. The tenant can use the EK certificate to verify the trustworthiness of the TPM.

Note

If EK or AK are mentioned outside of internal TPM signing operations, it usually references the EK_{pub} or AK_{pub} because it should not be possible extract the private keys out of the TPM.

Note

The Keylime agent currently generates a AK on every startup and sends the EK and EK certificate. This is done to keep then design simple by not requiring a third party to verify the EK. The EK (and EK certificate) is required to verify the authenticity of the AK once and Keylime does not require a new AK but currently registration only with an AK is not enabled because the agent does not implement persisting the AK.

3.1.3 Verifier

The verifier implements the actual attestation of an agent and sends revocation messages if an agent leaves the trusted state.

In the default **pull model**, once an agent is registered for attestation (using the tenant or the API directly) the verifier continuously pulls the required attestation data from the agent. This can include: a quote over the PCRs, the PCR values, NK public key, IMA log and UEFI event log. After that the quote is validated additional validation of the data can be configured.

Keylime also supports a **push model** where the agent initiates connections to the verifier and proactively submits attestation evidence. This is useful for environments where the verifier cannot directly reach the agent (e.g. behind firewalls or NAT). See *Push-Model Attestation* for details.

Static PCR values

The *tpm_policy* allows for simple checking of PCR values against a known good allowlist. In most cases this is only useful when the boot chain does not change often, there is a way to retrieve the values beforehand and the UEFI event log is unavailable. More information can be found in *User Selected PCR Monitoring*.

Measured Boot using the UEFI Event Log

On larger deployments it is not feasible to collect golden values for the PCR values used for measured boot. To make attestation still possible Keylime includes a policy engine for validating the UEFI event log. This is the preferred method because static PCR values are fragile because they change if something in the boot chain is updated (firmware, Shim, GRUB, Linux kernel, initrd, ...). More information can be found in *Use Measured Boot*.

IMA validation

Keylime allows to verify files measured by IMA against either a provided allowlist or a signature. This makes it for example easy to attest all files that were executed by root. More information can be found in *Runtime Integrity Monitoring*.

3.1.4 Tenant

The tenant is a commandline management tool shipped by Keylime to manage agents. This includes adding or removing the agent from attestation, validation of the EK certificate against a cert store and getting the status of an agent. It also provides the necessary tools for the payload mechanism and revocation actions.

For all the features of the tenant see the output of `keylime_tenant --help`.

3.2 Push-Model Attestation

> **Warning**
>
> Push-model attestation is currently experimental. The feature is functional but the API and configuration options may change in future releases. Please report issues at https://github.com/keylime/keylime/issues/?q=label:push-mode

3.2.1 Introduction

Traditional Keylime attestation uses a **pull model** where the verifier continuously polls agents for attestation data. The agent acts as a server and the verifier initiates connections to it. This model requires that the verifier can reach the agent over the network.

The **push model** reverses this communication direction: the agent initiates connections to the verifier and proactively sends attestation data. The verifier never connects to the agent. This makes push-model attestation suitable for environments where the verifier cannot directly reach the agent, such as:

- **Edge and IoT devices** behind firewalls or NAT
- **Hybrid cloud environments** with restricted network policies
- **Air-gapped networks** where inbound connections to agents are not permitted
- **Dynamic environments** where agent IP addresses change frequently

In push mode, the agent is a separate binary (`keylime-push-model-agent`) that implements the push attestation protocol using API version 3.0.

3.2.2 Architectural Overview

In pull-model attestation, the verifier runs a polling loop that periodically contacts each registered agent to request a TPM quote and associated evidence. The agent exposes an HTTPS server that responds to these requests.

In push-model attestation, this relationship is inverted:

- The **agent initiates** all connections to the verifier
- The agent does **not expose any HTTP endpoints** (no listening ports)
- The verifier accepts incoming attestation data from agents
- Verification is performed **asynchronously** after evidence is received
- An **event-driven timeout** system replaces the polling loop for monitoring agent liveness

The registrar interaction is unchanged: in both models, the agent registers itself with the registrar during startup.

Fig. 1: **Figure 1:** Push-Model Architecture

3.2.3 The Two-Phase Attestation Protocol

Push-model attestation uses a two-phase protocol for each attestation cycle.

Phase 1: Capabilities Negotiation

The agent begins an attestation cycle by sending its capabilities to the verifier. This tells the verifier what types of evidence the agent can produce and what cryptographic algorithms it supports.

1. The agent sends a `POST /v3/agents/{agent_id}/attestations` request to the verifier containing its supported evidence types (TPM quote parameters, IMA log capabilities, UEFI log capabilities) and the public attestation key (AK).
2. The verifier creates an attestation resource, selects cryptographic parameters (signature scheme, hash algorithm, PCRs to quote), generates a random challenge nonce, and returns a `201 Created` response with:
    - The challenge nonce for TPM quote generation
    - The chosen cryptographic parameters
    - The evidence types requested
    - A deadline (`challenges_expire_at`) by which evidence must be submitted

Phase 2: Evidence Submission

The agent collects the requested evidence and submits it to the verifier.

1. The agent generates a TPM quote using the challenge nonce from Phase 1, collects IMA and/or UEFI event logs as requested, and sends a `PATCH /v3/agents/{agent_id}/attestations/latest` request with the evidence.
2. The verifier returns a `202 Accepted` response immediately. The evidence is then verified asynchronously in a background worker process.
3. If verification succeeds, the attestation is marked as `pass`. If it fails, the attestation is marked as `fail` with a failure reason (`broken_evidence_chain` or `policy_violation`).
4. The response includes a `seconds_to_next_attestation` value in the `meta` field, indicating when the agent should start its next attestation cycle.

After a configurable interval, the agent begins a new cycle from Phase 1.

Agent State Machine

The push-model agent operates as a state machine with the following states:

Fig. 2: **Figure 2:** Push-Model Agent State Machine

- **Unregistered**: Initial state. The agent registers with the registrar.
- **Registered**: Registration succeeded. The agent begins negotiation with the verifier.
- **Negotiating**: The agent sends capabilities to the verifier (Phase 1) and waits for the challenge response.
- **Attesting**: The agent generates and sends evidence to the verifier (Phase 2). On success, the agent waits for the configured interval and transitions back to Negotiating.
- **RegistrationFailed**: Registration with the registrar failed. The agent waits and retries.
- **AttestationFailed**: An attestation attempt failed (network error or verifier rejection). The agent waits and retries from Negotiating.

The agent uses exponential backoff when retrying failed operations.

3.2.4 Authentication

Push-model attestation uses **Proof of Possession (PoP)** authentication instead of the mTLS client certificates used in pull mode. This is necessary because the agent acts as a client (not a server) and does not have certificates signed by the verifier's trusted CA.

The PoP authentication flow:

1. The agent creates a session by sending `POST /v3/sessions` with its agent ID and supported authentication methods.
2. The verifier responds with a challenge nonce.
3. The agent proves possession of its AK by signing the challenge using the TPM (`TPM2_Certify`) and sends the result via `PATCH /v3/sessions/{session_id}`.
4. If the signature is valid, the verifier issues a bearer token.
5. The agent includes this token in the `Authorization` header of all subsequent requests.
6. Tokens have a configurable expiration time and can be refreshed.

The TLS connection uses **server verification only**: the agent verifies the verifier's server certificate but does not present a client certificate. The agent needs the verifier's CA certificate for this verification.

For full details on the authorization framework, including the separation between agent and admin authentication, see *Authentication*.

3.2.5 Timeout Monitoring

In pull mode, the verifier detects unresponsive agents through its polling loop. In push mode, an event-driven timeout system serves this purpose.

The verifier monitors push-mode agents as follows:

1. When the verifier receives an attestation from an agent, it schedules a timeout for that agent. The timeout duration is `quote_interval * 5` seconds (where `quote_interval` is the verifier's configured quote interval).
2. If the agent does not submit a new attestation before the timeout fires, the verifier sets the agent's `accept_attestations` flag to `False`.
3. Once `accept_attestations` is `False`, the verifier rejects new attestation requests from that agent with a `403 Forbidden` response.
4. The agent can recover by re-registering or by administrator intervention (reactivation).

3.2.6 Comparison with Pull Model

Aspect	Pull Model	Push Model
Connection direction	Verifier connects to agent	Agent connects to verifier
Agent binary	`keylime_agent`	`keylime_push_model_agent`
Agent network requirements	Must expose HTTP port (default 9002)	No listening ports required
Firewall requirements	Inbound to agent from verifier	Outbound from agent to verifier
Authentication method	mTLS (agent as server)	PoP bearer tokens (agent as client)
API version	v2.x	v3.0
Verification trigger	Verifier polls on `quote_interval`	Agent pushes on `attestation_interval_seconds`
Liveness detection	Polling loop state machine	Event-driven timeout (`quote_interval * 5`)
Verifier configuration	`mode = pull` (default)	`mode = push`
Suitable for	Controlled networks, data centers	Edge, IoT, NAT, firewalled environments
Maturity	Stable	Experimental

For deployment and configuration instructions, see *Push-Model Attestation*. For the v3.0 API reference, see *RESTful API for Keylime (v3.0)*.

3.3 Attestation Security

Keylime's core purpose is to verify the attested state of a system. The verification outcome (whether the attestation is verified or not) may be used in various ways by the end user by integrating Keylime into their wider infrastructure, for instance:

- to produce alerts if an unauthorised change occurs somewhere in a user's fleet of machines (e.g., boot order is so configured that a server boots from an external drive);
- to authenticate a workload based on the state of the workload and the node on which it is running, in service of zero-trust principles; or

- to release keys from a key broker to unlock an encrypted data store once the data store system has been verified.

As a result, a user must have faith that the verification outcome reported by Keylime is correct for the specific system in question. It is crucial therefore to understand the security architecture and characteristics of Keylime and attestation technologies broadly, especially as the security of an attestation service (whether Keylime or another verification engine) depends heavily on the particular deployment.

Note

At the time of this writing, Keylime only supports TPM-based attestation of boot state as recorded in UEFI logs, of file system integrity as recorded in Linux IMA logs, and of a TPM's platform configuration registers (PCRs) directly. As Keylime may support other forms of attestation in the future, e.g., attestations produced by various trusted execution environments (TEEs), this page attempts to be agnostic as to the attestation technology being used in so far as is possible but does use UEFI and IMA attestation as concrete examples by which to illustrate the general concepts.

3.3.1 Attestation Terminology

At a high level, an attested node consists of a number of *attesting environments* which each consist of a stack of software and hardware. These collect *claims* about the state of the attested node (claims are also called *measurements*) and produce *evidence* that these claims may be believable (a collection of evidence, including claims, is what is usually referred to as an *attestation*). The evidence is authenticated cryptographically such that it can be verified to have been produced, at least in part, by a specific component.

An attesting environment can be further split into a *measuring environment* and a *certifying environment*[1]. The measuring environment collects claims/measurements and the certifying environment acts as a witness, certifying that it has seen the claims/measurements. For example, during boot with UEFI, the firmware produces a log of events which are measured into the TPM. Later, the TPM may be asked to certify the sequence of events which it received (this certification is also known as a *quote*). In this situation, the measuring environment consists of UEFI and the hardware platform it is running on, and the certifying environment is the TPM.

In some cases, the measuring environment and certifying environment could be the same. When attesting certain trusted execution environments (TEEs), for example, the TEE hardware may perform both the measuring and certifying tasks.

It is important to note that attesting environments are not required to be entirely separate from one another and, in fact, often share components. This is illustrated by the diagram below showing UEFI and IMA attestation being performed on the same node:

Fig. 3: **Figure 1:** Overlapping Attesting Environments

The red shaded area shows the attesting environment which attests the boot state, whereas the blue shaded area shows the attesting environment used to attest the integrity of files using IMA. The overlapping purple area contains the components common to both. In both attesting environments, the certifying environment is the TPM.

3.3.2 Trust Relationships

The trust that a user chooses to place in the verification results produced by a deployment of Keylime should derive from their trust in specific system components (*trust anchors*) and the cryptographic means by which this trust is apportioned to other components and data. This is dependent on the secure design of the hardware, firmware and software which produces the attestation evidence (collectively, the *attesting environment*), the secure design of Keylime and any extensions or integrations, and the configuration of the system by the user.

[1] *Attesting environments*, *claims*, and *evidence* are the terms preferred by the IETF's Remote Attestation Procedures (RATS) working group in their architecture specification, RFC 9334. Although they do not explicitly divide attesting environments into a *measuring environment* and *certifying environment* as we do here, separating claims collection and certification of claims into separate components is contemplated in section 3.1.

As such, contributors to the Keylime project and users of Keylime alike need to consider the resulting *chain of trust* when these units are composed together. To demonstrate this concept, a possible deployment is given in the below figure:

Fig. 4: **Figure 2:** Example Keylime Deployment Performing UEFI and IMA Verification

In this example, the user has installed the Keylime agent on a physical node which identifies itself to an instance of the Keylime registrar and delivers evidence to a separate Keylime verifier instance. As in the diagram from the previous section, the node is able to attest the contents of its UEFI boot log and the integrity of specific files using Linux IMA. The user has configured the verifier with a certain *verification policy*[2] which it will use to evaluate the evidence received in each periodic attestation.

Trust in the Measuring Environment

When the attested node boots, the UEFI firmware and the bootloader each have their turn to execute in the boot sequence. They both write entries to the boot log and, for each log entry, update registers in the TPM with a hash of that entry. Nothing in the operation of the TPM ensures that the log entries **accurately** describe the events which took place at boot time,[3]_ so the firmware and bootloader must be trusted to be honest when writing to the log.

Like any software component, the firmware and bootloader are subject to modification by legitimate users (e.g., when performing an update) and malicious parties. But because the node in question has a Baseboard Management Controller (BMC) which acts as an additional *hardware root of trust* together with the TPM, the user has a strong assurance that only the correct, authenticated firmware is loaded into memory. Additionally, assuming Secure Boot is enabled, UEFI will only launch the bootloader if it is correctly signed by an authorised OS vendor.

Warning

In physical systems, the low-level software, such as the processor microcode and UEFI firmware, is loaded into memory by a trusted component like a BMC. In contrast, **virtualised systems** running on multi-tenant hypervisors and hardware, such as a virtual machine (VM) running in a public cloud (like AWS or Azure), do not have typically have an equivalent hardware root of trust. Your *trusted computing base* is effectively the cloud service provider's (CSP's) entire infrastructure.

See *Virtual TPMs and the Root of Trust* below for considerations when operating in such an environment.

Note

The BMC may also perform authentication of certain hardware components, but this depends on the platform. We are therefore treating the entire hardware platform as a trust anchor in this example. As hardware manufacturers adopt SPDM, hardware authentication is expected to become more commonplace.

The environment which produces the boot log is therefore trusted transitively: log entries are generated by an authorised firmware and bootloader. The bootloader is trusted because the firmware which authenticates it is trusted. And the firmware is trusted because the BMC which loads it into memory is trusted.

The environment which produces file integrity logs is trusted in similar fashion. As IMA generates the logs, and IMA is part of the Linux kernel, it is authenticated by the bootloader before executing.

In both cases, trust in every component of the measuring environment can be established by tracing it to one or more trust anchors. Therefore, the measuring environment as a whole can be trusted.[4]_

[2] It is common for a verification policy to perform verification of evidence against a separate set of *reference values* or *reference measurements*. For the purposes of this page, we consider that any reference values are part of the verification policy itself, as the distinction should not impact security analysis.

Trust in the Certifying Environment

Even if we trust the measuring environment which produces the claims contained in UEFI and IMA logs, this alone is not enough to trust the logs themselves. This is because other components of the attested node outside the measuring environment could tamper with the logs, changing their contents before they arrive at the verifier.

As such, the measuring environment must be combined with a trusted certifying environment. In our example (and in the usual case), such an environment is made available by the TPM.

The TPM receives each log entry as it is written and uses this to calculate a digest (or hash) with a cryptographic one-way hash function. The input to the hash function is the log entry concatenated to any previous digest held in the TPM register assigned to that log. The output of the function is then used to overwrite the register's present contents. This is known as an *extend* operation.

As this *rolling hash* is deterministically calculated from each entry in sequence, it represents the entire history of the log from the time the system boots. If a previous entry were to change after originally being written to the log, the log would no longer match the TPM hash. The TPM does not allow the register containing the hash to be modified in any way other than by extending the rolling hash or by rebooting the system (which clears the TPM's registers).

> **Warning**
>
> The TPM does not authenticate the component from which it receives log entries. This means that any program on the system can append to the end of the log and the log would still match the rolling hash. This is of limited use to an attacker which wishes modify the boot log, as boot events appearing after the boot sequence has been logged as having finished would not make sense.
>
> In the case of IMA, this characteristic of the TPM means that an attacker could append its own events to the IMA log and it would not be possible to differentiate these from legitimate events produced by IMA itself. This should not matter for conventional uses where IMA is used to detect unauthorised file modifications. The operation of IMA guarantees that its measurement of a file at time of access is logged and used to extend the appropriate TPM register. The IMA log, however, cannot be used as a guarantee of the current state of a file.

Finally, when the logs are collected by the Keylime agent and sent to the verifier, the agent asks the TPM to certify one or more of its rolling hashes. The TPM signs the hashes with a private key known only by the TPM. The resulting *quote* is received by the verifier which can verify its authenticity against its knowledge of the TPM's public key. Then, the logs can be verified against the rolling hash.

In other words, the verifier can trust the received logs to contain the entries produced by one or more trusted measuring environments because the logs match against the rolling hash. The hash can be trusted because it can be confirmed from the signature to have been produced by a TPM which we trust to operate according to the TPM spec.

Warning

Physical TPMs are certified against the TPM specification and can be verified against a manufacturer certificate to establish authenticity. There is no certification programme for virtual TPMs (vTPM). Because of this, in a cloud environment, you typically have no assurance beyond the cloud service provider's (CSP's) word that the vTPM's behaviour is to spec.

See *Virtual TPMs and the Root of Trust* below for considerations when operating in such an environment.

Chaining Trust Across Attesting Environments

In the previous example, a chain of trust is formed in large part by virtue of Secure Boot, a UEFI feature which authenticates each component in the boot sequence. However, Secure Boot is imperfect. A motivated attacker can replace the bootloader of a system with an old, vulnerable version which is accepted by the UEFI firmware as legitimate because it has been signed by an authorised OS vendor. This type of attack has succeeded in the past and has proved difficult to remediate,[5]_ as signing keys cannot be easily revoked without breaking many systems, preventing them from booting.

Instead of relying on Secure Boot, it is better to authenticate the boot chain as part of your verification policy. This is possible because UEFI outputs the hash of the bootloader to the boot log when it loads it into memory. Your policy can check this against a set of *reference values* of legitimate, up-to-date bootloaders.

Note

The behaviour of UEFI when it loads the bootloader, including what logs are produced, is described in section 7 of the TCG PC Client Platform Firmware Profile Specification. You should verify the hash of every EFI application launched as part of the boot process to establish a complete chain of trust.

The bootloader, in similar fashion, measures the kernel to the boot log before passing control to the OS. As a result, it is possible to authenticate the kernel in your verification policy also.

From a security analysis perspective, it is important to grasp the following concept: the trust placed in an attesting environment may be **conditional** on a verification outcome of an attestation produced by another attesting environment. The attesting environment which produces a node's IMA log, for instance, may be trusted only if the attesting environment which produces the UEFI log containing the hash of the kernel is trusted.

3.3.3 Virtual TPMs and the Root of Trust

All our examples up to now have used a hardware certifying environment in the form of a TPM which is part of the hardware platform of the attested node. However, Keylime can perform TPM-based attestation using any TPM-like device, physical or virtual, which implements the TPM 2.0 standard. In the ideal scenario, whatever TPM is used should have a chain of trust which is rooted in hardware.[6]_

That said, there are situations in which only a TPM implemented in, and secured by, software is available. Such a virtual TPM (vTPM) needs to be located on a trusted system. For example, when attesting a VM running in a cloud environment, you may choose to trust a vTPM provided by your cloud service provider (CSP) and running as part of the hypervisor.

Note

Keylime was originally developed to attest VMs using the deep quotes provided by vTPM support in Xen, for which the root of trust was a hardware TPM. However, support beyond TPM 1.2 was never implemented. The vTPMs provided by most hypervisors today no longer have a chain of trust rooted in hardware.

In such case, you should still use whatever mechanism is provided by your CSP to authenticate the vTPM. For example, Google Cloud allows you to obtain the vTPM's endorsement key for a given VM from an HTTPS endpoint. Failure to do so would allow an attacker to swap quotes produced by your trusted vTPM with quotes produced by its own malicious vTPM.

Anchoring Trust in a Trusted Execution Environment (TEE)

> **Note**
>
> Keylime does not currently have specific support for TEEs. This section may be used as a guide for implementers of such functionality (whether within Keylime itself or as a third-party extension) and serves as additional discussion on trust anchors broadly.

In a confidential computing scenario, a vTPM may be running in a trusted execution environment (TEE) which has been attested and verified to be secure by virtue of the memory-protection guarantees granted by the CPU. In such case, the CPU would act as a hardware trust anchor. Trust in the software certifying environment provided by the vTPM would be established transitively in the manner described in the previous section: the CPU attests the state of the TEE and this attestation establishes trust in the vTPM.

Establishing trust in the measuring environment works similarly: for example, the instance of UEFI firmware running in a given TEE can be authenticated by checking an attestation produced by the CPU. Additionally, the attestation which authenticates a given measuring environment should be the same attestation which authenticates the certifying environment (the vTPM). This serves to bind the two environments and establish a unified attesting environment (in physical systems, this binding is established implicitly by the fact that hardware which loads the UEFI firmware, e.g., is physically connected to the TPM).

Additionally, because TEEs can be created on the fly, you may not be able to assume that only a single instance of a given vTPM exists, as you can with a physical TPM. Care must therefore be taken to bind each quote produced by a vTPM to its specific VM instance. Otherwise, a malicious hypervisor could clone a VM (including its vTPM) and interleave messages so that the verifier thinks quotes produced by one vTPM are produced by another.

> **Warning**
>
> While confidential computing technologies are still in their infancy, most CSPs have chosen to sacrifice security for simplicity. The result in many cases is that you are still required to trust the CSP's hypervisor or some other proprietary CSP component, beyond just the CPU. For instance, the security of Azure's OpenHCL paravisor (including its vTPM) depends on a Hyper-V feature called "Virtual Secure Mode".

3.3.4 Platform Identity

Fundamentally, the job of a verifier is to accept evidence from nodes on a network and apply the appropriate verification policy to produce a verification outcome for each node. As different nodes may have different policies, it is important that the verifier is able to reliably identify and authenticate the underlying platform. Otherwise, an attacker could cause the wrong verification policy to be applied to a node.

Depending on the specific deployment, this could have the affect of causing verification of a node to succeed when it should have failed. Or, alternatively, causing verification to fail when it should have succeeded, giving rise to a denial of service (DoS) scenario.

Whatever key is used to sign an attestation therefore needs to be bound to the individual node in question. Further, that binding needs to be performed by a trusted entity. The binding may be transitive so that the attestation signing key is bound to another key which itself is bound to the attested node.

In Keylime, attestations can be bound to the attested node in a number of different ways:

Binding to a TPM Endorsement Key

Attestations produced by a TPM are authenticated by an attestation key (AK) which is typically cryptographically bound to the TPM's endorsement key (EK). The authenticity of the EK can be determined by an EK certificate which is usually loaded into the TPM's non-volatile memory by the TPM manufacturer.

While the EK is required to be unique to the specific TPM, it is not linked to any identifying information about the device in which the TPM is installed (the EK certificate does not contain any such information). This is an intentional design choice by the Trusted Computing Group (TCG) which produces the TPM standard.

> **Note**
>
> The TPM 2.0 spec says that **a binding must be established** between the TPM and the platform before you can trust a TPM quote, but does not provide a built-in way to do so. This is covered in part 1, section 9.4.3.3 of the specification.

When the Keylime agent first starts on the node to be attested, by default, it registers its EK, EK certificate and an attestation key (AK), bound to the EK, with the registrar using an agent ID randomly generated by the agent or provided by the user. The user can then use the Keylime tenant or REST API to retrieve these from the registrar, using the agent ID, and enrol the AK with the verifier. Neither the registrar, the tenant, nor the verifier attempt to verify the identity of the node by default.

> **Note**
>
> Notice in the previous diagram (Figure 2) that there is no chain of trust from a trust anchor to the Keylime agent. This means that the Keylime agent cannot be trusted to report the correct agent ID to the registrar.

If the user wishes to rely solely on the EK as identity for the attested node, they are expected to manually verify the EK out of band themselves **before** enrolling the node for verification. This can be done using tpm2-tools.

Other Identity Binding Options

There are other ways of binding attestations produced by Keylime to a specific node. These may be more involved but are less fragile and therefore may be better from an operations perspective:

- If the node in question has been issued a Device Identity (DevID) by its manufacturer, the AK can be bound directly to this identity which itself is bound to the EK by the device manufacturer. The user simply needs to provide its IDevID and IAK certificates, which contain the serial number of the device or other user-facing identifying information, and the manufacturer's CA certificates. See the *IDevID and IAK* page.
- The user may construct an inventory database mapping node identifiers chosen by the user (e.g., hostnames) to an AK or EK. This database can be consulted before a node is added to the verifier by mechanisms available in Keylime.[6]_
- The user may set up their own automatic process to verify possession of an AK or EK as well as the identity of the node through a protocol like ACME or other procedure and add the node to the verifier only if these checks pass. This would need to be implemented as an unofficial extension to Keylime.

3.3.5 Threat Model

In the design of a secure system, it is prudent to define a threat model in terms of the capabilities of an idealised attacker. This has a number of advantages, not limited to the following:

- users are clear on the security properties they can expect from the system;
- developers have agreement on which attacks are in scope and which are out of scope; and

- the protocols utilised naturally lend themselves to analysis by outside parties.

In lieu of a full formal model, we give a plain English description, translatable to formal definitions, in the subsections below.

Security Goals

We give the main security property for Keylime by stating what a successful adversary must achieve:

> **A valid attack against Keylime is one in which an adversary can cause a mismatch between a verification outcome reported by a verifier and the correct, expected verification outcome for the verified node.**

This includes attacks in which:

- verification of a node is reported as having passed when the policy for the node should have resulted in a verification failure; or
- verification of a node is reported as having failed when the policy for the node should have resulted in a successful verification.

The latter is important to consider because, depending on how Keylime is used (e.g., if Keylime results are consumed by another system or used for authentication of non-person entities), this could be exploited to trigger cascading failures throughout the network.

The Capabilities of the Adversary

For our adversary, we consider a typical network-based (Dolev-Yao) attacker[7] which exercises full control over the network and can intercept, block and modify all messages but cannot break cryptographic primitives (all cryptography is assumed perfect). Because we need to consider attacks in which the adversary is resident on a node to be verified, we extend the "network" to include channels between the agent and any attesting environment (for TPM-based attestation, this includes communication between the TPM and the agent).

The adversary cannot corrupt (i.e., take control of, or impersonate) the verifier, registrar, tenant or any attesting environment, but has full control over the rest of the system, including the nodes' filesystem and memory.

Exclusions

Attacks which exploit poorly-defined verification policies or deficiencies in the information which can be obtained from a node's attesting environments (including IMA and UEFI logs) are necessarily out of scope. Additionally, we exclude attacks which are made possible by incorrect configuration by the user. Attacks which rely on modification of a system root of trust (such as the BMC or TPM) are also excluded.

Footnotes:

[7] Currently, this may be achieved by setting the `ek_check_script` configuration option in `/etc/keylime/tenant.conf`. In the future, it will be possible to use an external service queried by the registrar instead (see `PR 1670`_).

CHAPTER

FOUR

ADDITIONAL READING

This section contains a list of additional blog posts, research and talks about or related to Keylime.

4.1 Blogs entries

- Daniele Buono, Marcio A. Silva, Maurizio Drocco, Gheorghe Almasi, and James Bottomley. Extending server integrity with Durable Attestation. URL: https://research.ibm.com/blog/durable-attestation-cloud-security (visited on 2023-12-02).
- Kylie Foy. Keylime security software is deployed to IBM cloud. URL: https://news.mit.edu/2021/keylime-security-software-deployed-ibm-cloud-0727 (visited on 2023-12-02).
- Vikram Narayanan, Claudio Carvalho, Angelo Ruocco, Gheorghe Almási, James Bottomley, Mengmei Ye, Tobin Feldman-Fitzthum, Daniele Buono, Hubertus Franke, and Anton Burtsev. Remote attestation of SEV-SNP confidential VMs using e-vTPMs. URL: http://arxiv.org/abs/2303.16463 (visited on 2023-12-02), arXiv:2303.16463, doi:10.48550/arXiv.2303.16463.
- Michael Peters and Gheorghe Almasi. IBM implements remote attestation on Linux with a hardware root-of-trust using Keylime. URL: https://www.cncf.io/blog/2021/07/06/ibm-implements-remote-attestation-on-linux-with-a-hardware-root-of-trust-using-keylime/ (visited on 2023-12-02).
- Michael Peters, Marcio A. Silva, George Almási, James Bottomley, and Lily Sturmann. Keylime's durable attestation makes security auditable. URL: https://next.redhat.com/2023/04/25/keylimes-durable-attestation-makes-security-auditable/ (visited on 2023-12-02).
- Alberto Planas. MicroOS Remote Attestation with TPM and Keylime. URL: https://microos.opensuse.org/blog/2021-11-08-MicroOS-Keylime-TPM/ (visited on 2023-12-02).
- Patrick Uiterwijk. TPM2 Key Trust: where did Keylime go wrong. URL: https://puiterwijk.org/posts/tpm2-attestation-keylime-vulnerability/ (visited on 2023-12-02).
- Kimberly Underwood. Keylime Provides Root-of-Trust at Scale. URL: https://www.afcea.org/signal-media/keylime-provides-root-trust-scale (visited on 2023-12-02).

4.2 Academic Research

4.2.1 Original papers

- Amin Mosayyebzadeh, Gerardo Ravago, Apoorve Mohan, Ali Raza, Sahil Tikale, Nabil Schear, Trammell Hudson, Jason Hennessey, Naved Ansari, Kyle Hogan, Charles Munson, Larry Rudolph, Gene Cooperman, Peter Desnoyers, and Orran Krieger. A Secure Cloud with Minimal Provider Trust. *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, July 2018.

- Nabil Schear, Patrick T. Cable, Thomas M. Moyer, Bryan Richard, and Robert Rudd. Bootstrapping and maintaining trust in the cloud. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 65–77. ACM, December 2016. URL: https://dl.acm.org/doi/10.1145/2991079.2991104 (visited on 2023-12-02), doi:10.1145/2991079.2991104.

4.2.2 Research using Keylime

- Diana Gratiela Berbecaru and Silvia Sisinni. Counteracting software integrity attacks on IoT devices with remote attestation: a prototype. In *2022 26th International Conference on System Theory, Control and Computing (ICSTCC)*, 380–385. October 2022. URL: https://ieeexplore.ieee.org/document/9931765 (visited on 2023-12-02), doi:10.1109/ICSTCC55426.2022.9931765.
- Antonio Lioy, Dr Ignazio Pedone, and Dr Silvia Sisinni. TPM 2.0-based Attestation of a Kubernetes Cluster. *Politecnico di Torino*, 2023.
- Vikram Narayanan, Claudio Carvalho, Angelo Ruocco, Gheorghe Almási, James Bottomley, Mengmei Ye, Tobin Feldman-Fitzthum, Daniele Buono, Hubertus Franke, and Anton Burtsev. Remote attestation of SEV-SNP confidential VMs using e-vTPMs. URL: http://arxiv.org/abs/2303.16463 (visited on 2023-12-02), arXiv:2303.16463, doi:10.48550/arXiv.2303.16463.
- Silvia Sisinni, Davide Margaria, Ignazio Pedone, Antonio Lioy, and Andrea Vesco. Integrity Verification of Distributed Nodes in Critical Infrastructures. *Sensors 2022*, 22(18):6950, September 2022. URL: https://www.mdpi.com/1424-8220/22/18/6950 (visited on 2023-12-02), doi:10.3390/s22186950.

4.3 Talks and Live Demos

- Luke Hinds. Keylime - An Open Source TPM Project for Remote Trust of IoT. URL: https://www.youtube.com/watch?v=jtbWnod5hoY (visited on 2023-12-02).
- Luke Hinds. Keylime - An Open Source TPM Project for Remote Trust. URL: https://www.youtube.com/watch?v=YtPsruEqGeY (visited on 2023-12-02).
- Luke Hinds. Keylime Demo: Remote Trust for IoT, edge, and cloud. URL: https://www.youtube.com/watch?v=Qhr_aVBCZPw (visited on 2023-12-02).
- Luke Hinds. Keylime: Bootstrapping and Maintaining Trust. URL: https://www.youtube.com/watch?v=xPmv-G5V4I8 (visited on 2023-12-02).
- Alberto Planas. MicroOS Remote Attestation with TPM and Keylime. URL: https://www.youtube.com/watch?v=6F2mxG4YRKg (visited on 2023-12-02).
- Alberto Planas. Remote Attestation in MicroOS. 00:00:00 +0200. URL: https://media.ccc.de/v/3710-remote-attestation-in-microos (visited on 2023-12-02).
- Anderson Sasaki and Thore Sommer. Remote Attestation with Keylime. URL: https://archive.fosdem.org/2023/schedule/event/security_keylime/ (visited on 2023-12-02).
- Thore Sommer. Remote Attestation of the UEFI Event log. URL: https://vimeo.com/770419457 (visited on 2023-12-02).
- Thore Sommer. Writing Digital Exams secured by Remote Attestation and Cloud Computing. URL: https://www.youtube.com/watch?v=EXaPg2Yji4s (visited on 2023-12-02).
- Lily Sturmann and Michael Peters. Keylime: Bootstrap and Maintain Trust on the Edge, Cloud, and IoT. URL: https://www.youtube.com/watch?v=e_g32LxvOCk (visited on 2023-12-02).
- Andrew Toth. Keylime, Securing your Slice of the Cloud. URL: https://www.youtube.com/watch?v=O2x9gwq3BQQ (visited on 2023-12-02).

CHAPTER

FIVE

REST API'S

All Keylime APIs use *REST (Representational State Transfer)*.

Check the *Changelog* section for the differences between versions

5.1 RESTful API for Keylime (v2.1)

The version API version 2.1 was first implemented in Keylime 6.4.0

5.1.1 Verifier

GET /v2.1/agents/{agent_id:UUID}

Get status of agent *agent_id* from Verifier

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "operational_state": 7,
    "v": "yyNnlWwFRz1ZUzSe2YEpz9A5urtv6oywgttTF7VbBP4=",
    "ip": "127.0.0.1",
    "port": 9002,
    "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
    "vtpm_policy": "{\"23\": [\"ffffffffffffffffffffffffffffffffffffffff\", \
↪"0000000000000000000000000000000000000000\"], \"15\": [\
↪"0000000000000000000000000000000000000000\"], \"mask\": \"0x808000\"}",
    "meta_data": "{}",
    "has_mb_refstate": 0,
    "has_runtime_policy": 0,
    "accept_tpm_hash_algs": [
      "sha512",
```


(continues on next page)

(continued from previous page)


```
      "sha384",
      "sha256",
      "sha1"
    ],
    "accept_tpm_encryption_algs": [
      "ecc",
      "rsa"
    ],
    "accept_tpm_signing_algs": [
      "ecschnorr",
      "rsassa"
    ],
    "hash_alg": "sha256",
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "verifier_id": "default",
    "verifier_ip": "127.0.0.1",
    "verifier_port": 8881,
    "severity_level": 6,
    "last_event_id": "qoute_validation.quote_validation",
    "attestation_count": 240,
    "last_received_quote": 1676644582,
    "last_successful_attestation": 1676644462
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **operational_state** (*int*) – Current state of the agent in the CV. Defined in https://github.com/keylime/keylime/blob/master/keylime/common/states.py
- **v** (*string*) – V key for payload base64 encoded or null. Decoded length is 32 bytes
- **ip** (*string*) – Agents contact ip address for the CV
- **port** (*string*) – Agents contact port for the CV
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM
- **vtpm_policy** (*string*) – Static PCR policy and mask for vTPM
- **meta_data** (*string*) – Metadata about the agent. Normally contains certificate information if a CA is used.
- **has_mb_refstate** (*int*) – 1 if a measured boot refstate was provided via tenant, 0 otherwise.
- **has_runtime_policy** (*int*) – 1 if a runtime policy (allowlist and excludelist) was provided via tenant, 0 otherwise.
- **accept_tpm_hash_algs** (*list[string]*) – Accepted TPM hashing algorithms. sha1 must be enabled for IMA validation to work.

- **accept_tpm_encryption_algs** (*list[string]*) – Accepted TPM encryption algorithms.
- **accept_tpm_signing_algs** (*list[string]*) – Accepted TPM signing algorithms.
- **hash_alg** (*string*) – Used hashing algorithm.
- **enc_alg** (*string*) – Used encryption algorithm.
- **sign_alg** (*string*) – Used signing algorithm.
- **verifier_id** (*string*) – Name of the verifier that is used. (Only important if multiple verifiers are used)
- **verifier_ip** (*string*) – IP of the verifier that is used.
- **verifier_port** (*int*) – Port of the verifier that is used.
- **severity_level** (*int*) – Severity level of the agent. Might be *null*. Levels are the numeric representation of the severity labels.
- **last_event_id** (*string*) – ID of the last revocation event. Might be *null*.
- **attestation_count** (*int*) – Number of quotes received from the agent which have verified successfully.
- **last_received_quote** (*int*) – Timestamp of the last quote received from the agent irrespective of validity. A value of 0 indicates no quotes have been received. May be *null* after upgrading from a previous Keylime version.
- **last_successful_attestation** (*int*) – Timestamp of the last quote received from the agent which verified successfully. A value of 0 indicates no valid quotes have been received. May be *null* after upgrading from a previous Keylime version.

POST /v2.1/agents/{agent_id:UUID}

Add new agent *instance_id* to Verifier.

Example request:


```
{
  "v": "3HZMmIEc6yyjfoxdCwcOgPk/6X1GuNG+tlCmNgqBM/I=",
  "cloudagent_ip": "127.0.0.1",
  "cloudagent_port": 9002,
  "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
  "ak_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQDjZ4J2HO7ekIONAX/eYIzt7ziiVAqE/
↪1D7I9oEwIE88dIfqH0FQLJAg8u3+ZOgsJDQr9HiMhZRPhv8hRuia8ULdAomyOFA1cVzlBF+xcPUEemOIofbvcBNAoTY/
↪x49r8LpqAEUBBiUeOniQbjfRaV2S5cEAA92wHLQAPLF9Sbf3zNxCnbhtRkEi6C3NYl8/
↪FJqyu5Z9vvwEBBOFFTPasAxMtPm6a+Z5KJ4rDflipfaVcUvTKLIBRI7wkuXqhTR8BeIByK9upQ3iBo+FbYjWSf+BaN+wodMNg
↪",
  "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
  "runtime_policy_name": null,
```


(continues on next page)

(continued from previous page)

```
  "runtime_policy": "",
  "runtime_policy_sig": "",
  "runtime_policy_key": "",
  "mb_refstate": "null",
  "ima_sign_verification_keys": "[]",
  "metadata": "{\"cert_serial\": 719066720466992686663564415155145407427243959000, \
↪"subject\": \"/C=US/ST=MA/L=Lexington/O=Keylime/OU=53/CN=D432FBB3-D2F1-4A97-9EF7-
↪75BD81C00000\"}",
  "revocation_key": "-----BEGIN PRIVATE KEY----- (...) -----END PRIVATE KEY-----\n",
  "accept_tpm_hash_algs": [
    "sha512",
    "sha384",
    "sha256",
    "sha1"
  ],
  "accept_tpm_encryption_algs": [
    "ecc",
    "rsa"
  ],
  "accept_tpm_signing_algs": [
    "ecschnorr",
    "rsassa"
  ],
  "supported_version": "2.0"
}
```

Request JSON Object

- **v** (*string*) – (Optional) V key for payload base64 encoded. Decoded length is 32 bytes.
- **cloudagent_ip** (*string*) – Agents contact ip address for the CV.
- **cloudagent_port** (*string*) – Agents contact port for the CV.
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **ak_tpm** (*string*) – AK of the agent, base64-encoded, same as *aik_tpm* in the registrar.
- **mtls_cert** (*string*) – MTLS certificate of the agent, PEM encoded, same as in the registrar.
- **runtime_policy_name** (*string*) – Optional. If specified with a *runtime_policy* it is saved under that name, if specified without, then the policy with that name is loaded.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.
- **runtime_policy_sig** (*string*) – Optional runtime policy detached signature, base64-encoded. Must also provide *runtime_policy_key*.
- **runtime_policy_key** (*string*) – Optional runtime policy detached signature key, base64-encoded. Must also provide *runtime_policy_sig*.
- **mb_refstate** (*string*) – Measured boot reference state policy.
- **ima_sign_verification_keys** (*string*) – IMA signature verification public keyring JSON object string encoded.

- **metadata** (*string*) – Metadata about the agent. Contains *cert_serial* and *subject* if a CA is used with the tenant.
- **revocation_key** (*string*) – Key which is used to sign the revocation message of the agent.
- **accept_tpm_hash_algs** (*list[string]*) – Accepted TPM hashing algorithms. sha1 must be enabled for IMA validation to work.
- **accept_tpm_encryption_algs** (*list[string]*) – Accepted TPM encryption algorithms.
- **accept_tpm_signing_algs** (*list[string]*) – Accepted TPM signing algorithms.
- **supported_version** (*string*) – supported API version of the agent. *v* prefix must not be included.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`DELETE /v2.1/agents/{agent_id:UUID}`

Terminate instance *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`PUT /v2.1/agents/{agent_id:UUID}/reactivate`

Start agent *agent_id* (for an already bootstrapped *agent_id* node)

Example response:

```
{
  "code": 200,
  "status": "Success",
```

(continues on next page)

(continued from previous page)

```
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`PUT /v2.1/agents/{agent_id:UUID}/stop`

Stop Verifier polling on *agent_id*, but don't delete (for an already started *agent_id*). This will make the agent verification fail.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`POST /v2.1/allowlists/{runtime_policy_name:string}`

Add new named IMA policy *runtime_policy_name* to Verifier.

Example request:

```
{
  "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
  "runtime_policy": "",
  "runtime_policy_sig": "",
  "runtime_policy_key": ""
}
```

Request JSON Object

- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.

- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.
- **runtime_policy_sig** (*string*) – Optional runtime policy detached signature, base64-encoded. Must also provide *runtime_policy_key*.
- **runtime_policy_key** (*string*) – Optional runtime policy detached signature key, base64-encoded. Must also provide *runtime_policy_sig*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

GET /v2.1/allowlists/[runtime_policy_name:string]

If *runtime_policy_name* is provided, get the named runtime policies from the Verifier.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "name": "",
    "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
    "runtime_policy": ""
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **name** (*string*) – Name of the requested IMA policy.
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.

- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.

Otherwise, retrieve list of names of the runtime policies.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "runtimepolicy names": [
        "runtimepolicyname1",
        "runtimepolicyname2"
    ],
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **names** (*list[string] runtimepolicy*) – List of names of the runtime policies.

DELETE /v2.1/allowlist/{runtime_policy_name:string}

Delete IMA policy *runtime_policy_name*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

5.1.2 Agent

GET /version

Returns what API version the agent supports. This endpoint might not be implemented by all agents.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
```

(continues on next page)

(continued from previous page)

```
    "supported_version": "2.1"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **supported_version** (*string*) – The latest version the agent supports.

`GET /v2.1/keys/pubkey`

Retrieves the agent's public key.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **pubkey** (*string*) – Public rsa key of the agent used for encrypting V and U key.

`POST /v2.1/keys/vkey`

Send *v_key* to node.

Example request:

```
{
  "encrypted_key": "MN/F33jjuLiIuRH8fF7pMtw6Hoe2KG10zg+/xuuZLa5d1WB2aR6feVCwknZDe/
↪dhG51yB0tKau8fCNUz8KMxyWoFkalIY4vVG6DNpLouDjb+vMvI6RmVmCBwO5zx6R802wK2z2yUbcn11TU/
↪k2zHq34CNFIgI5pQu7cnLMzCLW6NLEp8N0IOQL6D+uV9emkheJH1g40xYwUaKoABWjZeaJN5dvKwbkpIf2m+CROmCNPCidh87
↪tZErh1zk+nUamtrgl25pEImw+Cn9RIVTd6fBkmzlGzch5foAqZCyZ0AhQ0ONuWw=="
}
```

Request JSON Object

- **encrypted_key** (*string*) – V key encrypted with the agent's public key base64 encoded.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`POST /v2.1/keys/ukey`

Send *u_key* to node (with optional payload)

Example request:

```
{
  "auth_tag" :
↪"3876c08b30c16c4140ee04300bb4262bbcc9034d8a2ed8c90784f13b484a570bf9da3d5c372141bd16d85de05c4c7cce
↪",
  "encrypted_key":
↪"iAckMZgZc8r43pF0iW8iwwAorD+rvnvF7AShhlz6+am+ryqW+907UynOrWrIrAseyVRE7ouHpr547gnwfF7oKeBFlEdWnE6E
↪y/
↪MmSuNR5pGQwZCueKI0ji3Nqq6heOgSvnMRC0PHgyumOsYiAnbDNyryvfwO4HsqdqMcEsBu1IVzU3EtJWhfQ8i/
↪UpvHy6Jq4bBh+mw5HZwmK93bmsLXNKgjPWAicsCZINUAPVMCUL7dcDd4zijsBxMxiZF7Js7V25wKKFer2zqKsE5omLy9sKotF
↪",
  "payload": "WcXpUr4G9yfvVaojNx6K2XZuDYRkFoZQhHrvZB+TKZqsq41g"
}
```

Request JSON Object

- **auth_tag** (*string*) – HMAC calculated with K key as key and UUID as data, using SHA-384 as the underlying hash algorithm
- **encrypted_key** (*string*) – U key encrypted with the agent's public key base64 encoded
- **payload** (*string*) – (optional) payload encrypted with K key base64 encoded.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

GET /v2.1/keys/verify

Get confirmation of bootstrap key derivation

Example request:

```
GET /v2.2/keys/verify?challenge=1234567890ABCDEFHIJK HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **challenge** – 20 character random string with [a-Z,0-9] as symbols.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "hmac":
↪"719d992fb7d2a0761785fd023fe1cf8a584b835e465e71e2ef2632ff4e9938c080bdefba26194d8ea69dd7f9adee6c18
↪"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **hmac** (*string*) – hmac with K key as key and the challenge

GET /v2.1/quotes/integrity

Get integrity quote from node

Example request:

```
GET /v2.2/quotes/integrity?nonce=1234567890ABCDEFHIJK&mask=0x10401&partial=0 HTTP/1.
↪1
Host: example.com
Accept: application/json
```

Query Parameters

- **nonce** – 20 character random string with [a-Z,0-9] as symbols.
- **mask** – Mask for what PCRs from the TPM are included in the quote.
- **partial** – Is either "0" or "1". If set to "1" the public key is excluded in the response.
- **ima_ml_entry** – (optional) Line offset of the IMA entry list. If not present, 0 is assumed.

Example Response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "quote": "r/
↪1RDR4AYABYABPihP2yz+HcGF0vD0c4qiKt4nvSOAARURVNUAAAAAAAyQ9AAAAAAAAAAAAEgGRAjABY2NgAAAAEABAMAAAEAFC
↪yx6OVUze9jTDvML9xkkK1ghXObCJ5gH+QX0udKfrLacm/
↪iMds28SBtVO0rjqDIoYqGgXhH2ZhwGNDwjRCp6HquvtBe7pGEgtZlxf7Hr3wQRLO3FtliBPBR6gjOo7NC/
↪uGsuPjdPU7c9ls29NgYSqdwShuNdRzwmZrF57umuUgF6GREFlxqLkGcbDIT1itV4zJZtI1caLVxqiH0Qv3sNqlNLsSHggkgc5
↪TsEZOq/
↪leCoLtyVGYghPeGwg0RJfbe8cdyBWCQ6nOA==:AQAAAAQAAwAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪ntmsqy2aDi6NhKnLKz4k4uEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪",
    "hash_alg": "sha256",
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
    "boottime": 123456,
    "ima_measurement_list": "10 367a111b682553da5340f977001689db8366056a ima-ng␣
↪sha256:94c0ac6d0ff747d8f1ca7fac89101a141f3e8f6a2c710717b477a026422766d6 boot_
↪aggregate\n",
    "ima_measurement_list_entry": 0,
    "mb_measurement_list":
↪"AAAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAAAAACEAAABTcGVjIElEIEV2ZW50MDMAAAAAAACAAIBAAAACwAgAAAAAAACAAA
↪[....]"
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **quote** (*string*) – TPM integrity quote
- **hash_alg** (*string*) – Used hash algorithm used in the quote (e.g. sha1, sha256, sha512).
- **enc_alg** (*string*) – Encryption algorithm used in the quote (ecc, rsa).
- **sign_alg** (*string*) – Signing algorthm used in the quote (rsassa, rsapss, ecdsa, ecdaa or ecschnorr).
- **pubkey** (*string*) – PEM encoded public portion of the NK (digest is measured into PCR 16).
- **boottime** (*int*) – Seconds since the system booted
- **ima_measurement_list** (*string*) – (optional) IMA entry list. Is included if *IMA_PCR* (10) is included in the mask
- **ima_measurement_list_entry** (*int*) – (optional) Starting line offset of the IMA entry list returned
- **mb_measurement_list** (*string*) – (optional) UEFI Eventlog list base64 encoded. Is included if PCR 0 is included in the mask

Quote format: The quote field contains the quote, the signature and the PCR values that make up the quote.

```
QUOTE_DATA := rTPM_QUOTE:TPM_SIG:TPM_PCRS
TPM_QUOTE  := base64(TPMS_ATTEST)
TPM_SIG    := base64(TPMT_SIGNATURE)
TPM_PCRS   := base64(tpm2_pcrs) // Can hold more that 8 PCR entries. This is a data␣
↪structure generated by tpm2_quote
```

`GET /v2.1/quotes/identity`

Get identity quote from node

Example request:

```
GET /v2.1/quotes/identity?nonce=1234567890ABCDEFHIJK HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **nonce** – 20 character random string with [a-Z,0-9] as symbols.

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "quote": "r/
↪1RDR4AYABYABPihP2yz+HcGF0vD0c4qiKt4nvSOAARURVNUAAAAAAAyQ9AAAAAAAAAAAAEgGRAjABY2NgAAAAEABAMAAAEAFC
↪yx6OVUze9jTDvML9xkkK1ghXObCJ5gH+QX0udKfrLacm/
↪iMds28SBtVO0rjqDIoYqGgXhH2ZhwGNDwjRCp6HquvtBe7pGEgtZlxf7Hr3wQRLO3FtliBPBR6gjOo7NC/
↪uGsuPjdPU7c9ls29NgYSqdwShuNdRzwmZrF57umuUgF6GREFlxqLkGcbDIT1itV4zJZtI1caLVxqiH0Qv3sNqlNLsSHggkgc5
↪TsEZOq/
↪leCoLtyVGYghPeGwg0RJfbe8cdyBWCQ6nOA==:AQAAAAQAAwAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪ntmsqy2aDi6NhKnLKz4k4uEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪",
    "hash_alg": "sha256",
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
    "boottime": 123456
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **quote** (*string*) – See *quotes/integrity*
- **hash_alg** (*string*) – See *quotes/integrity*
- **enc_alg** (*string*) – See *quotes/integrity*

- **sign_alg** (*string*) – See *quotes/integrity*
- **pubkey** (*string*) – See *quotes/integrity*
- **boottime** (*int*) – See *quotes/integrity*

5.1.3 Registrar

GET /v2.1/agents/

Get ordered list of registered agents

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "uuids": [
      "5e600bce-a5cb-4f5a-bf08-46d0b45081c5",
      "6dab10e4-6619-4ff9-9062-ee6ad23ec24d",
      "d432fbb3-d2f1-4a97-9ef7-75bd81c00000"
    ]
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **uuids** (*list*) – List of registered agents

GET /v2.1/agents/{agent_id:UUID}

Get EK certificate, AIK and optinal contact ip and port of agent *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQDjZ4J2HO7ekIONAX/eYIzt7ziiVAqE/
↪1D7I9oEwIE88dIfqH0FQLJAg8u3+ZOgsJDQr9HiMhZRPhv8hRuia8ULdAomyOFA1cVzlBF+xcPUEemOIofbvcBNAoTY/
↪x49r8LpqAEUBBiUeOniQbjfRaV2S5cEAA92wHLQAPLF9Sbf3zNxCnbhtRkEi6C3NYl8/
↪FJqyu5Z9vvwEBBOFFTPasAxMtPm6a+Z5KJ4rDflipfaVcUvTKLIBRI7wkuXqhTR8BeIByK9upQ3iBo+FbYjWSf+BaN+wodMNg
↪",
    "ek_tpm": "AToAAQALAAMAsgAgg3GXZ0SEs/
↪gakMyNRqXXJP1S124GUgtk8qHaGzMUaaoABgCAAEMAEAgAAAAAAAEA0YwlPPIoXryMvbD5cIokN9OkljL2mV1oDxy7ETBXBe1
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQ==",
    "ekcert":
↪"MIIEGTCCAoGgAwIBAgIBBTANBgkqhkiG9w0BAQsFADAYMRYwFAYDVQQDEw1zd3RwbS1sb2NhbGNhMB4XDTIxMDQwOTEyNDAy
```

(continues on next page)

(continued from previous page)


```
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQIDAQABo4HNMIHKMBAGA1UdJQQJMAcGBWeBB
↪wRIMEakRDBCMRYwFAYFZ4EFAgEMC2lkOjAwMDAxMDE0MRAwDgYFZ4EFAgIMBXN3dHBtMRYwFAYFZ4EFAgMMC2lkOjIwMTkxMD
↪wQCMAAwIgYDVR0JBBswGTAXBgVngQUCEDEOMAwMAzIuMAIBAAICAKIwHwYDVR0jBBgwFoAUaO+9FEi5yX/
↪GEnU+Vc6b3Si6JeAwDwYDVR0PAQH/BAUDAwcgADANBgkqhkiG9w0BAQsFAAOCAYEAaP/jI2i/
↪hXDrthtaZypQ8VUG5AWFnMDtgiMhDSaKwOBfyxiUiYMTggGYXLOXGIu1SJGBtRJsh3QSYgs2tJCnntWF9Jcpmk6kIW/
↪MC8shE+hdu/
↪gQZKjAPZS4QCLIldv+GVZdNYEIv2FYDsKl6Bq1qUsYhAb7z29Nu1itpdvja2qy7ODJ0u+ThccBuH60VGFclFdJg19dvVQMnfi
↪ZPTLNutJHmF0/Vk9W2pRym8SrUe8G6mwxVW8lP9M7fhovKTzoXVFW3gQWQeUxhvWOncXxtARFLp/
↪+f2mzGBRWxIslW17vpZ3QLlCdJ2C7P3U8x2tvkuyyDfz3/
↪pq+8ECupZhdSvpHlBnWvqs1tAWKW0qI9d0xNYjj3Kfl3Lfy7kqqe6FIkvbDlVhw3vnJlclW+M6D86jBulL9ze+3zyMxy2z8m7
↪",
    "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
    "ip": "127.0.0.1",
    "port": 9002,
    "regcount": 1
  }
}
```


Response JSON Object

- **aik_tpm** (*string*) – base64 encoded AIK. The AIK format is TPM2B_PUBLIC from tpm2-tss.
- **ek_tpm** (*string*) – base64 encoded EK. When a *ekcert* is submitted it will be the public key of that certificate.
- **ekcert** (*string*) – base64 encoded EK certificate. Should be in *DER* format. Gets extracted from NV *0x1c00002*.
- **mtls_cert** (*string*) – Agent HTTPS server certificate. PEM encoded.
- **ip** (*string*) – IPv4 address for contacting the agent. Might be *null*.
- **port** (*integer*) – Port for contacting the agent. Might be *null*.

`POST /v2.1/agents/{agent_id:UUID}`

Add agent *agent_id* to registrar.

Example request:


```
{
  "ekcert":
↪"MIIEGTCCAoGgAwIBAgIBBTANBgkqhkiG9w0BAQsFADAYMRYwFAYDVQQDEw1zd3RwbS1sb2NhbGNhMB4XDTIxMDQwOTEyNDAy
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQIDAQABo4HNMIHKMBAGA1UdJQQJMAcGBWeBB
↪wRIMEakRDBCMRYwFAYFZ4EFAgEMC2lkOjAwMDAxMDE0MRAwDgYFZ4EFAgIMBXN3dHBtMRYwFAYFZ4EFAgMMC2lkOjIwMTkxMD
↪wQCMAAwIgYDVR0JBBswGTAXBgVngQUCEDEOMAwMAzIuMAIBAAICAKIwHwYDVR0jBBgwFoAUaO+9FEi5yX/
↪GEnU+Vc6b3Si6JeAwDwYDVR0PAQH/BAUDAwcgADANBgkqhkiG9w0BAQsFAAOCAYEAaP/jI2i/
↪hXDrthtaZypQ8VUG5AWFnMDtgiMhDSaKwOBfyxiUiYMTggGYXLOXGIu1SJGBtRJsh3QSYgs2tJCnntWF9Jcpmk6kIW/
↪MC8shE+hdu/
```


(continues on next page)

(continued from previous page)


```
↪gQZKjAPZS4QCLIldv+GVZdNYEIv2FYDsKl6Bq1qUsYhAb7z29Nu1itpdvja2qy7ODJ0u+ThccBuH60VGFclFdJg19dvVQMnf
↪ZPTLNutJHmF0/Vk9W2pRym8SrUe8G6mwxVW8lP9M7fhovKTzoXVFW3gQWQeUxhvWOncXxtARFLp/
↪+f2mzGBRWxIslW17vpZ3QLlCdJ2C7P3U8x2tvkuyyDfz3/
↪pq+8ECupZhdSvpHlBnWvqs1tAWKW0qI9d0xNYjj3Kfl3Lfy7kqqe6FIkvbDlVhw3vnJlclW+M6D86jBulL9ze+3zyMxy2z8m7
↪",
  "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQCg5mMzNFqdlUbW8uI/
↪GuMcIIvOXXTohHFTas59JlwrJQVed+5klWP+j7tI7492YPmCnoZvP4T4YdT1PN7tHHGfF81AeMnuw5GV5RkW/
↪QeSD+ssB4f6AfuzYJgBkc28zKmpRRHUbwN4rb/
↪HnJgRXdXsuIcnOqGcC39pD0kiu5TrN6hekjxTQtfAbIlQwwDwHCxKWdtH5x7avd15hqc6cBc2gjTQksXrk+OiMwOFTJ68n0qY
↪mVmd8XhPeYUoMlweXBOwc3e9zM9lZmMvregrFHKYc7CXChz",
  "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
  "ip": "127.0.0.1",
  "port": "9002"
}
```


Request JSON Object

- **ekcert** (*string*) – base64 encoded EK certificate. Should be in *DER* format. Gets extracted from NV *0x1c00002*.
- **aik_tpm** (*string*) – base64 encoded AIK. The AIK format is TPM2B_PUBLIC from tpm2-tss.
- **mtls_cert** (*string*) – Agent HTTPS server certificate. PEM encoded.
- **ip** (*string*) – (Optional) contact IPv4 address for the verifier and tenant to use.
- **port** (*string*) – (Optional) contact port for the verifier and tenant to use.

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "blob": "utzA3gAAAAEARAAgC/
↪w9LP1PKZ9thEk+GkMg4m+tkc9TkavcvFiFL6xbXM2q2fTRyKmQnxuCJc0tQdgsRXMftGiKJyA/
↪SUo8kGNVmcNfAQCs79kl9Ir49JJ8rfyMfDIqOuSVlu9PhxGUOeVzAdxyUmPxq5Qp0s431n/KeL/
↪5nUaVXC+qpOftF4bmVtXwLGTTUbKtyT3GG+9ujkjiwHCQhSKTQ8HiuARgXXh13ntFsJ75PBD5dWauLTuciYZI/
↪WQDVXAcgMnQNxodJUi9ir1GxJWz8zufjVQTVjrlgsgeBdOKbB6+H81K1d9prWhZaVLP+wIwO3YuWgtNHNi90E1z/
↪dah2pzfUpLvJo3lNZ4bJgrJUR507AokGKIFm7EfOf+5WWWAvGxGtgqTJB27vgE0CVBLEuDUHoRcLVBi1Np4GGNTByalxbulg8
↪"
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **blob** (*string*) – base64 encoded blob containing the *aik_tpm* name and a challenge. Is encrypted with *ek_tpm*.

`DELETE /v2.1/agents/{agent_id:UUID}`

Remove agent *agent_id* from registrar.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`PUT /v2.1/agents/{agent_id:UUID}/activate`

Activate physical agent *agent_id*

Example request:

```
{
  "auth_tag":
↪"7087ba88746886262de743587ed97aea6b6e3f32755de5d85415c40feef3169bc58d38855ddb96e32efdd8745d0bdfe
↪"
}
```

Request JSON Object

- **auth_tag** (*string*) – hmac containing the challenge from *blob* and the *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

5.2 RESTful API for Keylime (v2.2)

The version API version 2.2 was first implemented in Keylime 7.11.0

5.2.1 Verifier

GET /v2.2/agents/{agent_id:UUID}

Get status of agent *agent_id* from Verifier

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "operational_state": 7,
    "v": "yyNnlWwFRz1ZUzSe2YEpz9A5urtv6oywgttTF7VbBP4=",
    "ip": "127.0.0.1",
    "port": 9002,
    "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
    "vtpm_policy": "{\"23\": [\"ffffffffffffffffffffffffffffffffffffffff\", \
↪"0000000000000000000000000000000000000000\"], \"15\": [\
↪"0000000000000000000000000000000000000000\"], \"mask\": \"0x808000\"}",
    "meta_data": "{}",
    "has_mb_refstate": 0,
    "has_runtime_policy": 0,
    "accept_tpm_hash_algs": [
      "sha512",
      "sha384",
      "sha256",
      "sha1"
    ],
    "accept_tpm_encryption_algs": [
      "ecc",
      "rsa"
    ],
    "accept_tpm_signing_algs": [
      "ecschnorr",
      "rsassa"
    ],
    "hash_alg": "sha256",
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "verifier_id": "default",
    "verifier_ip": "127.0.0.1",
    "verifier_port": 8881,
    "severity_level": 6,
    "last_event_id": "qoute_validation.quote_validation",
    "attestation_count": 240,
```


(continues on next page)

(continued from previous page)

```
    "last_received_quote": 1676644582,
    "last_successful_attestation": 1676644462
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **operational_state** (*int*) – Current state of the agent in the CV. Defined in https://github.com/keylime/keylime/blob/master/keylime/common/states.py
- **v** (*string*) – V key for payload base64 encoded or null. Decoded length is 32 bytes
- **ip** (*string*) – Agents contact ip address for the CV
- **port** (*string*) – Agents contact port for the CV
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM
- **vtpm_policy** (*string*) – Static PCR policy and mask for vTPM
- **meta_data** (*string*) – Metadata about the agent. Normally contains certificate information if a CA is used.
- **has_mb_refstate** (*int*) – 1 if a measured boot refstate was provided via tenant, 0 otherwise.
- **has_runtime_policy** (*int*) – 1 if a runtime policy (allowlist and excludelist) was provided via tenant, 0 otherwise.
- **accept_tpm_hash_algs** (*list[string]*) – Accepted TPM hashing algorithms. sha1 must be enabled for IMA validation to work.
- **accept_tpm_encryption_algs** (*list[string]*) – Accepted TPM encryption algorithms.
- **accept_tpm_signing_algs** (*list[string]*) – Accepted TPM signing algorithms.
- **hash_alg** (*string*) – Used hashing algorithm.
- **enc_alg** (*string*) – Used encryption algorithm.
- **sign_alg** (*string*) – Used signing algorithm.
- **verifier_id** (*string*) – Name of the verifier that is used. (Only important if multiple verifiers are used)
- **verifier_ip** (*string*) – IP of the verifier that is used.
- **verifier_port** (*int*) – Port of the verifier that is used.
- **severity_level** (*int*) – Severity level of the agent. Might be *null*. Levels are the numeric representation of the severity labels.
- **last_event_id** (*string*) – ID of the last revocation event. Might be *null*.
- **attestation_count** (*int*) – Number of quotes received from the agent which have verified successfully.

- **last_received_quote** (*int*) – Timestamp of the last quote received from the agent irrespective of validity. A value of 0 indicates no quotes have been received. May be *null* after upgrading from a previous Keylime version.
- **last_successful_attestation** (*int*) – Timestamp of the last quote received from the agent which verified successfully. A value of 0 indicates no valid quotes have been received. May be *null* after upgrading from a previous Keylime version.

POST /v2.2/agents/{agent_id:UUID}

Add new agent *instance_id* to Verifier.

Example request:

```
{
  "v": "3HZMmIEc6yyjfoxdCwcOgPk/6X1GuNG+tlCmNgqBM/I=",
  "cloudagent_ip": "127.0.0.1",
  "cloudagent_port": 9002,
  "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
  "ak_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQDjZ4J2HO7ekIONAX/eYIzt7ziiVAqE/
↪1D7I9oEwIE88dIfqH0FQLJAg8u3+ZOgsJDQr9HiMhZRPhv8hRuia8ULdAomyOFA1cVzlBF+xcPUEemOIofbvcBNAoTY/
↪x49r8LpqAEUBBiUeOniQbjfRaV2S5cEAA92wHLQAPLF9Sbf3zNxCnbhtRkEi6C3NYl8/
↪FJqyu5Z9vvwEBBOFFTPasAxMtPm6a+Z5KJ4rDflipfaVcUvTKLIBRI7wkuXqhTR8BeIByK9upQ3iBo+FbYjWSf+BaN+wodMNg
↪",
  "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
  "runtime_policy_name": null,
  "runtime_policy": "",
  "runtime_policy_sig": "",
  "runtime_policy_key": "",
  "mb_refstate": "null",
  "ima_sign_verification_keys": "[]",
  "metadata": "{\"cert_serial\": 719066720466992686663564415155145407427243959000, \
↪"subject\": \"/C=US/ST=MA/L=Lexington/O=Keylime/OU=53/CN=D432FBB3-D2F1-4A97-9EF7-
↪75BD81C00000\"}",
  "revocation_key": "-----BEGIN PRIVATE KEY----- (...) -----END PRIVATE KEY-----\n",
  "accept_tpm_hash_algs": [
    "sha512",
    "sha384",
    "sha256",
    "sha1"
  ],
  "accept_tpm_encryption_algs": [
    "ecc",
    "rsa"
  ],
  "accept_tpm_signing_algs": [
```

(continues on next page)

(continued from previous page)

```
    "ecschnorr",
    "rsassa"
  ],
  "supported_version": "2.0"
}
```

Request JSON Object

- **v** (*string*) – (Optional) V key for payload base64 encoded. Decoded length is 32 bytes.
- **cloudagent_ip** (*string*) – Agents contact ip address for the CV.
- **cloudagent_port** (*string*) – Agents contact port for the CV.
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **ak_tpm** (*string*) – AK of the agent, base64-encoded, same as *aik_tpm* in the registrar.
- **mtls_cert** (*string*) – MTLS certificate of the agent, PEM encoded, same as in the registrar.
- **runtime_policy_name** (*string*) – Optional. If specified with a *runtime_policy* it is saved under that name, if specified without, then the policy with that name is loaded.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.
- **runtime_policy_sig** (*string*) – Optional runtime policy detached signature, base64-encoded. Must also provide *runtime_policy_key*.
- **runtime_policy_key** (*string*) – Optional runtime policy detached signature key, base64-encoded. Must also provide *runtime_policy_sig*.
- **mb_refstate** (*string*) – Measured boot reference state policy.
- **ima_sign_verification_keys** (*string*) – IMA signature verification public keyring JSON object string encoded.
- **metadata** (*string*) – Metadata about the agent. Contains *cert_serial* and *subject* if a CA is used with the tenant.
- **revocation_key** (*string*) – Key which is used to sign the revocation message of the agent.
- **accept_tpm_hash_algs** (*list[string]*) – Accepted TPM hashing algorithms. sha1 must be enabled for IMA validation to work.
- **accept_tpm_encryption_algs** (*list[string]*) – Accepted TPM encryption algorithms.
- **accept_tpm_signing_algs** (*list[string]*) – Accepted TPM signing algorithms.
- **supported_version** (*string*) – supported API version of the agent. *v* prefix must not be included.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`DELETE /v2.2/agents/{agent_id:UUID}`

Terminate instance *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`PUT /v2.2/agents/{agent_id:UUID}/reactivate`

Start agent *agent_id* (for an already bootstrapped *agent_id* node)

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`PUT /v2.2/agents/{agent_id:UUID}/stop`

Stop Verifier polling on *agent_id*, but don't delete (for an already started *agent_id*). This will make the agent verification fail.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`POST /v2.2/allowlists/{runtime_policy_name:string}`

Add new named IMA policy *runtime_policy_name* to Verifier.

Example request:

```
{
  "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
  "runtime_policy": "",
  "runtime_policy_sig": "",
  "runtime_policy_key": ""
}
```

Request JSON Object

- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.
- **runtime_policy_sig** (*string*) – Optional runtime policy detached signature, base64-encoded. Must also provide *runtime_policy_key*.
- **runtime_policy_key** (*string*) – Optional runtime policy detached signature key, base64-encoded. Must also provide *runtime_policy_sig*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

GET /v2.2/allowlists/[runtime_policy_name:string]

If *runtime_policy_name* is provided, get the named runtime policies from the Verifier.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "name": "",
    "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
    "runtime_policy": ""
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **name** (*string*) – Name of the requested IMA policy.
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.

Otherwise, retrieve list of names of the runtime policies.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "runtimepolicy names": [
        "runtimepolicyname1",
        "runtimepolicyname2"
    ],
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string

- **results** (*object*) – Results as a JSON object
- **names** (*list[string] runtimepolicy*) – List of names of the runtime policies.

`DELETE /v2.2/allowlist/{runtime_policy_name:string}`

Delete IMA policy *runtime_policy_name*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`GET /v2.2/verify/identity`

Verify the identity of a node monitored by keylime

Example request:

```
GET /v2.2/verify/identity?agent_uuid=e1ef9f28-be55-47b0-a6c1-8bef90294b93&hash_
↪alg=sha256&nonce=DGHFH6EQVYGKP7YHNVEAFQQR5TN4W4JA&quote=r/1RDR4AYACIACzy[...]␣
↪HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **agent_uuid** – The UUID of the Agent being verified.
- **hash_alg** – The hash algorithm used by the Keylime agent and TPM.
- **nonce** – The onetime nonce being used for identity verification.
- **quote** – The TPM quoted nonce from the Keylime agent.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "valid": 1
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string

- **results** (*object*) – Results as a JSON object
- **valid** (*int*) – A boolean 1 for valid, 0 for invalid identity.

5.2.2 Agent

GET /version

Returns what API version the agent supports. This endpoint might not be implemented by all agents.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "supported_version": "2.0"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **supported_version** (*string*) – The latest version the agent supports.

GET /v2.2/agent/info

Retrieves information about an agent

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "agent_uuid": "e1ef9f28-be55-47b0-a6c1-8bef90294b93",
    "tpm_hash_alg": "sha256",
    "tpm_enc_alg": "rsa",
    "tpm_sign_alg": "rsassa",
    "ak_handle": "1078035599"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **agent_uuid** (*string*) – The UUID of the agent.
- **tpm_hash_alg** (*string*) – The hashing algorithm used by this agent's TPM device.
- **tpm_enc_alg** (*string*) – The encryption algorithm used by this agent's TPM device.

- **tpm_sign_alg** (*string*) – The signing algorithm used by this agent's TPM device.
- **ak** (*string*) – The Attestation Key handle of the TPM device used by this agent.

`GET /v2.2/keys/pubkey`

Retrieves the agent's public key.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **pubkey** (*string*) – Public rsa key of the agent used for encrypting V and U key.

`POST /v2.2/keys/vkey`

Send *v_key* to node.

Example request:

```
{
  "encrypted_key": "MN/F33jjuLiIuRH8fF7pMtw6Hoe2KG10zg+/xuuZLa5d1WB2aR6feVCwknZDe/
↪dhG51yB0tKau8fCNUz8KMxyWoFkalIY4vVG6DNpLouDjb+vMvI6RmVmCBwO5zx6R802wK2z2yUbcn11TU/
↪k2zHq34CNFIgI5pQu7cnLMzCLW6NLEp8N0IOQL6D+uV9emkheJH1g40xYwUaKoABWjZeaJN5dvKwbkpIf2m+CROmCNPCidh87
↪tZErh1zk+nUamtrgl25pEImw+Cn9RIVTd6fBkmzlGzch5foAqZCyZ0AhQ0ONuWw=="
}
```

Request JSON Object

- **encrypted_key** (*string*) – V key encrypted with the agent's public key base64 encoded.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`POST /v2.2/keys/ukey`

Send *u_key* to node (with optional payload)

Example request:

```
{
  "auth_tag" :
↪"3876c08b30c16c4140ee04300bb4262bbcc9034d8a2ed8c90784f13b484a570bf9da3d5c372141bd16d85de05c4c7cce
↪",
  "encrypted_key":
↪"iAckMZgZc8r43pF0iW8iwwAorD+rvnvF7AShhlz6+am+ryqW+907UynOrWrIrAseyVRE7ouHpr547gnwfF7oKeBFlEdWnE6F
↪y/
↪MmSuNR5pGQwZCueKI0ji3Nqq6heOgSvnMRC0PHgyumOsYiAnbDNyryvfwO4HsqdqMcEsBu1IVzU3EtJWhfQ8i/
↪UpvHy6Jq4bBh+mw5HZwmK93bmsLXNKgjPWAicsCZINUAPVMCUL7dcDd4zijsBxMxiZF7Js7V25wKKFer2zqKsE5omLy9sKotF
↪",
  "payload": "WcXpUr4G9yfvVaojNx6K2XZuDYRkFoZQhHrvZB+TKZqsq41g"
}
```

Request JSON Object

- **auth_tag** (*string*) – HMAC calculated with K key as key and UUID as data, using SHA-384 as the underlying hash algorithm
- **encrypted_key** (*string*) – U key encrypted with the agent's public key base64 encoded
- **payload** (*string*) – (optional) payload encrypted with K key base64 encoded.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`GET /v2.2/keys/verify`

Get confirmation of bootstrap key derivation

Example request:

```
GET /v2.2/keys/verify?challenge=1234567890ABCDEFHIJK HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **challenge** – 20 character random string with [a-Z,0-9] as symbols.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "hmac":
↪"719d992fb7d2a0761785fd023fe1cf8a584b835e465e71e2ef2632ff4e9938c080bdefba26194d8ea69dd7f9adee6c18
↪"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **hmac** (*string*) – hmac with K key as key and the challenge

`GET /v2.2/quotes/integrity`

Get integrity quote from node

Example request:

```
GET /v2.2/quotes/integrity?nonce=1234567890ABCDEFHIJK&mask=0x10401&partial=0 HTTP/1.
↪1
Host: example.com
Accept: application/json
```

Query Parameters

- **nonce** – 20 character random string with [a-Z,0-9] as symbols.
- **mask** – Mask for what PCRs from the TPM are included in the quote.
- **partial** – Is either "0" or "1". If set to "1" the public key is excluded in the response.
- **ima_ml_entry** – (optional) Line offset of the IMA entry list. If not present, 0 is assumed.

Example Response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "quote": "r/
↪1RDR4AYABYABPihP2yz+HcGF0vD0c4qiKt4nvSOAARURVNUAAAAAAAyQ9AAAAAAAAAAAAEgGRAjABY2NgAAAAEABAMAAAEAFC
↪yx6OVUze9jTDvML9xkkK1ghXObCJ5gH+QX0udKfrLacm/
↪iMds28SBtVO0rjqDIoYqGgXhH2ZhwGNDwjRCp6HquvtBe7pGEgtZlxf7Hr3wQRLO3FtliBPBR6gjOo7NC/
↪uGsuPjdPU7c9ls29NgYSqdwShuNdRzwmZrF57umuUgF6GREFlxqLkGcbDIT1itV4zJZtI1caLVxqiH0Qv3sNqlNLsSHggkgc5
↪TsEZOq/
↪leCoLtyVGYghPeGwg0RJfbe8cdyBWCQ6nOA==:AQAAAAQAAwAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪ntmsqy2aDi6NhKnLKz4k4uEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪",
    "hash_alg": "sha256",
```

(continues on next page)

(continued from previous page)

```
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
    "boottime": 123456,
    "ima_measurement_list": "10 367a111b682553da5340f977001689db8366056a ima-ng␣
↪sha256:94c0ac6d0ff747d8f1ca7fac89101a141f3e8f6a2c710717b477a026422766d6 boot_
↪aggregate\n",
    "ima_measurement_list_entry": 0,
    "mb_measurement_list":
↪"AAAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAAAAACEAAABTcGVjIElEIEV2ZW50MDMAAAAAAAACAAIBAAAACwAgAAAAAAACAA
↪[....]"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **quote** (*string*) – TPM integrity quote
- **hash_alg** (*string*) – Used hash algorithm used in the quote (e.g. sha1, sha256, sha512).
- **enc_alg** (*string*) – Encryption algorithm used in the quote (ecc, rsa).
- **sign_alg** (*string*) – Signing algorthm used in the quote (rsassa, rsapss, ecdsa, ecdaa or ecschnorr).
- **pubkey** (*string*) – PEM encoded public portion of the NK (digest is measured into PCR 16).
- **boottime** (*int*) – Seconds since the system booted
- **ima_measurement_list** (*string*) – (optional) IMA entry list. Is included if *IMA_PCR* (10) is included in the mask
- **ima_measurement_list_entry** (*int*) – (optional) Starting line offset of the IMA entry list returned
- **mb_measurement_list** (*string*) – (optional) UEFI Eventlog list base64 encoded. Is included if PCR 0 is included in the mask

Quote format: The quote field contains the quote, the signature and the PCR values that make up the quote.

```
QUOTE_DATA := rTPM_QUOTE:TPM_SIG:TPM_PCRS
TPM_QUOTE  := base64(TPMS_ATTEST)
TPM_SIG    := base64(TPMT_SIGNATURE)
TPM_PCRS   := base64(tpm2_pcrs) // Can hold more that 8 PCR entries. This is a data␣
↪structure generated by tpm2_quote
```

GET /v2.2/quotes/identity

Get identity quote from node

Example request:

```
GET /v2.1/quotes/identity?nonce=1234567890ABCDEFHIJK HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **nonce** – 20 character random string with [a-Z,0-9] as symbols.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "quote": "r/
↪1RDR4AYABYABPihP2yz+HcGF0vD0c4qiKt4nvSOAARURVNUAAAAAAAyQ9AAAAAAAAAAAAEgGRAjABY2NgAAAAEABAMAAAEAFC
↪yx6OVUze9jTDvML9xkkK1ghXObCJ5gH+QX0udKfrLacm/
↪iMds28SBtVO0rjqDIoYqGgXhH2ZhwGNDwjRCp6HquvtBe7pGEgtZlxf7Hr3wQRLO3FtliBPBR6gjOo7NC/
↪uGsuPjdPU7c9ls29NgYSqdwShuNdRzwmZrF57umuUgF6GREFlxqLkGcbDIT1itV4zJZtI1caLVxqiH0Qv3sNqlNLsSHggkgc5
↪TsEZOq/
↪leCoLtyVGYghPeGwg0RJfbe8cdyBWCQ6nOA==:AQAAAAQAAwAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪ntmsqy2aDi6NhKnLKz4k4uEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪",
    "hash_alg": "sha256",
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
    "boottime": 123456
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **quote** (*string*) – See *quotes/integrity*
- **hash_alg** (*string*) – See *quotes/integrity*
- **enc_alg** (*string*) – See *quotes/integrity*
- **sign_alg** (*string*) – See *quotes/integrity*
- **pubkey** (*string*) – See *quotes/integrity*
- **boottime** (*int*) – See *quotes/integrity*

5.2.3 Registrar

GET /v2.2/agents/

Get ordered list of registered agents

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "uuids": [
      "5e600bce-a5cb-4f5a-bf08-46d0b45081c5",
      "6dab10e4-6619-4ff9-9062-ee6ad23ec24d",
      "d432fbb3-d2f1-4a97-9ef7-75bd81c00000"
    ]
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **uuids** (*list*) – List of registered agents

`GET /v2.2/agents/{agent_id:UUID}`

Get EK certificate, AIK and optinal contact ip and port of agent *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQDjZ4J2HO7ekIONAX/eYIzt7ziiVAqE/
↪1D7I9oEwIE88dIfqH0FQLJAg8u3+ZOgsJDQr9HiMhZRPhv8hRuia8ULdAomyOFA1cVzlBF+xcPUEemOIofbvcBNAoTY/
↪x49r8LpqAEUBBiUeOniQbjfRaV2S5cEAA92wHLQAPLF9Sbf3zNxCnbhtRkEi6C3NYl8/
↪FJqyu5Z9vvwEBBOFFTPasAxMtPm6a+Z5KJ4rDflipfaVcUvTKLIBRI7wkuXqhTR8BeIByK9upQ3iBo+FbYjWSf+BaN+wodMNg
↪",
    "ek_tpm": "AToAAQALAAMAsgAgg3GXZ0SEs/
↪gakMyNRqXXJP1S124GUgtk8qHaGzMUaaoABgCAAEMAEAgAAAAAAAEA0YwlPPIoXryMvbD5cIokN9OkljL2mV1oDxy7ETBXBe1
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQ==",
    "ekcert":
↪"MIIEGTCCAoGgAwIBAgIBBTANBgkqhkiG9w0BAQsFADAYMRYwFAYDVQQDEw1zd3RwbS1sb2NhbGNhMB4XDTIxMDQwOTEyNDAy
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQIDAQABo4HNMIHKMBAGA1UdJQQJMAcGBWeBB
↪wRIMEakRDBCMRYwFAYFZ4EFAgEMC2lkOjAwMDAxMDE0MRAwDgYFZ4EFAgIMBXN3dHBtMRYwFAYFZ4EFAgMMC2lkOjIwMTkxMD
↪wQCMAAwIgYDVR0JBBswGTAXBgVngQUCEDEOMAwMAzIuMAIBAAICAKIwHwYDVR0jBBgwFoAUaO+9FEi5yX/
↪GEnU+Vc6b3Si6JeAwDwYDVR0PAQH/BAUDAwcgADANBgkqhkiG9w0BAQsFAAOCAYEAaP/jI2i/
↪hXDrthtaZypQ8VUG5AWFnMDtgiMhDSaKwOBfyxiUiYMTggGYXLOXGIu1SJGBtRJsh3QSYgs2tJCnntWF9Jcpmk6kIW/
↪MC8shE+hdu/
↪gQZKjAPZS4QCLIldv+GVZdNYEIv2FYDsKl6Bq1qUsYhAb7z29Nu1itpdvja2qy7ODJ0u+ThccBuH60VGFclFdJg19dvVQMnfi
↪ZPTLNutJHmF0/Vk9W2pRym8SrUe8G6mwxVW8lP9M7fhovKTzoXVFW3gQWQeUxhvWOncXxtARFLp/
```

(continues on next page)

(continued from previous page)

```
↪+f2mzGBRWxIslW17vpZ3QLlCdJ2C7P3U8x2tvkuyyDfz3/
↪pq+8ECupZhdSvpHlBnWvqs1tAWKW0qI9d0xNYjj3Kfl3Lfy7kqqe6FIkvbDlVhw3vnJlclW+M6D86jBulL9ze+3zyMxy2z8m7
↪",
    "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
    "ip": "127.0.0.1",
    "port": 9002,
    "regcount": 1
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **aik_tpm** (*string*) – base64 encoded AIK. The AIK format is TPM2B_PUBLIC from tpm2-tss.
- **ek_tpm** (*string*) – base64 encoded EK. When a *ekcert* is submitted it will be the public key of that certificate.
- **ekcert** (*string*) – base64 encoded EK certificate. Should be in *DER* format. Gets extracted from NV *0x1c00002*.
- **mtls_cert** (*string*) – Agent HTTPS server certificate. PEM encoded.
- **ip** (*string*) – IPv4 address for contacting the agent. Might be *null*.
- **port** (*integer*) – Port for contacting the agent. Might be *null*.

POST /v2.2/agents/{agent_id:UUID}

Add agent *agent_id* to registrar.

Example request:

```
{
  "ekcert":
↪"MIIEGTCCAoGgAwIBAgIBBTANBgkqhkiG9w0BAQsFADAYMRYwFAYDVQQDEw1zd3RwbS1sb2NhbGNhMB4XDTIxMDQwOTEyNDAy
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQIDAQABo4HNMIHKMBAGA1UdJQQJMAcGBWeBB
↪wRIMEakRDBCMRYwFAYFZ4EFAgEMC2lkOjAwMDAxMDE0MRAwDgYFZ4EFAgIMBXN3dHBtMRYwFAYFZ4EFAgMMC2lkOjIwMTkxMD
↪wQCMAAwIgYDVR0JBBswGTAXBgVngQUCEDEOMAwMAzIuMAIBAAICAKIwHwYDVR0jBBgwFoAUaO+9FEi5yX/
↪GEnU+Vc6b3Si6JeAwDwYDVR0PAQH/BAUDAwcgADANBgkqhkiG9w0BAQsFAAOCAYEAaP/jI2i/
↪hXDrthtaZypQ8VUG5AWFnMDtgiMhDSaKwOBfyxiUiYMTggGYXLOXGIu1SJGBtRJsh3QSYgs2tJCnntWF9Jcpmk6kIW/
↪MC8shE+hdu/
↪gQZKjAPZS4QCLIldv+GVZdNYEIv2FYDsKl6Bq1qUsYhAb7z29Nu1itpdvja2qy7ODJ0u+ThccBuH60VGFclFdJg19dvVQMnf
↪ZPTLNutJHmF0/Vk9W2pRym8SrUe8G6mwxVW8lP9M7fhovKTzoXVFW3gQWQeUxhvWOncXxtARFLp/
↪+f2mzGBRWxIslW17vpZ3QLlCdJ2C7P3U8x2tvkuyyDfz3/
↪pq+8ECupZhdSvpHlBnWvqs1tAWKW0qI9d0xNYjj3Kfl3Lfy7kqqe6FIkvbDlVhw3vnJlclW+M6D86jBulL9ze+3zyMxy2z8m7
↪",
  "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQCg5mMzNFqdlUbW8uI/
```

(continues on next page)

(continued from previous page)


```
↪GuMcIIvOXXTohHFTas59JlwrJQVed+5klWP+j7tI7492YPmCnoZvP4T4YdT1PN7tHHGfF81AeMnuw5GV5RkW/
↪QeSD+ssB4f6AfuzYJgBkc28zKmpRRHUbwN4rb/
↪HnJgRXdXsuIcnOqGcC39pD0kiu5TrN6hekjxTQtfAbIlQwwDwHCxKWdtH5x7avd15hqc6cBc2gjTQksXrk+OiMwOFTJ68n0qY
↪mVmd8XhPeYUoMlweXBOwc3e9zM9lZmMvregrFHKYc7CXChz",
  "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
  "ip": "127.0.0.1",
  "port": "9002"
}
```


Request JSON Object

- **ekcert** (*string*) – base64 encoded EK certificate. Should be in *DER* format. Gets extracted from NV *0x1c00002*.
- **aik_tpm** (*string*) – base64 encoded AIK. The AIK format is TPM2B_PUBLIC from tpm2-tss.
- **mtls_cert** (*string*) – Agent HTTPS server certificate. PEM encoded.
- **ip** (*string*) – (Optional) contact IPv4 address for the verifier and tenant to use.
- **port** (*string*) – (Optional) contact port for the verifier and tenant to use.

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "blob": "utzA3gAAAAEARAAgC/
↪w9LP1PKZ9thEk+GkMg4m+tkc9TkavcvFiFL6xbXM2q2fTRyKmQnxuCJc0tQdgsRXMftGiKJyA/
↪SUo8kGNVmcNfAQCs79kl9Ir49JJ8rfyMfDIqOuSVlu9PhxGUOeVzAdxyUmPxq5Qp0s431n/KeL/
↪5nUaVXC+qpOftF4bmVtXwLGTTUbKtyT3GG+9ujkjiwHCQhSKTQ8HiuARgXXh13ntFsJ75PBD5dWauLTuciYZI/
↪WQDVXAcgMnQNxodJUi9ir1GxJWz8zufjVQTVjrlgsgeBdOKbB6+H81K1d9prWhZaVLP+wIwO3YuWgtNHNi90E1z/
↪dah2pzfUpLvJo3lNZ4bJgrJUR507AokGKIFm7EfOf+5WWWAvGxGtgqTJB27vgE0CVBLEuDUHoRcLVBi1Np4GGNTByalxbulg8
↪"
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **blob** (*string*) – base64 encoded blob containing the *aik_tpm* name and a challenge. Is encrypted with *ek_tpm*.

`DELETE /v2.2/agents/{agent_id:UUID}`

Remove agent *agent_id* from registrar.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`PUT /v2.2/agents/{agent_id:UUID}/activate`

Activate physical agent *agent_id*

Example request:

```
{
  "auth_tag":
↪"7087ba88746886262de743587ed97aea6b6e3f32755de5d85415c40feef3169bc58d38855ddb96e32efdd8745d0bdfe
↪"
}
```

Request JSON Object

- **auth_tag** (*string*) – hmac containing the challenge from *blob* and the *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

5.3 RESTful API for Keylime (v2.3)

The version API version 2.3 was first implemented in Keylime 7.12.0

5.3.1 Verifier

`GET /v2.3/agents/{agent_id:UUID}`

Get status of agent *agent_id* from Verifier

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "operational_state": 7,
    "v": "yyNnlWwFRz1ZUzSe2YEpz9A5urtv6oywgttTF7VbBP4=",
    "ip": "127.0.0.1",
    "port": 9002,
    "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
    "vtpm_policy": "{\"23\": [\"ffffffffffffffffffffffffffffffffffffffff\", \
↪"0000000000000000000000000000000000000000\"], \"15\": [\
↪"0000000000000000000000000000000000000000\"], \"mask\": \"0x808000\"}",
    "meta_data": "{}",
    "has_mb_refstate": 0,
    "has_runtime_policy": 0,
    "accept_tpm_hash_algs": [
      "sha512",
      "sha384",
      "sha256",
      "sha1"
    ],
    "accept_tpm_encryption_algs": [
      "ecc",
      "rsa"
    ],
    "accept_tpm_signing_algs": [
      "ecschnorr",
      "rsassa"
    ],
    "hash_alg": "sha256",
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "verifier_id": "default",
    "verifier_ip": "127.0.0.1",
    "verifier_port": 8881,
    "severity_level": 6,
    "last_event_id": "qoute_validation.quote_validation",
    "attestation_count": 240,
    "last_received_quote": 1676644582,
    "last_successful_attestation": 1676644462
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **operational_state** (*int*) – Current state of the agent in the CV. Defined in https://github.com/keylime/keylime/blob/master/keylime/common/states.py
- **v** (*string*) – V key for payload base64 encoded or null. Decoded length is 32 bytes
- **ip** (*string*) – Agents contact ip address for the CV
- **port** (*string*) – Agents contact port for the CV
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM
- **vtpm_policy** (*string*) – Static PCR policy and mask for vTPM
- **meta_data** (*string*) – Metadata about the agent. Normally contains certificate information if a CA is used.
- **has_mb_refstate** (*int*) – 1 if a measured boot refstate was provided via tenant, 0 otherwise.
- **has_runtime_policy** (*int*) – 1 if a runtime policy (allowlist and excludelist) was provided via tenant, 0 otherwise.
- **accept_tpm_hash_algs** (*list[string]*) – Accepted TPM hashing algorithms. sha1 must be enabled for IMA validation to work.
- **accept_tpm_encryption_algs** (*list[string]*) – Accepted TPM encryption algorithms.
- **accept_tpm_signing_algs** (*list[string]*) – Accepted TPM signing algorithms.
- **hash_alg** (*string*) – Used hashing algorithm.
- **enc_alg** (*string*) – Used encryption algorithm.
- **sign_alg** (*string*) – Used signing algorithm.
- **verifier_id** (*string*) – Name of the verifier that is used. (Only important if multiple verifiers are used)
- **verifier_ip** (*string*) – IP of the verifier that is used.
- **verifier_port** (*int*) – Port of the verifier that is used.
- **severity_level** (*int*) – Severity level of the agent. Might be *null*. Levels are the numeric representation of the severity labels.
- **last_event_id** (*string*) – ID of the last revocation event. Might be *null*.
- **attestation_count** (*int*) – Number of quotes received from the agent which have verified successfully.
- **last_received_quote** (*int*) – Timestamp of the last quote received from the agent irrespective of validity. A value of 0 indicates no quotes have been received. May be *null* after upgrading from a previous Keylime version.
- **last_successful_attestation** (*int*) – Timestamp of the last quote received from the agent which verified successfully. A value of 0 indicates no valid quotes have been received. May be *null* after upgrading from a previous Keylime version.

POST /v2.3/agents/{agent_id:UUID}

Add new agent *instance_id* to Verifier.

Example request:


```
{
  "v": "3HZMmIEc6yyjfoxdCwcOgPk/6X1GuNG+tlCmNgqBM/I=",
  "cloudagent_ip": "127.0.0.1",
  "cloudagent_port": 9002,
  "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
  "ak_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQDjZ4J2HO7ekIONAX/eYIzt7ziiVAqE/
↪1D7I9oEwIE88dIfqH0FQLJAg8u3+ZOgsJDQr9HiMhZRPhv8hRuia8ULdAomyOFA1cVzlBF+xcPUEemOIofbvcBNAoTY/
↪x49r8LpqAEUBBiUeOniQbjfRaV2S5cEAA92wHLQAPLF9Sbf3zNxCnbhtRkEi6C3NYl8/
↪FJqyu5Z9vvwEBBOFFTPasAxMtPm6a+Z5KJ4rDflipfaVcUvTKLIBRI7wkuXqhTR8BeIByK9upQ3iBo+FbYjWSf+BaN+wodMNg
↪",
  "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
  "runtime_policy_name": null,
  "runtime_policy": "",
  "runtime_policy_sig": "",
  "runtime_policy_key": "",
  "mb_refstate": "null",
  "ima_sign_verification_keys": "[]",
  "metadata": "{\"cert_serial\": 71906672046699268666356441515514540742724395900, \
↪"subject\": \"/C=US/ST=MA/L=Lexington/O=Keylime/OU=53/CN=D432FBB3-D2F1-4A97-9EF7-
↪75BD81C00000\"}",
  "revocation_key": "-----BEGIN PRIVATE KEY----- (...) -----END PRIVATE KEY-----\n",
  "accept_tpm_hash_algs": [
    "sha512",
    "sha384",
    "sha256",
    "sha1"
  ],
  "accept_tpm_encryption_algs": [
    "ecc",
    "rsa"
  ],
  "accept_tpm_signing_algs": [
    "ecschnorr",
    "rsassa"
  ],
  "supported_version": "2.0"
}
```


Request JSON Object

- **v** (*string*) – (Optional) V key for payload base64 encoded. Decoded length is 32 bytes.
- **cloudagent_ip** (*string*) – Agents contact ip address for the CV.
- **cloudagent_port** (*string*) – Agents contact port for the CV.
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **ak_tpm** (*string*) – AK of the agent, base64-encoded, same as *aik_tpm* in the registrar.
- **mtls_cert** (*string*) – MTLS certificate of the agent, PEM encoded, same as in the registrar.
- **runtime_policy_name** (*string*) – Optional. If specified with a *runtime_policy* it is saved under that name, if specified without, then the policy with that name is loaded.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.
- **runtime_policy_sig** (*string*) – Optional runtime policy detached signature, base64-encoded. Must also provide *runtime_policy_key*.
- **runtime_policy_key** (*string*) – Optional runtime policy detached signature key, base64-encoded. Must also provide *runtime_policy_sig*.
- **mb_refstate** (*string*) – Measured boot reference state policy.
- **ima_sign_verification_keys** (*string*) – IMA signature verification public keyring JSON object string encoded.
- **metadata** (*string*) – Metadata about the agent. Contains *cert_serial* and *subject* if a CA is used with the tenant.
- **revocation_key** (*string*) – Key which is used to sign the revocation message of the agent.
- **accept_tpm_hash_algs** (*list[string]*) – Accepted TPM hashing algorithms. sha1 must be enabled for IMA validation to work.
- **accept_tpm_encryption_algs** (*list[string]*) – Accepted TPM encryption algorithms.
- **accept_tpm_signing_algs** (*list[string]*) – Accepted TPM signing algorithms.
- **supported_version** (*string*) – supported API version of the agent. *v* prefix must not be included.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`DELETE /v2.3/agents/{agent_id:UUID}`

Terminate instance *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

`PUT /v2.3/agents/{agent_id:UUID}/reactivate`

Start agent *agent_id* (for an already bootstrapped *agent_id* node)

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`PUT /v2.3/agents/{agent_id:UUID}/stop`

Stop Verifier polling on *agent_id*, but don't delete (for an already started *agent_id*). This will make the agent verification fail.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`POST /v2.3/allowlists/{runtime_policy_name:string}`

Add new named IMA policy *runtime_policy_name* to Verifier.

Example request:

```
{
  "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
  "runtime_policy": "",
  "runtime_policy_sig": "",
  "runtime_policy_key": ""
}
```


Request JSON Object

- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.
- **runtime_policy_sig** (*string*) – Optional runtime policy detached signature, base64-encoded. Must also provide *runtime_policy_key*.
- **runtime_policy_key** (*string*) – Optional runtime policy detached signature key, base64-encoded. Must also provide *runtime_policy_sig*.

`GET /v2.3/allowlists/[runtime_policy_name:string]`

If *runtime_policy_name* is provided, get the named runtime policies from the Verifier.

Example responses:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "name": "",
    "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
    "runtime_policy": ""
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **name** (*string*) – Name of the requested IMA policy.
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.

Otherwise, retrieve list of names of the runtime policies.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "runtimepolicy names": [
        "runtimepolicyname1",
        "runtimepolicyname2"
    ],
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **names** (*list[string] runtimepolicy*) – List of names of the runtime policies.

`DELETE /v2.3/allowlist/{runtime_policy_name:string}`

Delete IMA policy *runtime_policy_name*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`GET /v2.3/verify/identity`

Verify the identity of a node monitored by keylime

Example request:

```
GET /v2.2/verify/identity?agent_uuid=e1ef9f28-be55-47b0-a6c1-8bef90294b93&hash_
↪alg=sha256&nonce=DGHFH6EQVYGKP7YHNVEAFQQR5TN4W4JA&quote=r/1RDR4AYACIACzy[...]␣
↪HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **agent_uuid** – The UUID of the Agent being verified.
- **hash_alg** – The hash algorithm used by the Keylime agent and TPM.
- **nonce** – The onetime nonce being used for identity verification.
- **quote** – The TPM quoted nonce from the Keylime agent.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "valid": 1
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object

Request JSON Object

- **valid** (*int*) – A boolean 1 for valid, 0 for invalid identity.

`GET /v2.3/mbpolicies/{policy_name:string}`

Get the measured boot policy named *policy_name*

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code

- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object

Request JSON Object

- **valid** (*int*) – A boolean 1 for valid, 0 for invalid identity.

5.3.2 Agent

GET /version

Returns what API version the agent supports. This endpoint might not be implemented by all agents.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "supported_version": "2.0"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **supported_version** (*string*) – The latest version the agent supports.

GET /v2.3/agent/info

Retrieves information about an agent

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "agent_uuid": "e1ef9f28-be55-47b0-a6c1-8bef90294b93",
    "tpm_hash_alg": "sha256",
    "tpm_enc_alg": "rsa",
    "tpm_sign_alg": "rsassa",
    "ak_handle": "1078035599"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **agent_uuid** (*string*) – The UUID of the agent.

- **tpm_hash_alg** (*string*) – The hashing algorithm used by this agent's TPM device.
- **tpm_enc_alg** (*string*) – The encryption algorithm used by this agent's TPM device.
- **tpm_sign_alg** (*string*) – The signing algorithm used by this agent's TPM device.
- **ak** (*string*) – The Attestation Key handle of the TPM device used by this agent.

GET /v2.3/keys/pubkey

Retrieves the agent's public key.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **pubkey** (*string*) – Public rsa key of the agent used for encrypting V and U key.

POST /v2.3/keys/vkey

Send *v_key* to node.

Example request:

```
{
  "encrypted_key": "MN/F33jjuLiIuRH8fF7pMtw6Hoe2KG10zg+/xuuZLa5d1WB2aR6feVCwknZDe/
↪dhG51yB0tKau8fCNUz8KMxyWoFkalIY4vVG6DNpLouDjb+vMvI6RmVmCBwO5zx6R802wK2z2yUbcn11TU/
↪k2zHq34CNFIgI5pQu7cnLMzCLW6NLEp8N0IOQL6D+uV9emkheJH1g40xYwUaKoABWjZeaJN5dvKwbkpIf2m+CROmCNPCidh87
↪tZErh1zk+nUamtrgl25pEImw+Cn9RIVTd6fBkmzlGzch5foAqZCyZ0AhQ0ONuWw=="
}
```

Request JSON Object

- **encrypted_key** (*string*) – V key encrypted with the agent's public key base64 encoded.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code

- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`POST /v2.3/keys/ukey`

Send *u_key* to node (with optional payload)

Example request:

```
{
  "auth_tag" :
↪"3876c08b30c16c4140ee04300bb4262bbcc9034d8a2ed8c90784f13b484a570bf9da3d5c372141bd16d85de05c4c7cce
↪",
  "encrypted_key":
↪"iAckMZgZc8r43pF0iW8iwwAorD+rvnvF7AShhlz6+am+ryqW+907UynOrWrIrAseyVRE7ouHpr547gnwfF7oKeBFlEdWnE6F
↪y/
↪MmSuNR5pGQwZCueKI0ji3Nqq6heOgSvnMRC0PHgyumOsYiAnbDNyryvfwO4HsqdqMcEsBu1IVzU3EtJWhfQ8i/
↪UpvHy6Jq4bBh+mw5HZwmK93bmsLXNKgjPWAicsCZINUAPVMCUL7dcDd4zijsBxMxiZF7Js7V25wKKFer2zqKsE5omLy9sKotF
↪",
  "payload": "WcXpUr4G9yfvVaojNx6K2XZuDYRkFoZQhHrvZB+TKZqsq41g"
}
```

Request JSON Object

- **auth_tag** (*string*) – HMAC calculated with K key as key and UUID as data, using SHA-384 as the underlying hash algorithm
- **encrypted_key** (*string*) – U key encrypted with the agent's public key base64 encoded
- **payload** (*string*) – (optional) payload encrypted with K key base64 encoded.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`GET /v2.3/keys/verify`

Get confirmation of bootstrap key derivation

Example request:

```
GET /v2.2/keys/verify?challenge=1234567890ABCDEFHIJK HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **challenge** – 20 character random string with [a-Z,0-9] as symbols.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "hmac":
↪"719d992fb7d2a0761785fd023fe1cf8a584b835e465e71e2ef2632ff4e9938c080bdefba26194d8ea69dd7f9adee6c18
↪"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **hmac** (*string*) – hmac with K key as key and the challenge

GET /v2.3/quotes/integrity

Get integrity quote from node

Example request:

```
GET /v2.2/quotes/integrity?nonce=1234567890ABCDEFHIJK&mask=0x10401&partial=0 HTTP/1.
↪1
Host: example.com
Accept: application/json
```

Query Parameters

- **nonce** – 20 character random string with [a-Z,0-9] as symbols.
- **mask** – Mask for what PCRs from the TPM are included in the quote.
- **partial** – Is either "0" or "1". If set to "1" the public key is excluded in the response.
- **ima_ml_entry** – (optional) Line offset of the IMA entry list. If not present, 0 is assumed.

Example Response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "quote": "r/
↪1RDR4AYABYABPihP2yz+HcGF0vD0c4qiKt4nvSOAARURVNUAAAAAAAyQ9AAAAAAAAAAAAEgGRAjABY2NgAAAAEABAMAAAEAFC
↪yx6OVUze9jTDvML9xkkK1ghXObCJ5gH+QX0udKfrLacm/
↪iMds28SBtVO0rjqDIoYqGgXhH2ZhwGNDwjRCp6HquvtBe7pGEgtZlxf7Hr3wQRLO3FtliBPBR6gjOo7NC/
↪uGsuPjdPU7c9ls29NgYSqdwShuNdRzwmZrF57umuUgF6GREFlxqLkGcbDIT1itV4zJZtI1caLVxqiH0Qv3sNqlNLsSHggkgc5
↪TsEZOq/
↪leCoLtyVGYghPeGwg0RJfbe8cdyBWCQ6nOA==:AQAAAAQAAwAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

(continues on next page)

(continued from previous page)


```
↪ntmsqy2aDi6NhKnLKz4k4uEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪",
    "hash_alg": "sha256",
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
    "boottime": 123456,
    "ima_measurement_list": "10 367a111b682553da5340f977001689db8366056a ima-ng␣
↪sha256:94c0ac6d0ff747d8f1ca7fac89101a141f3e8f6a2c710717b477a026422766d6 boot_
↪aggregate\n",
    "ima_measurement_list_entry": 0,
    "mb_measurement_list":
↪"AAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACEAAABTcGVjIElEIEV2ZW50MDMAAAAAAAACAAIBAAAACwAgAAAAAAAACAAA
↪[....]"
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **quote** (*string*) – TPM integrity quote
- **hash_alg** (*string*) – Used hash algorithm used in the quote (e.g. sha1, sha256, sha512).
- **enc_alg** (*string*) – Encryption algorithm used in the quote (ecc, rsa).
- **sign_alg** (*string*) – Signing algorthm used in the quote (rsassa, rsapss, ecdsa, ecdaa or ecschnorr).
- **pubkey** (*string*) – PEM encoded public portion of the NK (digest is measured into PCR 16).
- **boottime** (*int*) – Seconds since the system booted
- **ima_measurement_list** (*string*) – (optional) IMA entry list. Is included if *IMA_PCR* (10) is included in the mask
- **ima_measurement_list_entry** (*int*) – (optional) Starting line offset of the IMA entry list returned
- **mb_measurement_list** (*string*) – (optional) UEFI Eventlog list base64 encoded. Is included if PCR 0 is included in the mask

Quote format: The quote field contains the quote, the signature and the PCR values that make up the quote.

```
QUOTE_DATA := rTPM_QUOTE:TPM_SIG:TPM_PCRS
TPM_QUOTE  := base64(TPMS_ATTEST)
TPM_SIG    := base64(TPMT_SIGNATURE)
TPM_PCRS   := base64(tpm2_pcrs) // Can hold more that 8 PCR entries. This is a data␣
↪structure generated by tpm2_quote
```

`GET /v2.3/quotes/identity`

Get identity quote from node

Example request:

```
GET /v2.1/quotes/identity?nonce=1234567890ABCDEFHIJK HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **nonce** – 20 character random string with [a-Z,0-9] as symbols.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "quote": "r/
↪1RDR4AYABYABPihP2yz+HcGF0vD0c4qiKt4nvSOAARURVNUAAAAAAAyQ9AAAAAAAAAAAAEgGRAjABY2NgAAAAEABAMAAAEAF
↪yx6OVUze9jTDvML9xkkK1ghXObCJ5gH+QX0udKfrLacm/
↪iMds28SBtVO0rjqDIoYqGgXhH2ZhwGNDwjRCp6HquvtBe7pGEgtZlxf7Hr3wQRLO3FtliBPBR6gjOo7NC/
↪uGsuPjdPU7c9ls29NgYSqdwShuNdRzwmZrF57umuUgF6GREFlxqLkGcbDIT1itV4zJZtI1caLVxqiH0Qv3sNqlNLsSHggkgc5
↪TsEZOq/
↪leCoLtyVGYghPeGwg0RJfbe8cdyBWCQ6nOA==:AQAAAAQAAwAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪ntmsqy2aDi6NhKnLKz4k4uEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪",
    "hash_alg": "sha256",
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
    "boottime": 123456
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **quote** (*string*) – See *quotes/integrity*
- **hash_alg** (*string*) – See *quotes/integrity*
- **enc_alg** (*string*) – See *quotes/integrity*
- **sign_alg** (*string*) – See *quotes/integrity*
- **pubkey** (*string*) – See *quotes/integrity*
- **boottime** (*int*) – See *quotes/integrity*

5.3.3 Registrar

GET /v2.3/agents/

Get ordered list of registered agents

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "uuids": [
      "5e600bce-a5cb-4f5a-bf08-46d0b45081c5",
      "6dab10e4-6619-4ff9-9062-ee6ad23ec24d",
      "d432fbb3-d2f1-4a97-9ef7-75bd81c00000"
    ]
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **uuids** (*list*) – List of registered agents

GET /v2.3/agents/{agent_id:UUID}

Get EK certificate, AIK and optinal contact ip and port of agent *agent_id*.

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQDjZ4J2HO7ekIONAX/eYIzt7ziiVAqE/
↪1D7I9oEwIE88dIfqH0FQLJAg8u3+ZOgsJDQr9HiMhZRPhv8hRuia8ULdAomyOFA1cVzlBF+xcPUEemOIofbvcBNAoTY/
↪x49r8LpqAEUBBiUeOniQbjfRaV2S5cEAA92wHLQAPLF9Sbf3zNxCnbhtRkEi6C3NYl8/
↪FJqyu5Z9vvwEBBOFFTPasAxMtPm6a+Z5KJ4rDflipfaVcUvTKLIBRI7wkuXqhTR8BeIByK9upQ3iBo+FbYjWSf+BaN+wodMNg
↪",
    "ek_tpm": "AToAAQALAAMAsgAgg3GXZ0SEs/
↪gakMyNRqXXJP1S124GUgtk8qHaGzMUaaoABgCAAEMAEAgAAAAAAAEA0YwlPPIoXryMvbD5cIokN9OkljL2mV1oDxy7ETBXBe1
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQ==",
    "ekcert":
↪"MIIEGTCCAoGgAwIBAgIBBTANBgkqhkiG9w0BAQsFADAYMRYwFAYDVQQDEw1zd3RwbS1sb2NhbGNhMB4XDTIxMDQwOTEyNDAy
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQIDAQABo4HNMIHKMBAGA1UdJQQJMAcGBWeBBQ
↪wRIMEakRDBCMRYwFAYFZ4EFAgEMC2lkOjAwMDAxMDE0MRAwDgYFZ4EFAgIMBXN3dHBtMRYwFAYFZ4EFAgMMC2lkOjIwMTkxMD
```


(continues on next page)

(continued from previous page)


```
↪wQCMAAwIgYDVR0JBBswGTAXBgVngQUCEDEOMAwMAzIuMAIBAAICAKIwHwYDVR0jBBgwFoAUaO+9FEi5yX/
↪GEnU+Vc6b3Si6JeAwDwYDVR0PAQH/BAUDAwcgADANBgkqhkiG9w0BAQsFAAOCAYEAaP/jI2i/
↪hXDrthtaZypQ8VUG5AWFnMDtgiMhDSaKwOBfyxiUiYMTggGYXLOXGIu1SJGBtRJsh3QSYgs2tJCnntWF9Jcpmk6kIW/
↪MC8shE+hdu/
↪gQZKjAPZS4QCLIldv+GVZdNYEIv2FYDsKl6Bq1qUsYhAb7z29Nu1itpdvja2qy7ODJ0u+ThccBuH60VGFclFdJg19dvVQMnfi
↪ZPTLNutJHmF0/Vk9W2pRym8SrUe8G6mwxVW8lP9M7fhovKTzoXVFW3gQWQeUxhvWOncXxtARFLp/
↪+f2mzGBRWxIslW17vpZ3QLlCdJ2C7P3U8x2tvkuyyDfz3/
↪pq+8ECupZhdSvpHlBnWvqs1tAWKW0qI9d0xNYjj3Kfl3Lfy7kqqe6FIkvbDlVhw3vnJlclW+M6D86jBulL9ze+3zyMxy2z8m7
↪",
    "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
    "ip": "127.0.0.1",
    "port": 9002,
    "regcount": 1
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **aik_tpm** (*string*) – base64 encoded AIK. The AIK format is TPM2B_PUBLIC from tpm2-tss.
- **ek_tpm** (*string*) – base64 encoded EK. When a *ekcert* is submitted it will be the public key of that certificate.
- **ekcert** (*string*) – base64 encoded EK certificate. Should be in *DER* format. Gets extracted from NV *0x1c00002*.
- **mtls_cert** (*string*) – Agent HTTPS server certificate. PEM encoded.
- **ip** (*string*) – IPv4 address for contacting the agent. Might be *null*.
- **port** (*integer*) – Port for contacting the agent. Might be *null*.

`POST /v2.3/agents/{agent_id:UUID}`

Add agent *agent_id* to registrar.

Example request:


```
{
  "ekcert":
↪"MIIEGTCCAoGgAwIBAgIBBTANBgkqhkiG9w0BAQsFADAYMRYwFAYDVQQDEw1zd3RwbS1sb2NhbGNhMB4XDTIxMDQwOTEyNDAy
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQIDAQABo4HNMIHKMBAGA1UdJQQJMAcGBWeBB
↪wRIMEakRDBCMRYwFAYFZ4EFAgEMC2lkOjAwMDAxMDE0MRAwDgYFZ4EFAgIMBXN3dHBtMRYwFAYFZ4EFAgMMC2lkOjIwMTkxMD
↪wQCMAAwIgYDVR0JBBswGTAXBgVngQUCEDEOMAwMAzIuMAIBAAICAKIwHwYDVR0jBBgwFoAUaO+9FEi5yX/
↪GEnU+Vc6b3Si6JeAwDwYDVR0PAQH/BAUDAwcgADANBgkqhkiG9w0BAQsFAAOCAYEAaP/jI2i/
↪hXDrthtaZypQ8VUG5AWFnMDtgiMhDSaKwOBfyxiUiYMTggGYXLOXGIu1SJGBtRJsh3QSYgs2tJCnntWF9Jcpmk6kIW/
↪MC8shE+hdu/
```


(continues on next page)

(continued from previous page)


```
↪gQZKjAPZS4QCLIldv+GVZdNYEIv2FYDsKl6Bq1qUsYhAb7z29Nu1itpdvja2qy7ODJ0u+ThccBuH60VGFclFdJg19dvVQMnf
↪ZPTLNutJHmF0/Vk9W2pRym8SrUe8G6mwxVW8lP9M7fhovKTzoXVFW3gQWQeUxhvWOncXxtARFLp/
↪+f2mzGBRWxIslW17vpZ3QLlCdJ2C7P3U8x2tvkuyyDfz3/
↪pq+8ECupZhdSvpHlBnWvqs1tAWKW0qI9d0xNYjj3Kfl3Lfy7kqqe6FIkvbDlVhw3vnJlclW+M6D86jBulL9ze+3zyMxy2z8m7
↪",
  "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQCg5mMzNFqdlUbW8uI/
↪GuMcIIvOXXTohHFTas59JlwrJQVed+5klWP+j7tI7492YPmCnoZvP4T4YdT1PN7tHHGfF81AeMnuw5GV5RkW/
↪QeSD+ssB4f6AfuzYJgBkc28zKmpRRHUbwN4rb/
↪HnJgRXdXsuIcnOqGcC39pD0kiu5TrN6hekjxTQtfAbIlQwwDwHCxKWdtH5x7avd15hqc6cBc2gjTQksXrk+OiMwOFTJ68n0qY
↪mVmd8XhPeYUoMlweXBOwc3e9zM9lZmMvregrFHKYc7CXChz",
  "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
  "ip": "127.0.0.1",
  "port": "9002"
}
```


Request JSON Object

- **ekcert** (*string*) – base64 encoded EK certificate. Should be in *DER* format. Gets extracted from NV *0x1c00002*.
- **aik_tpm** (*string*) – base64 encoded AIK. The AIK format is TPM2B_PUBLIC from tpm2-tss.
- **mtls_cert** (*string*) – Agent HTTPS server certificate. PEM encoded.
- **ip** (*string*) – (Optional) contact IPv4 address for the verifier and tenant to use.
- **port** (*string*) – (Optional) contact port for the verifier and tenant to use.

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "blob": "utzA3gAAAAEARAAgC/
↪w9LP1PKZ9thEk+GkMg4m+tkc9TkavcvFiFL6xbXM2q2fTRyKmQnxuCJc0tQdgsRXMftGiKJyA/
↪SUo8kGNVmcNfAQCs79kl9Ir49JJ8rfyMfDIqOuSVlu9PhxGUOeVzAdxyUmPxq5Qp0s431n/KeL/
↪5nUaVXC+qpOftF4bmVtXwLGTTUbKtyT3GG+9ujkjiwHCQhSKTQ8HiuARgXXh13ntFsJ75PBD5dWauLTuciYZI/
↪WQDVXAcgMnQNxodJUi9ir1GxJWz8zufjVQTVjrlgsgeBdOKbB6+H81K1d9prWhZaVLP+wIwO3YuWgtNHNi90E1z/
↪dah2pzfUpLvJo3lNZ4bJgrJUR507AokGKIFm7EfOf+5WWWAvGxGtgqTJB27vgE0CVBLEuDUHoRcLVBi1Np4GGNTByalxbulg8
↪"
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **blob** (*string*) – base64 encoded blob containing the *aik_tpm* name and a challenge. Is encrypted with *ek_tpm*.

`DELETE /v2.3/agents/{agent_id:UUID}`

Remove agent *agent_id* from registrar.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`PUT /v2.3/agents/{agent_id:UUID}/activate`

Activate physical agent *agent_id*

Example request:

```
{
  "auth_tag":
↪"7087ba88746886262de743587ed97aea6b6e3f32755de5d85415c40feef3169bc58d38855ddb96e32efdd8745d0bdfe
↪"
}
```

Request JSON Object

- **auth_tag** (*string*) – hmac containing the challenge from *blob* and the *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`GET /version`

Get current and supported API versions

Example request:

```
GET /version HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "current_version": "2.3",
    "supported_versions": [
      "1.0",
      "2.0",
      "2.1",
      "2.2",
      "2.3"
    ]
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **current_version** (*string*) – Current API version
- **supported_versions** (*list*) – List of supported API versions

5.4 RESTful API for Keylime (v2.4)

The version API version 2.4 was first implemented in Keylime 7.13.0

5.4.1 Verifier

GET /v2.4/agents/{agent_id:UUID}

Get status of agent *agent_id* from Verifier

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "operational_state": 7,
    "v": "yyNnlWwFRz1ZUzSe2YEpz9A5urtv6oywgttTF7VbBP4=",
    "ip": "127.0.0.1",
    "port": 9002,
    "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
```

(continues on next page)

(continued from previous page)


```
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
    "vtpm_policy": "{\"23\": [\"ffffffffffffffffffffffffffffffffffffffff\", \
↪"0000000000000000000000000000000000000000\"], \"15\": [\
↪"0000000000000000000000000000000000000000\"], \"mask\": \"0x808000\"}",
    "meta_data": "{}",
    "has_mb_refstate": 0,
    "has_runtime_policy": 0,
    "accept_tpm_hash_algs": [
      "sha512",
      "sha384",
      "sha256",
      "sha1"
    ],
    "accept_tpm_encryption_algs": [
      "ecc",
      "rsa"
    ],
    "accept_tpm_signing_algs": [
      "ecschnorr",
      "rsassa"
    ],
    "hash_alg": "sha256",
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "verifier_id": "default",
    "verifier_ip": "127.0.0.1",
    "verifier_port": 8881,
    "severity_level": 6,
    "last_event_id": "qoute_validation.quote_validation",
    "attestation_count": 240,
    "last_received_quote": 1676644582,
    "last_successful_attestation": 1676644462
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **operational_state** (*int*) – Current state of the agent in the CV. Defined in https://github.com/keylime/keylime/blob/master/keylime/common/states.py
- **v** (*string*) – V key for payload base64 encoded or null. Decoded length is 32 bytes
- **ip** (*string*) – Agents contact ip address for the CV

- **port** (*string*) – Agents contact port for the CV
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM
- **vtpm_policy** (*string*) – Static PCR policy and mask for vTPM
- **meta_data** (*string*) – Metadata about the agent. Normally contains certificate information if a CA is used.
- **has_mb_refstate** (*int*) – 1 if a measured boot refstate was provided via tenant, 0 otherwise.
- **has_runtime_policy** (*int*) – 1 if a runtime policy (allowlist and excludelist) was provided via tenant, 0 otherwise.
- **accept_tpm_hash_algs** (*list[string]*) – Accepted TPM hashing algorithms. sha1 must be enabled for IMA validation to work.
- **accept_tpm_encryption_algs** (*list[string]*) – Accepted TPM encryption algorithms.
- **accept_tpm_signing_algs** (*list[string]*) – Accepted TPM signing algorithms.
- **hash_alg** (*string*) – Used hashing algorithm.
- **enc_alg** (*string*) – Used encryption algorithm.
- **sign_alg** (*string*) – Used signing algorithm.
- **verifier_id** (*string*) – Name of the verifier that is used. (Only important if multiple verifiers are used)
- **verifier_ip** (*string*) – IP of the verifier that is used.
- **verifier_port** (*int*) – Port of the verifier that is used.
- **severity_level** (*int*) – Severity level of the agent. Might be *null*. Levels are the numeric representation of the severity labels.
- **last_event_id** (*string*) – ID of the last revocation event. Might be *null*.
- **attestation_count** (*int*) – Number of quotes received from the agent which have verified successfully.
- **last_received_quote** (*int*) – Timestamp of the last quote received from the agent irrespective of validity. A value of 0 indicates no quotes have been received. May be *null* after upgrading from a previous Keylime version.
- **last_successful_attestation** (*int*) – Timestamp of the last quote received from the agent which verified successfully. A value of 0 indicates no valid quotes have been received. May be *null* after upgrading from a previous Keylime version.

POST /v2.4/agents/{agent_id:UUID}

Add new agent *instance_id* to Verifier.

Example request:

```
{
  "v": "3HZMmIEc6yyjfoxdCwcOgPk/6X1GuNG+tlCmNgqBM/I=",
  "cloudagent_ip": "127.0.0.1",
  "cloudagent_port": 9002,
  "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
```

(continues on next page)

(continued from previous page)


```
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
  "ak_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQDjZ4J2HO7ekIONAX/eYIzt7ziiVAqE/
↪1D7I9oEwIE88dIfqH0FQLJAg8u3+ZOgsJDQr9HiMhZRPhv8hRuia8ULdAomyOFA1cVzlBF+xcPUEemOIofbvcBNAoTY/
↪x49r8LpqAEUBBiUeOniQbjfRaV2S5cEAA92wHLQAPLF9Sbf3zNxCnbhtRkEi6C3NYl8/
↪FJqyu5Z9vvwEBBOFFTPasAxMtPm6a+Z5KJ4rDflipfaVcUvTKLIBRI7wkuXqhTR8BeIByK9upQ3iBo+FbYjWSf+BaN+wodMNg
↪",
  "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
  "runtime_policy_name": null,
  "runtime_policy": "",
  "runtime_policy_sig": "",
  "runtime_policy_key": "",
  "mb_refstate": "null",
  "ima_sign_verification_keys": "[]",
  "metadata": "{\"cert_serial\": 719066720466992686663564415155145407427243959000, \
↪"subject\": \"/C=US/ST=MA/L=Lexington/O=Keylime/OU=53/CN=D432FBB3-D2F1-4A97-9EF7-
↪75BD81C00000\"}",
  "revocation_key": "-----BEGIN PRIVATE KEY----- (...) -----END PRIVATE KEY-----\n",
  "accept_tpm_hash_algs": [
    "sha512",
    "sha384",
    "sha256",
    "sha1"
  ],
  "accept_tpm_encryption_algs": [
    "ecc",
    "rsa"
  ],
  "accept_tpm_signing_algs": [
    "ecschnorr",
    "rsassa"
  ],
  "supported_version": "2.0"
}
```


Request JSON Object

- **v** (*string*) – (Optional) V key for payload base64 encoded. Decoded length is 32 bytes.
- **cloudagent_ip** (*string*) – Agents contact ip address for the CV.
- **cloudagent_port** (*string*) – Agents contact port for the CV.
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **ak_tpm** (*string*) – AK of the agent, base64-encoded, same as *aik_tpm* in the registrar.
- **mtls_cert** (*string*) – MTLS certificate of the agent, PEM encoded, same as in the regis-

trar.

- **runtime_policy_name** (*string*) – Optional. If specified with a *runtime_policy* it is saved under that name, if specified without, then the policy with that name is loaded.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.
- **runtime_policy_sig** (*string*) – Optional runtime policy detached signature, base64-encoded. Must also provide *runtime_policy_key*.
- **runtime_policy_key** (*string*) – Optional runtime policy detached signature key, base64-encoded. Must also provide *runtime_policy_sig*.
- **mb_refstate** (*string*) – Measured boot reference state policy.
- **ima_sign_verification_keys** (*string*) – IMA signature verification public keyring JSON object string encoded.
- **metadata** (*string*) – Metadata about the agent. Contains *cert_serial* and *subject* if a CA is used with the tenant.
- **revocation_key** (*string*) – Key which is used to sign the revocation message of the agent.
- **accept_tpm_hash_algs** (*list[string]*) – Accepted TPM hashing algorithms. sha1 must be enabled for IMA validation to work.
- **accept_tpm_encryption_algs** (*list[string]*) – Accepted TPM encryption algorithms.
- **accept_tpm_signing_algs** (*list[string]*) – Accepted TPM signing algorithms.
- **supported_version** (*string*) – supported API version of the agent. *v* prefix must not be included.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`DELETE /v2.4/agents/{agent_id:UUID}`

Terminate instance *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

PUT /v2.4/agents/{agent_id:UUID}/reactivate

Start agent *agent_id* (for an already bootstrapped *agent_id* node)

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

PUT /v2.4/agents/{agent_id:UUID}/stop

Stop Verifier polling on *agent_id*, but don't delete (for an already started *agent_id*). This will make the agent verification fail.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

POST /v2.4/allowlists/{runtime_policy_name:string}

Add new named IMA policy *runtime_policy_name* to Verifier.

Example request:

```
{
  "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
  "runtime_policy": "",
  "runtime_policy_sig": "",
```

(continues on next page)

(continued from previous page)

```
  "runtime_policy_key": ""
}
```

Request JSON Object

- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.
- **runtime_policy_sig** (*string*) – Optional runtime policy detached signature, base64-encoded. Must also provide *runtime_policy_key*.
- **runtime_policy_key** (*string*) – Optional runtime policy detached signature key, base64-encoded. Must also provide *runtime_policy_sig*.

`GET /v2.4/allowlists/[runtime_policy_name:string]`

If *runtime_policy_name* is provided, get the named runtime policies from the Verifier.

Example responses:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "name": "",
    "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001\
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000\
↪"], \"mask\": \"0x408000\"}",
    "runtime_policy": ""
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **name** (*string*) – Name of the requested IMA policy.
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.

Otherwise, retrieve list of names of the runtime policies.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "runtimepolicy names": [
        "runtimepolicyname1",
        "runtimepolicyname2"
    ],
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **names** (*list[string] runtimepolicy*) – List of names of the runtime policies.

DELETE /v2.4/allowlist/{runtime_policy_name:string}

Delete IMA policy *runtime_policy_name*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

GET /v2.4/verify/identity

Verify the identity of a node monitored by keylime

Example request:

```
GET /v2.4/verify/identity?agent_uuid=e1ef9f28-be55-47b0-a6c1-8bef90294b93&hash_
↪alg=sha256&nonce=DGHFH6EQVYGKP7YHNVEAFQQR5TN4W4JA&quote=r/1RDR4AYACIACzy[...]␣
↪HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **agent_uuid** – The UUID of the Agent being verified.
- **hash_alg** – The hash algorithm used by the Keylime agent and TPM.
- **nonce** – The onetime nonce being used for identity verification.

- **quote** – The TPM quoted nonce from the Keylime agent.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "valid": 1
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object

Request JSON Object

- **valid** (*int*) – A boolean 1 for valid, 0 for invalid identity.

`GET /v2.4/mbpolicies/{policy_name:string}`

Get the measured boot policy named *policy_name*

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object

Request JSON Object

- **valid** (*int*) – A boolean 1 for valid, 0 for invalid identity.

`POST /v2.4/verify/evidence`

Verify the evidence against policy. This is useful for 3rd party integrations for things like: * CI/CD pipelines that generate policy * Fleet management systems that manage their own trust but want to check attestation evidence like TPM quotes, IMA logs, Measured Boot against some policy

Example request:

```
POST /v2.4/verify/evidence HTTP/1.1
Host: example.com
Accept: application/json
```

(continues on next page)

(continued from previous page)

```
{
  "nonce": "DGHFH6EQVYGKP7YHNVEAFQQR5TN4W4JA",
  "quote": "r/1RDR4AYACIACzy[...]",
  "hash_alg": "sha256",
  "tpm_ak": "ARgAAQALAAUAcgAAABAAFAALCAAA[...]",
  "tpm_ek": "BABEwIE88dIfqH0FQLJAg8u3+ZOg[...]",
  "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\",[...]",
  "runtime_policy": "{\"meta\": {\"version\": 1, \"timestamp\": "2025-02-24␣
↪21:33:17.574168"}, \"digests\": {\"/boot/System.map-6.2.9-300.fc38.x86_64": [
↪"dc720f9c236[...]",
  "mb_policy": "[...]",
  "ima_measurement_list": "10 0adefe762c149c7cec19da62f0da1297fcfbffff ima-ng␣
↪sha256:0000000000000000000000000000000000000000000000000000000000000000 boot_
↪aggregate[...]",
}
```

Request JSON Object

- **nonce** (*string*) – The onetime nonce being used for identity verification.
- **quote** (*string*) – The quote from the TPM
- **hash_alg** (*string*) – The hashing algorithm used by the TPM
- **tpm_ak** (*string*) – AK of the agent, base64-encoded.
- **tpm_ek** (*string*) – EK of the agent, base64-encoded.
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote. Optional
- **runtime_policy** (*string*) – JSON document of a Keylime Runtime (IMA) policy. Optional
- **mb_policy** (*string*) – JSON document of a Keylime Measured Boot policy. Optional
- **ima_measurement_list** (*string*) – The ASCII content of the IMA measurement list. Usually found at /sys/kernel/security/ima/ascii_runtime_measurements. Optional depending on runtime_policy.
- **mb_list** (*string*) – The binary contents of the boot log. Usually found at something like /sys/kernel/security/tpm0/binary_bios_measurements. Optional depending on mb_policy.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "valid": 0,
    "failures": [
        {
          "type": "ima.validation.ima-ng.not_in_allowlist",
          "context": { "message": "File not found in allowlist: /root/evil.sh" },
        }
```

(continues on next page)

(continued from previous page)

```
    ]
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object

Request JSON Object

- **valid** (*int*) – A boolean 1 for valid, 0 for invalid evidence
- **failures** (*array*) – A list of optional failure objects for the different ways the evidence failed verification
- **type** (*string*) – The Keylime specific type of failure
- **context** (*object*) – More context, such as a human readable message about the failure

5.4.2 Agent

GET /version

Returns what API version the agent supports. This endpoint might not be implemented by all agents.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "supported_version": "2.0"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **supported_version** (*string*) – The latest version the agent supports.

GET /v2.4/agent/info

Retrieves information about an agent

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "agent_uuid": "e1ef9f28-be55-47b0-a6c1-8bef90294b93",
```

(continues on next page)

(continued from previous page)

```
    "tpm_hash_alg": "sha256",
    "tpm_enc_alg": "rsa",
    "tpm_sign_alg": "rsassa",
    "ak_handle": "1078035599"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **agent_uuid** (*string*) – The UUID of the agent.
- **tpm_hash_alg** (*string*) – The hashing algorithm used by this agent's TPM device.
- **tpm_enc_alg** (*string*) – The encryption algorithm used by this agent's TPM device.
- **tpm_sign_alg** (*string*) – The signing algorithm used by this agent's TPM device.
- **ak** (*string*) – The Attestation Key handle of the TPM device used by this agent.

GET /v2.4/keys/pubkey

Retrieves the agent's public key.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **pubkey** (*string*) – Public rsa key of the agent used for encrypting V and U key.

POST /v2.4/keys/vkey

Send *v_key* to node.

Example request:

```
{
  "encrypted_key": "MN/F33jjuLiIuRH8fF7pMtw6Hoe2KG10zg+/xuuZLa5d1WB2aR6feVCwknZDe/
↪dhG51yB0tKau8fCNUz8KMxyWoFkalIY4vVG6DNpLouDjb+vMvI6RmVmCBwO5zx6R802wK2z2yUbcn11TU/
↪k2zHq34CNFIgI5pQu7cnLMzCLW6NLEp8N0IOQL6D+uV9emkheJH1g40xYwUaKoABWjZeaJN5dvKwbkpIf2m+CROmCNPCidh87
```

(continues on next page)

(continued from previous page)

```
↪tZErh1zk+nUamtrgl25pEImw+Cn9RIVTd6fBkmzlGzch5foAqZCyZ0AhQ0ONuWw=="
}
```

Request JSON Object

- **encrypted_key** (*string*) – V key encrypted with the agent's public key base64 encoded.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`POST /v2.4/keys/ukey`

Send *u_key* to node (with optional payload)

Example request:

```
{
  "auth_tag" :
↪"3876c08b30c16c4140ee04300bb4262bbcc9034d8a2ed8c90784f13b484a570bf9da3d5c372141bd16d85de05c4c7cce
↪",
  "encrypted_key":
↪"iAckMZgZc8r43pF0iW8iwwAorD+rvnvF7AShhlz6+am+ryqW+907UynOrWrIrAseyVRE7ouHpr547gnwfF7oKeBFlEdWnE6F
↪y/
↪MmSuNR5pGQwZCueKI0ji3Nqq6heOgSvnMRC0PHgyumOsYiAnbDNyryvfwO4HsqdqMcEsBu1IVzU3EtJWhfQ8i/
↪UpvHy6Jq4bBh+mw5HZwmK93bmsLXNKgjPWAicsCZINUAPVMCUL7dcDd4zijsBxMxiZF7Js7V25wKKFer2zqKsE5omLy9sKotF
↪",
  "payload": "WcXpUr4G9yfvVaojNx6K2XZuDYRkFoZQhHrvZB+TKZqsq41g"
}
```

Request JSON Object

- **auth_tag** (*string*) – HMAC calculated with K key as key and UUID as data, using SHA-384 as the underlying hash algorithm
- **encrypted_key** (*string*) – U key encrypted with the agent's public key base64 encoded
- **payload** (*string*) – (optional) payload encrypted with K key base64 encoded.

Example response:

```
{
  "code": 200,
  "status": "Success",
```

(continues on next page)

(continued from previous page)

```
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

GET /v2.4/keys/verify

Get confirmation of bootstrap key derivation

Example request:

```
GET /v2.4/keys/verify?challenge=1234567890ABCDEFHIJK HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **challenge** – 20 character random string with [a-Z,0-9] as symbols.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "hmac":
↪"719d992fb7d2a0761785fd023fe1cf8a584b835e465e71e2ef2632ff4e9938c080bdefba26194d8ea69dd7f9adee6c18
↪"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **hmac** (*string*) – hmac with K key as key and the challenge

GET /v2.4/quotes/integrity

Get integrity quote from node

Example request:

```
GET /v2.4/quotes/integrity?nonce=1234567890ABCDEFHIJK&mask=0x10401&partial=0 HTTP/1.
↪1
Host: example.com
Accept: application/json
```

Query Parameters

- **nonce** – 20 character random string with [a-Z,0-9] as symbols.
- **mask** – Mask for what PCRs from the TPM are included in the quote.
- **partial** – Is either "0" or "1". If set to "1" the public key is excluded in the response.
- **ima_ml_entry** – (optional) Line offset of the IMA entry list. If not present, 0 is assumed.

Example Response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "quote": "r/
↪1RDR4AYABYABPihP2yz+HcGF0vD0c4qiKt4nvSOAARURVNUAAAAAAAyQ9AAAAAAAAAAAAEgGRAjABY2NgAAAAEABAMAAAEAFC
↪yx6OVUze9jTDvML9xkkK1ghXObCJ5gH+QX0udKfrLacm/
↪iMds28SBtVO0rjqDIoYqGgXhH2ZhwGNDwjRCp6HquvtBe7pGEgtZlxf7Hr3wQRLO3FtliBPBR6gjOo7NC/
↪uGsuPjdPU7c9ls29NgYSqdwShuNdRzwmZrF57umuUgF6GREFlxqLkGcbDIT1itV4zJZtI1caLVxqiH0Qv3sNqlNLsSHggkgc5
↪TsEZOq/
↪leCoLtyVGYghPeGwg0RJfbe8cdyBWCQ6nOA==:AQAAAQAAwAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪ntmsqy2aDi6NhKnLKz4k4uEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪",
    "hash_alg": "sha256",
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
    "boottime": 123456,
    "ima_measurement_list": "10 367a111b682553da5340f977001689db8366056a ima-ng␣
↪sha256:94c0ac6d0ff747d8f1ca7fac89101a141f3e8f6a2c710717b477a026422766d6 boot_
↪aggregate\n",
    "ima_measurement_list_entry": 0,
    "mb_measurement_list":
↪"AAAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAAAAACEAAABTcGVjIElEIEV2ZW50MDMAAAAAAAACAAIBAAAACwAgAAAAAAACAAA
↪[....]"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **quote** (*string*) – TPM integrity quote
- **hash_alg** (*string*) – Used hash algorithm used in the quote (e.g. sha1, sha256, sha512).
- **enc_alg** (*string*) – Encryption algorithm used in the quote (ecc, rsa).
- **sign_alg** (*string*) – Signing algorthm used in the quote (rsassa, rsapss, ecdsa, ecdaa or ecschnorr).
- **pubkey** (*string*) – PEM encoded public portion of the NK (digest is measured into PCR 16).

- **boottime** (*int*) – Seconds since the system booted
- **ima_measurement_list** (*string*) – (optional) IMA entry list. Is included if *IMA_PCR* (10) is included in the mask
- **ima_measurement_list_entry** (*int*) – (optional) Starting line offset of the IMA entry list returned
- **mb_measurement_list** (*string*) – (optional) UEFI Eventlog list base64 encoded. Is included if PCR 0 is included in the mask

Quote format: The quote field contains the quote, the signature and the PCR values that make up the quote.

```
QUOTE_DATA := rTPM_QUOTE:TPM_SIG:TPM_PCRS
TPM_QUOTE  := base64(TPMS_ATTEST)
TPM_SIG    := base64(TPMT_SIGNATURE)
TPM_PCRS   := base64(tpm2_pcrs) // Can hold more that 8 PCR entries. This is a data␣
↪structure generated by tpm2_quote
```

GET /v2.4/quotes/identity

Get identity quote from node

Example request:

```
GET /v2.4/quotes/identity?nonce=1234567890ABCDEFHIJK HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **nonce** – 20 character random string with [a-Z,0-9] as symbols.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "quote": "r/
↪1RDR4AYABYABPihP2yz+HcGF0vD0c4qiKt4nvSOAARURVNUAAAAAAAyQ9AAAAAAAAAAAAEgGRAjABY2NgAAAAEABAMAAAEAFC
↪yx6OVUze9jTDvML9xkkK1ghXObCJ5gH+QX0udKfrLacm/
↪iMds28SBtVO0rjqDIoYqGgXhH2ZhwGNDwjRCp6HquvtBe7pGEgtZlxf7Hr3wQRLO3FtliBPBR6gjOo7NC/
↪uGsuPjdPU7c9ls29NgYSqdwShuNdRzwmZrF57umuUgF6GREFlxqLkGcbDIT1itV4zJZtI1caLVxqiH0Qv3sNqlNLsSHggkgc5
↪TsEZOq/
↪leCoLtyVGYghPeGwg0RJfbe8cdyBWCQ6nOA==:AQAAAAQAAwAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪ntmsqy2aDi6NhKnLKz4k4uEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪",
    "hash_alg": "sha256",
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
    "boottime": 123456
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **quote** (*string*) – See *quotes/integrity*
- **hash_alg** (*string*) – See *quotes/integrity*
- **enc_alg** (*string*) – See *quotes/integrity*
- **sign_alg** (*string*) – See *quotes/integrity*
- **pubkey** (*string*) – See *quotes/integrity*
- **boottime** (*int*) – See *quotes/integrity*

5.4.3 Registrar

`GET /v2.4/agents/`

Get ordered list of registered agents

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "uuids": [
      "5e600bce-a5cb-4f5a-bf08-46d0b45081c5",
      "6dab10e4-6619-4ff9-9062-ee6ad23ec24d",
      "d432fbb3-d2f1-4a97-9ef7-75bd81c00000"
    ]
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **uuids** (*list*) – List of registered agents

`GET /v2.4/agents/{agent_id:UUID}`

Get EK certificate, AIK and optinal contact ip and port of agent *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQDjZ4J2HO7ekIONAX/eYIzt7ziiVAqE/
↪1D7I9oEwIE88dIfqH0FQLJAg8u3+ZOgsJDQr9HiMhZRPhv8hRuia8ULdAomyOFA1cVzlBF+xcPUEemOIofbvcBNAoTY/
↪x49r8LpqAEUBBiUeOniQbjfRaV2S5cEAA92wHLQAPLF9Sbf3zNxCnbhtRkEi6C3NYl8/
↪FJqyu5Z9vvwEBBOFFTPasAxMtPm6a+Z5KJ4rDflipfaVcUvTKLIBRI7wkuXqhTR8BeIByK9upQ3iBo+FbYjWSf+BaN+wodMNg
```

(continues on next page)

(continued from previous page)


```
↪",
    "ek_tpm": "AToAAQALAAMAsgAgg3GXZ0SEs/
↪gakMyNRqXXJP1S124GUgtk8qHaGzMUaaoABgCAAEMAEAgAAAAAAAEA0YwlPPIoXryMvbD5cIokN9OkljL2mV1oDxy7ETBXBe1
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQ==",
    "ekcert":
↪"MIIEGTCCAoGgAwIBAgIBBTANBgkqhkiG9w0BAQsFADAYMRYwFAYDVQQDEw1zd3RwbS1sb2NhbGNhMB4XDTIxMDQwOTEyNDAy
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQIDAQABo4HNMIHKMBAGA1UdJQQJMAcGBWeBB
↪wRIMEakRDBCMRYwFAYFZ4EFAgEMC2lkOjAwMDAxMDE0MRAwDgYFZ4EFAgIMBXN3dHBtMRYwFAYFZ4EFAgMMC2lkOjIwMTkxMD
↪wQCMAAwIgYDVR0JBBswGTAXBgVngQUCEDEOMAwMAzIuMAIBAAICAKIwHwYDVR0jBBgwFoAUaO+9FEi5yX/
↪GEnU+Vc6b3Si6JeAwDwYDVR0PAQH/BAUDAwcgADANBgkqhkiG9w0BAQsFAAOCAYEAaP/jI2i/
↪hXDrthtaZypQ8VUG5AWFnMDtgiMhDSaKwOBfyxiUiYMTggGYXLOXGIu1SJGBtRJsh3QSYgs2tJCnntWF9Jcpmk6kIW/
↪MC8shE+hdu/
↪gQZKjAPZS4QCLIldv+GVZdNYEIv2FYDsKl6Bq1qUsYhAb7z29Nu1itpdvja2qy7ODJ0u+ThccBuH60VGFclFdJg19dvVQMnf
↪ZPTLNutJHmF0/Vk9W2pRym8SrUe8G6mwxVW8lP9M7fhovKTzoXVFW3gQWQeUxhvWOncXxtARFLp/
↪+f2mzGBRWxIslW17vpZ3QLlCdJ2C7P3U8x2tvkuyyDfz3/
↪pq+8ECupZhdSvpHlBnWvqs1tAWKW0qI9d0xNYjj3Kfl3Lfy7kqqe6FIkvbDlVhw3vnJlclW+M6D86jBulL9ze+3zyMxy2z8m7
↪",
    "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
    "ip": "127.0.0.1",
    "port": 9002,
    "regcount": 1
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **aik_tpm** (*string*) – base64 encoded AIK. The AIK format is TPM2B_PUBLIC from tpm2-tss.
- **ek_tpm** (*string*) – base64 encoded EK. When a *ekcert* is submitted it will be the public key of that certificate.
- **ekcert** (*string*) – base64 encoded EK certificate. Should be in *DER* format. Gets extracted from NV *0x1c00002*.
- **mtls_cert** (*string*) – Agent HTTPS server certificate. PEM encoded.
- **ip** (*string*) – IPv4 address for contacting the agent. Might be *null*.
- **port** (*integer*) – Port for contacting the agent. Might be *null*.

`POST /v2.4/agents/{agent_id:UUID}`

Add agent *agent_id* to registrar.

Example request:

```
{
  "ekcert":
↪"MIIEGTCCAoGgAwIBAgIBBTANBgkqhkiG9w0BAQsFADAYMRYwFAYDVQQDEw1zd3RwbS1sb2NhbGNhMB4XDTIxMDQwOTEyNDAy
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQIDAQABo4HNMIHKMBAGA1UdJQQJMAcGBWeBB
↪wRIMEakRDBCMRYwFAYFZ4EFAgEMC2lkOjAwMDAxMDE0MRAwDgYFZ4EFAgIMBXN3dHBtMRYwFAYFZ4EFAgMMC2lkOjIwMTkxMD
↪wQCMAAwIgYDVR0JBBswGTAXBgVngQUCEDEOMAwMAzIuMAIBAAICAKIwHwYDVR0jBBgwFoAUaO+9FEi5yX/
↪GEnU+Vc6b3Si6JeAwDwYDVR0PAQH/BAUDAwcgADANBgkqhkiG9w0BAQsFAAOCAYEAaP/jI2i/
↪hXDrthtaZypQ8VUG5AWFnMDtgiMhDSaKwOBfyxiUiYMTggGYXLOXGIu1SJGBtRJsh3QSYgs2tJCnntWF9Jcpmk6kIW/
↪MC8shE+hdu/
↪gQZKjAPZS4QCLIldv+GVZdNYEIv2FYDsKl6Bq1qUsYhAb7z29Nu1itpdvja2qy7ODJ0u+ThccBuH60VGFclFdJg19dvVQMnf
↪ZPTLNutJHmF0/Vk9W2pRym8SrUe8G6mwxVW8lP9M7fhovKTzoXVFW3gQWQeUxhvWOncXxtARFLp/
↪+f2mzGBRWxIslW17vpZ3QLlCdJ2C7P3U8x2tvkuyyDfz3/
↪pq+8ECupZhdSvpHlBnWvqs1tAWKW0qI9d0xNYjj3Kfl3Lfy7kqqe6FIkvbDlVhw3vnJlclW+M6D86jBulL9ze+3zyMxy2z8m7
↪",
  "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQCg5mMzNFqdlUbW8uI/
↪GuMcIIvOXXTohHFTas59JlwrJQVed+5klWP+j7tI7492YPmCnoZvP4T4YdT1PN7tHHGfF81AeMnuw5GV5RkW/
↪QeSD+ssB4f6AfuzYJgBkc28zKmpRRHUbwN4rb/
↪HnJgRXdXsuIcnOqGcC39pD0kiu5TrN6hekjxTQtfAbIlQwwDwHCxKWdtH5x7avd15hqc6cBc2gjTQksXrk+OiMwOFTJ68n0qY
↪mVmd8XhPeYUoMlweXBOwc3e9zM9lZmMvregrFHKYc7CXChz",
  "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
  "ip": "127.0.0.1",
  "port": "9002"
}
```


Request JSON Object

- **ekcert** (*string*) – base64 encoded EK certificate. Should be in *DER* format. Gets extracted from NV *0x1c00002*.
- **aik_tpm** (*string*) – base64 encoded AIK. The AIK format is TPM2B_PUBLIC from tpm2-tss.
- **mtls_cert** (*string*) – Agent HTTPS server certificate. PEM encoded.
- **ip** (*string*) – (Optional) contact IPv4 address for the verifier and tenant to use.
- **port** (*string*) – (Optional) contact port for the verifier and tenant to use.

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "blob": "utzA3gAAAAEARAAgC/
↪w9LP1PKZ9thEk+GkMg4m+tkc9TkavcvFiFL6xbXM2q2fTRyKmQnxuCJc0tQdgsRXMftGiKJyA/
↪SUo8kGNVmcNfAQCs79kl9Ir49JJ8rfyMfDIqOuSVlu9PhxGUOeVzAdxyUmPxq5Qp0s431n/KeL/
↪5nUaVXC+qpOftF4bmVtXwLGTTUbKtyT3GG+9ujkjiwHCQhSKTQ8HiuARgXXh13ntFsJ75PBD5dWauLTuciYZI/
↪WQDVXAcgMnQNxodJUi9ir1GxJWz8zufjVQTVjrlgsgeBdOKbB6+H81K1d9prWhZaVLP+wIwO3YuWgtNHNi90E1z/
↪dah2pzfUpLvJo3lNZ4bJgrJUR507AokGKIFm7EfOf+5WWWAvGxGtgqTJB27vgE0CVBLEuDUHoRcLVBi1Np4GGNTByalxbulg8
↪"
```


(continues on next page)

(continued from previous page)

```
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **blob** (*string*) – base64 encoded blob containing the *aik_tpm* name and a challenge. Is encrypted with *ek_tpm*.

`DELETE /v2.4/agents/{agent_id:UUID}`

Remove agent *agent_id* from registrar.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`PUT /v2.4/agents/{agent_id:UUID}/activate`

Activate physical agent *agent_id*

Example request:

```
{
  "auth_tag":
↪"7087ba88746886262de743587ed97aea6b6e3f32755de5d85415c40feef3169bc58d38855ddb96e32efdd8745d0bdfe
↪"
}
```

Request JSON Object

- **auth_tag** (*string*) – hmac containing the challenge from *blob* and the *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`GET /version`

Get current and supported API versions

Example request:

```
GET /version HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "current_version": "2.3",
    "supported_versions": [
      "1.0",
      "2.0",
      "2.1",
      "2.2",
      "2.3"
    ]
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **current_version** (*string*) – Current API version
- **supported_versions** (*list*) – List of supported API versions

5.5 RESTful API for Keylime (v2.5)

The version API version 2.5 was first implemented in Keylime 7.14.0

5.5.1 Verifier

`GET /v2.5/agents/{agent_id:UUID}`

Get status of agent *agent_id* from Verifier

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "operational_state": 7,
    "v": "yyNnlWwFRz1ZUzSe2YEpz9A5urtv6oywgttTF7VbBP4=",
    "ip": "127.0.0.1",
    "port": 9002,
    "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
    "vtpm_policy": "{\"23\": [\"ffffffffffffffffffffffffffffffffffffffff\", \
↪"0000000000000000000000000000000000000000\"], \"15\": [\
↪"0000000000000000000000000000000000000000\"], \"mask\": \"0x808000\"}",
    "meta_data": "{}",
    "has_mb_refstate": 0,
    "has_runtime_policy": 0,
    "accept_tpm_hash_algs": [
      "sha512",
      "sha384",
      "sha256",
      "sha1"
    ],
    "accept_tpm_encryption_algs": [
      "ecc",
      "rsa"
    ],
    "accept_tpm_signing_algs": [
      "ecschnorr",
      "rsassa"
    ],
    "hash_alg": "sha256",
    "enc_alg": "rsa2048",
    "sign_alg": "rsassa",
    "verifier_id": "default",
    "verifier_ip": "127.0.0.1",
    "verifier_port": 8881,
    "severity_level": 6,
    "last_event_id": "qoute_validation.quote_validation",
    "attestation_count": 240,
    "last_received_quote": 1676644582,
    "last_successful_attestation": 1676644462,
    "attestation_status": "PASS",
    "attestation_period": "2s",
    "maximum_attestation_interval": "10s"
  }
```


(continues on next page)

(continued from previous page)

```
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **operational_state** (*int*) – Current state of the agent in the CV. Defined in https://github.com/keylime/keylime/blob/master/keylime/common/states.py
- **v** (*string*) – V key for payload base64 encoded or null. Decoded length is 32 bytes
- **ip** (*string*) – Agents contact ip address for the CV
- **port** (*string*) – Agents contact port for the CV
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM
- **vtpm_policy** (*string*) – Static PCR policy and mask for vTPM
- **meta_data** (*string*) – Metadata about the agent. Normally contains certificate information if a CA is used.
- **has_mb_refstate** (*int*) – 1 if a measured boot refstate was provided via tenant, 0 otherwise.
- **has_runtime_policy** (*int*) – 1 if a runtime policy (allowlist and excludelist) was provided via tenant, 0 otherwise.
- **accept_tpm_hash_algs** (*list[string]*) – Accepted TPM hashing algorithms. sha1 must be enabled for IMA validation to work.
- **accept_tpm_encryption_algs** (*list[string]*) – Accepted TPM encryption algorithms.
- **accept_tpm_signing_algs** (*list[string]*) – Accepted TPM signing algorithms.
- **hash_alg** (*string*) – Used hashing algorithm.
- **enc_alg** (*string*) – Used encryption algorithm. In API v2.5+, this includes explicit bit-length (e.g., rsa2048, ecc256). Generic formats (rsa, ecc) are normalized to their defaults.
- **sign_alg** (*string*) – Used signing algorithm.
- **verifier_id** (*string*) – Name of the verifier that is used. (Only important if multiple verifiers are used)
- **verifier_ip** (*string*) – IP of the verifier that is used.
- **verifier_port** (*int*) – Port of the verifier that is used.
- **severity_level** (*int*) – Severity level of the agent. Might be *null*. Levels are the numeric representation of the severity labels.
- **last_event_id** (*string*) – ID of the last revocation event. Might be *null*.
- **attestation_count** (*int*) – Number of quotes received from the agent which have verified successfully.
- **last_received_quote** (*int*) – Timestamp of the last quote received from the agent irrespective of validity. A value of 0 indicates no quotes have been received. May be *null* after upgrading from a previous Keylime version.

- **last_successful_attestation** (*int*) – Timestamp of the last quote received from the agent which verified successfully. A value of 0 indicates no valid quotes have been received. May be *null* after upgrading from a previous Keylime version.
- **attestation_status** (*string*) – Current attestation status of the agent. Possible values: "PASS" (agent is successfully attesting), "FAIL" (agent attestation failed or timed out), "PENDING" (agent enrolled but not yet attested).
- **attestation_period** (*string*) – Configured attestation interval (e.g., "2s"). Derived from the `quote_interval` verifier configuration option.
- **maximum_attestation_interval** (*string*) – Maximum time allowed between attestations before the agent is considered timed out in PUSH mode (e.g., "10s"). Typically `quote_interval * 5` seconds.

POST /v2.5/agents/{agent_id:UUID}

Add new agent *instance_id* to Verifier.

Example request:


```
{
  "v": "3HZMmIEc6yyjfoxdCwcOgPk/6X1GuNG+tlCmNgqBM/I=",
  "cloudagent_ip": "127.0.0.1",
  "cloudagent_port": 9002,
  "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
  "ak_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQDjZ4J2HO7ekIONAX/eYIzt7ziiVAqE/
↪1D7I9oEwIE88dIfqH0FQLJAg8u3+ZOgsJDQr9HiMhZRPhv8hRuia8ULdAomyOFA1cVzlBF+xcPUEemOIofbvcBNAoTY/
↪x49r8LpqAEUBBiUeOniQbjfRaV2S5cEAA92wHLQAPLF9Sbf3zNxCnbhtRkEi6C3NYl8/
↪FJqyu5Z9vvwEBBOFFTPasAxMtPm6a+Z5KJ4rDflipfaVcUvTKLIBRI7wkuXqhTR8BeIByK9upQ3iBo+FbYjWSf+BaN+wodMNg
↪",
  "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
  "runtime_policy_name": null,
  "runtime_policy": "",
  "runtime_policy_sig": "",
  "runtime_policy_key": "",
  "mb_refstate": "null",
  "ima_sign_verification_keys": "[]",
  "metadata": "{\"cert_serial\": 71906672046699268666356441515514540742724395900, \
↪"subject\": \"/C=US/ST=MA/L=Lexington/O=Keylime/OU=53/CN=D432FBB3-D2F1-4A97-9EF7-
↪75BD81C00000\"}",
  "revocation_key": "-----BEGIN PRIVATE KEY----- (...) -----END PRIVATE KEY-----\n",
  "accept_tpm_hash_algs": [
    "sha512",
    "sha384",
    "sha256",
    "sha1"
```


(continues on next page)

(continued from previous page)

```
  ],
  "accept_tpm_encryption_algs": [
    "ecc",
    "rsa"
  ],
  "accept_tpm_signing_algs": [
    "ecschnorr",
    "rsassa"
  ],
  "supported_version": "2.0"
}
```

Request JSON Object

- **v** (*string*) – (Optional) V key for payload base64 encoded. Decoded length is 32 bytes.
- **cloudagent_ip** (*string*) – Agents contact ip address for the CV.
- **cloudagent_port** (*string*) – Agents contact port for the CV.
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **ak_tpm** (*string*) – AK of the agent, base64-encoded, same as *aik_tpm* in the registrar.
- **mtls_cert** (*string*) – MTLS certificate of the agent, PEM encoded, same as in the registrar.
- **runtime_policy_name** (*string*) – Optional. If specified with a *runtime_policy* it is saved under that name, if specified without, then the policy with that name is loaded.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.
- **runtime_policy_sig** (*string*) – Optional runtime policy detached signature, base64-encoded. Must also provide *runtime_policy_key*.
- **runtime_policy_key** (*string*) – Optional runtime policy detached signature key, base64-encoded. Must also provide *runtime_policy_sig*.
- **mb_refstate** (*string*) – Measured boot reference state policy.
- **ima_sign_verification_keys** (*string*) – IMA signature verification public keyring JSON object string encoded.
- **metadata** (*string*) – Metadata about the agent. Contains *cert_serial* and *subject* if a CA is used with the tenant.
- **revocation_key** (*string*) – Key which is used to sign the revocation message of the agent.
- **accept_tpm_hash_algs** (*list[string]*) – Accepted TPM hashing algorithms. sha1 must be enabled for IMA validation to work.
- **accept_tpm_encryption_algs** (*list[string]*) – Accepted TPM encryption algorithms.
- **accept_tpm_signing_algs** (*list[string]*) – Accepted TPM signing algorithms.
- **supported_version** (*string*) – supported API version of the agent. *v* prefix must not be included.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`DELETE /v2.5/agents/{agent_id:UUID}`

Terminate instance *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

`PUT /v2.5/agents/{agent_id:UUID}/reactivate`

Start agent *agent_id* (for an already bootstrapped *agent_id* node)

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`PUT /v2.5/agents/{agent_id:UUID}/stop`

Stop Verifier polling on *agent_id*, but don't delete (for an already started *agent_id*). This will make the agent verification fail.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`POST /v2.5/allowlists/{runtime_policy_name:string}`

Add new named IMA policy *runtime_policy_name* to Verifier.

Example request:

```
{
  "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
  "runtime_policy": "",
  "runtime_policy_sig": "",
  "runtime_policy_key": ""
}
```

Request JSON Object

- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.
- **runtime_policy_sig** (*string*) – Optional runtime policy detached signature, base64-encoded. Must also provide *runtime_policy_key*.
- **runtime_policy_key** (*string*) – Optional runtime policy detached signature key, base64-encoded. Must also provide *runtime_policy_sig*.

`GET /v2.5/allowlists/[runtime_policy_name:string]`

If *runtime_policy_name* is provided, get the named runtime policies from the Verifier.

Example responses:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "name": "",
    "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
↪", \"ffffffffffffffffffffffffffffffffffffffff\", \
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff\", \
```

(continues on next page)

(continued from previous page)

```
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪"], \"15\": [\"0000000000000000000000000000000000000000\", \
↪"0000000000000000000000000000000000000000000000000000000000000000\", \
↪"00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪"], \"mask\": \"0x408000\"}",
    "runtime_policy": ""
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **name** (*string*) – Name of the requested IMA policy.
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote.
- **runtime_policy** (*string*) – Runtime policy JSON object, base64 encoded.

Otherwise, retrieve list of names of the runtime policies.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "runtimepolicy names": [
        "runtimepolicyname1",
        "runtimepolicyname2"
    ],
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **names** (*list[string] runtimepolicy*) – List of names of the runtime policies.

`DELETE /v2.5/allowlist/{runtime_policy_name:string}`

Delete IMA policy *runtime_policy_name*.

Example response:

```
{
  "code": 200,
  "status": "Success",
```

(continues on next page)

(continued from previous page)

```
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`GET /v2.5/verify/identity`

Verify the identity of a node monitored by keylime

Example request:

```
GET /v2.5/verify/identity?agent_uuid=e1ef9f28-be55-47b0-a6c1-8bef90294b93&hash_
↪alg=sha256&nonce=DGHFH6EQVYGKP7YHNVEAFQQR5TN4W4JA&quote=r/1RDR4AYACIACzy[...]␣
↪HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **agent_uuid** – The UUID of the Agent being verified.
- **hash_alg** – The hash algorithm used by the Keylime agent and TPM.
- **nonce** – The onetime nonce being used for identity verification.
- **quote** – The TPM quoted nonce from the Keylime agent.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "valid": 1
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object

Request JSON Object

- **valid** (*int*) – A boolean 1 for valid, 0 for invalid identity.

`GET /v2.5/mbpolicies/{policy_name:string}`

Get the measured boot policy named *policy_name*

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object

Request JSON Object

- **valid** (*int*) – A boolean 1 for valid, 0 for invalid identity.

POST /v2.5/verify/evidence

Verify the evidence against policy. This is useful for 3rd party integrations for things like: * CI/CD pipelines that generate policy * Fleet management systems that manage their own trust but want to check attestation evidence like TPM quotes, IMA logs, Measured Boot against some policy

Example request:

```
POST /v2.5/verify/evidence HTTP/1.1
Host: example.com
Accept: application/json

 {
   "type": "tpm",
   "data": {
     "nonce": "DGHFH6EQVYGKP7YHNVEAFQQR5TN4W4JA",
     "quote": "r/1RDR4AYACIACzy[...]",
     "hash_alg": "sha256",
     "tpm_ak": "ARgAAQALAAUAcgAAABAAFAALCAAA[...]",
     "tpm_ek": "BABEwIE88dIfqH0FQLJAg8u3+ZOg[...]",
     "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\",[...]",
     "runtime_policy": "{\"meta\": {\"version\": 1, \"timestamp\": "2025-02-24␣
↪21:33:17.574168"}, \"digests\": {\"/boot/System.map-6.2.9-300.fc38.x86_64": [
↪"dc720f9c236[...]",
     "mb_policy": "[...]",
     "ima_measurement_list": "10 0adefe762c149c7cec19da62f0da1297fcfbffff ima-ng␣
↪sha256:0000000000000000000000000000000000000000000000000000000000000000 boot_
↪aggregate[...]",
   }
 }
```

Request JSON Object

- **nonce** (*string*) – The onetime nonce being used for identity verification.
- **quote** (*string*) – The quote from the TPM
- **hash_alg** (*string*) – The hashing algorithm used by the TPM

- **tpm_ak** (*string*) – AK of the agent, base64-encoded.
- **tpm_ek** (*string*) – EK of the agent, base64-encoded.
- **tpm_policy** (*string*) – Static PCR policy and mask for TPM. Is a string encoded dictionary that also includes a *mask* for which PCRs should be included in a quote. Optional
- **runtime_policy** (*string*) – JSON document of a Keylime Runtime (IMA) policy. Optional
- **mb_policy** (*string*) – JSON document of a Keylime Measured Boot policy. Optional
- **ima_measurement_list** (*string*) – The ASCII content of the IMA measurement list. Usually found at /sys/kernel/security/ima/ascii_runtime_measurements. Optional depending on runtime_policy.
- **mb_list** (*string*) – The binary contents of the boot log. Usually found at something like /sys/kernel/security/tpm0/binary_bios_measurements. Optional depending on mb_policy.

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "valid": false,
    "failures": [
        {
          "type": "ima.validation.ima-ng.not_in_allowlist",
          "context": { "message": "File not found in allowlist: /root/evil.sh" },
        }
    ],
    "claims": {
        "nonce": "DGHFH6EQVYGKP7YHNVEAFQQR5TN4W4JA",
        "quote": "r/1RDR4AYACIACzy[...]",
        "hash_alg": "sha256",
        "tpm_ak": "ARgAAQALAAUAcgAAABAAFAALCAAA[...]",
        "tpm_ek": "BABEwIE88dIfqH0FQLJAg8u3+ZOg[...]",
        "tpm_policy": "{\"22\": [\"0000000000000000000000000000000000000000000000000000000000000001\", \
↪"0000000000000000000000000000000000000000000000000000000000000001\",[...]",
        "runtime_policy": "{\"meta\": {\"version\": 1, \"timestamp\": "2025-02-24␣
↪21:33:17.574168"}, \"digests\": {\"/boot/System.map-6.2.9-300.fc38.x86_64": [
↪"dc720f9c236[...]",
        "mb_policy": "[...]",
        "ima_measurement_list": "10 0adefe762c149c7cec19da62f0da1297fcfbffff ima-ng␣
↪sha256:0000000000000000000000000000000000000000000000000000000000000000 boot_
↪aggregate[...]",
    }
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object

Request JSON Object

- **valid** (*int*) – A boolean 1 for valid, 0 for invalid evidence
- **failures** (*array*) – A list of optional failure objects for the different ways the evidence failed verification
- **type** (*string*) – The Keylime specific type of failure
- **context** (*object*) – More context, such as a human readable message about the failure
- **claims** (*object*) – The claims that were used for verification. May be empty if verification was unsuccessful.

5.5.2 Agent

GET /version

Returns what API version(s) the agent supports. This endpoint might not be implemented by all agents.

Example response (new format):

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "supported_version": "2.2",
    "supported_versions": ["2.1", "2.2"]
  }
}
```

Example response (legacy format):

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "supported_version": "2.2"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **supported_version** (*string*) – The latest API version the agent supports (for backward compatibility)
- **supported_versions** (*array*) – (optional) Array of all API versions the agent supports. Agents implementing version negotiation include this field. The tenant/verifier will negotiate to use the highest mutually supported version.

GET /v2.5/agent/info

Retrieves information about an agent

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "agent_uuid": "e1ef9f28-be55-47b0-a6c1-8bef90294b93",
    "tpm_hash_alg": "sha256",
    "tpm_enc_alg": "rsa2048",
    "tpm_sign_alg": "rsassa",
    "ak_handle": "1078035599"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **agent_uuid** (*string*) – The UUID of the agent.
- **tpm_hash_alg** (*string*) – The hashing algorithm used by this agent's TPM device.
- **tpm_enc_alg** (*string*) – The encryption algorithm used by this agent's TPM device. In API v2.5+, this includes explicit bit-length (e.g., `rsa2048`, `ecc256`). Generic formats (`rsa`, `ecc`) from earlier API versions are normalized to their default bit-lengths.
- **tpm_sign_alg** (*string*) – The signing algorithm used by this agent's TPM device.
- **ak_handle** (*string*) – The Attestation Key handle of the TPM device used by this agent.

GET /v2.5/keys/pubkey

Retrieves the agent's public key.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **pubkey** (*string*) – Public rsa key of the agent used for encrypting V and U key.

POST /v2.5/keys/vkey

Send *v_key* to node.

Example request:

```
{
  "encrypted_key": "MN/F33jjuLiIuRH8fF7pMtw6Hoe2KG10zg+/xuuZLa5d1WB2aR6feVCwknZDe/
↪dhG51yB0tKau8fCNUz8KMxyWoFkalIY4vVG6DNpLouDjb+vMvI6RmVmCBwO5zx6R802wK2z2yUbcn11TU/
↪k2zHq34CNFIgI5pQu7cnLMzCLW6NLEp8N0IOQL6D+uV9emkheJH1g40xYwUaKoABWjZeaJN5dvKwbkpIf2m+CROmCNPCidh87
↪tZErh1zk+nUamtrgl25pEImw+Cn9RIVTd6fBkmzlGzch5foAqZCyZ0AhQ0ONuWw=="
}
```


Request JSON Object

- **encrypted_key** (*string*) – V key encrypted with the agent's public key base64 encoded.

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`POST /v2.5/keys/ukey`

Send *u_key* to node (with optional payload)

Example request:


```
{
  "auth_tag" :
↪"3876c08b30c16c4140ee04300bb4262bbcc9034d8a2ed8c90784f13b484a570bf9da3d5c372141bd16d85de05c4c7cce
↪",
  "encrypted_key":
↪"iAckMZgZc8r43pF0iW8iwwAorD+rvnvF7AShhlz6+am+ryqW+907UynOrWrIrAseyVRE7ouHpr547gnwfF7oKeBFlEdWnE6F
↪y/
↪MmSuNR5pGQwZCueKI0ji3Nqq6heOgSvnMRC0PHgyumOsYiAnbDNyryvfwO4HsqdqMcEsBu1IVzU3EtJWhfQ8i/
↪UpvHy6Jq4bBh+mw5HZwmK93bmsLXNKgjPWAicsCZINUAPVMCUL7dcDd4zijsBxMxiZF7Js7V25wKKFer2zqKsE5omLy9sKotF
↪",
  "payload": "WcXpUr4G9yfvVaojNx6K2XZuDYRkFoZQhHrvZB+TKZqsq41g"
}
```


Request JSON Object

- **auth_tag** (*string*) – HMAC calculated with K key as key and UUID as data, using SHA-384 as the underlying hash algorithm
- **encrypted_key** (*string*) – U key encrypted with the agent's public key base64 encoded
- **payload** (*string*) – (optional) payload encrypted with K key base64 encoded.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`GET /v2.5/keys/verify`

Get confirmation of bootstrap key derivation

Example request:

```
GET /v2.5/keys/verify?challenge=1234567890ABCDEFHIJK HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **challenge** – 20 character random string with [a-Z,0-9] as symbols.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "hmac":
↪"719d992fb7d2a0761785fd023fe1cf8a584b835e465e71e2ef2632ff4e9938c080bdefba26194d8ea69dd7f9adee6c18
↪"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **hmac** (*string*) – hmac with K key as key and the challenge

`GET /v2.5/quotes/integrity`

Get integrity quote from node

Example request:

```
GET /v2.5/quotes/integrity?nonce=1234567890ABCDEFHIJK&mask=0x10401&partial=0 HTTP/1.
↪1
Host: example.com
Accept: application/json
```

Query Parameters

- **nonce** – 20 character random string with [a-Z,0-9] as symbols.
- **mask** – Mask for what PCRs from the TPM are included in the quote.
- **partial** – Is either "0" or "1". If set to "1" the public key is excluded in the response.
- **ima_ml_entry** – (optional) Line offset of the IMA entry list. If not present, 0 is assumed.

Example Response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "quote": "r/
↪1RDR4AYABYABPihP2yz+HcGF0vD0c4qiKt4nvSOAARURVNUAAAAAAAyQ9AAAAAAAAAAAAEgGRAjABY2NgAAAAEABAMAAAEAF
↪yx6OVUze9jTDvML9xkkK1ghXObCJ5gH+QX0udKfrLacm/
↪iMds28SBtVO0rjqDIoYqGgXhH2ZhwGNDwjRCp6HquvtBe7pGEgtZlxf7Hr3wQRLO3FtliBPBR6gjOo7NC/
↪uGsuPjdPU7c9ls29NgYSqdwShuNdRzwmZrF57umuUgF6GREFlxqLkGcbDIT1itV4zJZtI1caLVxqiH0Qv3sNqlNLsSHggkgc5
↪TsEZOq/
↪leCoLtyVGYghPeGwg0RJfbe8cdyBWCQ6nOA==:AQAAAAQAAwAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪ntmsqy2aDi6NhKnLKz4k4uEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪",
    "hash_alg": "sha256",
    "enc_alg": "rsa2048",
    "sign_alg": "rsassa",
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
    "boottime": 123456,
    "ima_measurement_list": "10 367a111b682553da5340f977001689db8366056a ima-ng␣
↪sha256:94c0ac6d0ff747d8f1ca7fac89101a141f3e8f6a2c710717b477a026422766d6 boot_
↪aggregate\n",
    "ima_measurement_list_entry": 0,
    "mb_measurement_list":
↪"AAAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAAAAACEAAABTcGVjIElEIEV2ZW50MDMAAAAAAAACAAIBAAAACwAgAAAAAAACAAA
↪[....]"
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **quote** (*string*) – TPM integrity quote
- **hash_alg** (*string*) – Hash algorithm used in the quote. Supported values: `sha1`, `sha256`, `sha384`, `sha512`, `sm3_256`.

- **enc_alg** (*string*) – Encryption algorithm used in the quote. In API v2.5+, this includes explicit bit-length specifications. Supported values: `rsa`, `rsa1024`, `rsa2048`, `rsa3072`, `rsa4096`, `ecc`, `ecc192`, `ecc224`, `ecc256`, `ecc384`, `ecc521`, `ecc_sm2`. Generic formats (`rsa`, `ecc`) are normalized to `rsa2048` and `ecc256` respectively.
- **sign_alg** (*string*) – Signing algorithm used in the quote. Supported values: `rsassa`, `rsapss`, `ecdsa`, `ecdaa`, `ecschnorr`.
- **pubkey** (*string*) – PEM encoded public portion of the NK (digest is measured into PCR 16).
- **boottime** (*int*) – Seconds since the system booted
- **ima_measurement_list** (*string*) – (optional) IMA entry list. Is included if *IMA_PCR* (10) is included in the mask
- **ima_measurement_list_entry** (*int*) – (optional) Starting line offset of the IMA entry list returned
- **mb_measurement_list** (*string*) – (optional) UEFI Eventlog list base64 encoded. Is included if PCR 0 is included in the mask

Quote format: The quote field contains the quote, the signature and the PCR values that make up the quote.

```
QUOTE_DATA := rTPM_QUOTE:TPM_SIG:TPM_PCRS
TPM_QUOTE  := base64(TPMS_ATTEST)
TPM_SIG    := base64(TPMT_SIGNATURE)
TPM_PCRS   := base64(tpm2_pcrs) // Can hold more that 8 PCR entries. This is a data␣
↪structure generated by tpm2_quote
```

GET /v2.5/quotes/identity

Get identity quote from node

Example request:

```
GET /v2.5/quotes/identity?nonce=1234567890ABCDEFHIJK HTTP/1.1
Host: example.com
Accept: application/json
```

Query Parameters

- **nonce** – 20 character random string with [a-Z,0-9] as symbols.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "quote": "r/
↪1RDR4AYABYABPihP2yz+HcGF0vD0c4qiKt4nvSOAARURVNUAAAAAAAyQ9AAAAAAAAAAAAEgGRAjABY2NgAAAAEABAMAAAEAFC
↪yx6OVUze9jTDvML9xkkK1ghXObCJ5gH+QX0udKfrLacm/
↪iMds28SBtVO0rjqDIoYqGgXhH2ZhwGNDwjRCp6HquvtBe7pGEgtZlxf7Hr3wQRLO3FtliBPBR6gjOo7NC/
↪uGsuPjdPU7c9ls29NgYSqdwShuNdRzwmZrF57umuUgF6GREFlxqLkGcbDIT1itV4zJZtI1caLVxqiH0Qv3sNqlNLsSHggkgc5
↪TsEZOq/
↪leCoLtyVGYghPeGwg0RJfbe8cdyBWCQ6nOA==:AQAAAAQAAwAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪ntmsqy2aDi6NhKnLKz4k4uEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

(continues on next page)

(continued from previous page)

```
↪",
    "hash_alg": "sha256",
    "enc_alg": "rsa2048",
    "sign_alg": "rsassa",
    "pubkey": "-----BEGIN PUBLIC KEY----- (...) -----END PUBLIC KEY-----\n"
    "boottime": 123456
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **quote** (*string*) – See *quotes/integrity*
- **hash_alg** (*string*) – See *quotes/integrity*
- **enc_alg** (*string*) – See *quotes/integrity*
- **sign_alg** (*string*) – See *quotes/integrity*
- **pubkey** (*string*) – See *quotes/integrity*
- **boottime** (*int*) – See *quotes/integrity*

5.5.3 Registrar

`GET /v2.5/agents/`

Get ordered list of registered agents

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "uuids": [
      "5e600bce-a5cb-4f5a-bf08-46d0b45081c5",
      "6dab10e4-6619-4ff9-9062-ee6ad23ec24d",
      "d432fbb3-d2f1-4a97-9ef7-75bd81c00000"
    ]
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **uuids** (*list*) – List of registered agents

GET /v2.5/agents/{agent_id:UUID}

Get EK certificate, AIK and optinal contact ip and port of agent *agent_id*.

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {
    "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQDjZ4J2HO7ekIONAX/eYIzt7ziiVAqE/
↪1D7I9oEwIE88dIfqH0FQLJAg8u3+ZOgsJDQr9HiMhZRPhv8hRuia8ULdAomyOFA1cVzlBF+xcPUEemOIofbvcBNAoTY/
↪x49r8LpqAEUBBiUeOniQbjfRaV2S5cEAA92wHLQAPLF9Sbf3zNxCnbhtRkEi6C3NYl8/
↪FJqyu5Z9vvwEBBOFFTPasAxMtPm6a+Z5KJ4rDflipfaVcUvTKLIBRI7wkuXqhTR8BeIByK9upQ3iBo+FbYjWSf+BaN+wodMNg
↪",
    "ek_tpm": "AToAAQALAAMAsgAgg3GXZ0SEs/
↪gakMyNRqXXJP1S124GUgtk8qHaGzMUaaoABgCAAEMAEAgAAAAAAAEA0YwlPPIoXryMvbD5cIokN9OkljL2mV1oDxy7ETBXBe1
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQ==",
    "ekcert":
↪"MIIEGTCCAoGgAwIBAgIBBTANBgkqhkiG9w0BAQsFADAYMRYwFAYDVQQDEw1zd3RwbS1sb2NhbGNhMB4XDTIxMDQwOTEyNDAy
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQIDAQABo4HNMIHKMBAGA1UdJQQJMAcGBWeBB
↪wRIMEakRDBCMRYwFAYFZ4EFAgEMC2lkOjAwMDAxMDE0MRAwDgYFZ4EFAgIMBXN3dHBtMRYwFAYFZ4EFAgMMC2lkOjIwMTkxMD
↪wQCMAAwIgYDVR0JBBswGTAXBgVngQUCEDEOMAwMAzIuMAIBAAICAKIwHwYDVR0jBBgwFoAUaO+9FEi5yX/
↪GEnU+Vc6b3Si6JeAwDwYDVR0PAQH/BAUDAwcgADANBgkqhkiG9w0BAQsFAAOCAYEAaP/jI2i/
↪hXDrthtaZypQ8VUG5AWFnMDtgiMhDSaKwOBfyxiUiYMTggGYXLOXGIu1SJGBtRJsh3QSYgs2tJCnntWF9Jcpmk6kIW/
↪MC8shE+hdu/
↪gQZKjAPZS4QCLIldv+GVZdNYEIv2FYDsKl6Bq1qUsYhAb7z29Nu1itpdvja2qy7ODJ0u+ThccBuH60VGFclFdJg19dvVQMnf
↪ZPTLNutJHmF0/Vk9W2pRym8SrUe8G6mwxVW8lP9M7fhovKTzoXVFW3gQWQeUxhvWOncXxtARFLp/
↪+f2mzGBRWxIslW17vpZ3QLlCdJ2C7P3U8x2tvkuyyDfz3/
↪pq+8ECupZhdSvpHlBnWvqs1tAWKW0qI9d0xNYjj3Kfl3Lfy7kqqe6FIkvbDlVhw3vnJlclW+M6D86jBulL9ze+3zyMxy2z8m7
↪",
    "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
    "ip": "127.0.0.1",
    "port": 9002,
    "regcount": 1
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **aik_tpm** (*string*) – base64 encoded AIK. The AIK format is TPM2B_PUBLIC from tpm2-tss.
- **ek_tpm** (*string*) – base64 encoded EK. When a *ekcert* is submitted it will be the public key of that certificate.

- **ekcert** (*string*) – base64 encoded EK certificate. Should be in *DER* format. Gets extracted from NV *0x1c00002*.
- **mtls_cert** (*string*) – Agent HTTPS server certificate. PEM encoded.
- **ip** (*string*) – IPv4 address for contacting the agent. Might be *null*.
- **port** (*integer*) – Port for contacting the agent. Might be *null*.

`POST /v2.5/agents/{agent_id:UUID}`

Add agent *agent_id* to registrar.

Example request:

```
{
  "ekcert":
↪"MIIEGTCCAoGgAwIBAgIBBTANBgkqhkiG9w0BAQsFADAYMRYwFAYDVQQDEw1zd3RwbS1sb2NhbGNhMB4XDTIxMDQwOTEyNDAy
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQIDAQABo4HNMIHKMBAGA1UdJQQJMAcGBWeBB
↪wRIMEakRDBCMRYwFAYFZ4EFAgEMC2lkOjAwMDAxMDE0MRAwDgYFZ4EFAgIMBXN3dHBtMRYwFAYFZ4EFAgMMC2lkOjIwMTkxMD
↪wQCMAAwIgYDVR0JBBswGTAXBgVngQUCEDEOMAwMAzIuMAIBAAICAKIwHwYDVR0jBBgwFoAUaO+9FEi5yX/
↪GEnU+Vc6b3Si6JeAwDwYDVR0PAQH/BAUDAwcgADANBgkqhkiG9w0BAQsFAAOCAYEAaP/jI2i/
↪hXDrthtaZypQ8VUG5AWFnMDtgiMhDSaKwOBfyxiUiYMTggGYXLOXGIu1SJGBtRJsh3QSYgs2tJCnntWF9Jcpmk6kIW/
↪MC8shE+hdu/
↪gQZKjAPZS4QCLIldv+GVZdNYEIv2FYDsKl6Bq1qUsYhAb7z29Nu1itpdvja2qy7ODJ0u+ThccBuH60VGFclFdJg19dvVQMnf
↪ZPTLNutJHmF0/Vk9W2pRym8SrUe8G6mwxVW8lP9M7fhovKTzoXVFW3gQWQeUxhvWOncXxtARFLp/
↪+f2mzGBRWxIslW17vpZ3QLlCdJ2C7P3U8x2tvkuyyDfz3/
↪pq+8ECupZhdSvpHlBnWvqs1tAWKW0qI9d0xNYjj3Kfl3Lfy7kqqe6FIkvbDlVhw3vnJlclW+M6D86jBulL9ze+3zyMxy2z8m7
↪",
  "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQCg5mMzNFqdlUbW8uI/
↪GuMcIIvOXXTohHFTas59JlwrJQVed+5klWP+j7tI7492YPmCnoZvP4T4YdT1PN7tHHGfF81AeMnuw5GV5RkW/
↪QeSD+ssB4f6AfuzYJgBkc28zKmpRRHUbwN4rb/
↪HnJgRXdXsuIcnOqGcC39pD0kiu5TrN6hekjxTQtfAbIlQwwDwHCxKWdtH5x7avd15hqc6cBc2gjTQksXrk+OiMwOFTJ68n0qY
↪mVmd8XhPeYUoMlweXBOwc3e9zM9lZmMvregrFHKYc7CXChz",
  "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
  "ip": "127.0.0.1",
  "port": "9002"
}
```

Request JSON Object

- **ekcert** (*string*) – base64 encoded EK certificate. Should be in *DER* format. Gets extracted from NV *0x1c00002*.
- **aik_tpm** (*string*) – base64 encoded AIK. The AIK format is TPM2B_PUBLIC from tpm2-tss.
- **mtls_cert** (*string*) – Agent HTTPS server certificate. PEM encoded.
- **ip** (*string*) – (Optional) contact IPv4 address for the verifier and tenant to use.
- **port** (*string*) – (Optional) contact port for the verifier and tenant to use.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "blob": "utzA3gAAAAEARAAgC/
↪w9LP1PKZ9thEk+GkMg4m+tkc9TkavcvFiFL6xbXM2q2fTRyKmQnxuCJc0tQdgsRXMftGiKJyA/
↪SUo8kGNVmcNfAQCs79kl9Ir49JJ8rfyMfDIqOuSVlu9PhxGUOeVzAdxyUmPxq5Qp0s431n/KeL/
↪5nUaVXC+qpOftF4bmVtXwLGTTUbKtyT3GG+9ujkjiwHCQhSKTQ8HiuARgXXh13ntFsJ75PBD5dWauLTuciYZI/
↪WQDVXAcgMnQNxodJUi9ir1GxJWz8zufjVQTVjrlgsgeBdOKbB6+H81K1d9prWhZaVLP+wIwO3YuWgtNHNi90E1z/
↪dah2pzfUpLvJo3lNZ4bJgrJUR507AokGKIFm7EfOf+5WWWAvGxGtgqTJB27vgE0CVBLEuDUHoRcLVBi1Np4GGNTByalxbulg8
↪"
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **blob** (*string*) – base64 encoded blob containing the *aik_tpm* name and a challenge. Is encrypted with *ek_tpm*.

`DELETE /v2.5/agents/{agent_id:UUID}`

Remove agent *agent_id* from registrar.

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`PUT /v2.5/agents/{agent_id:UUID}/activate`

Activate physical agent *agent_id*

Example request:


```
{
  "auth_tag":
↪"7087ba88746886262de743587ed97aea6b6e3f32755de5d85415c40feef3169bc58d38855ddb96e32efdd8745d0bdfe
↪"
}
```


Request JSON Object

- **auth_tag** (*string*) – hmac containing the challenge from *blob* and the *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`GET /version`

Get current and supported API versions

Example request:

```
GET /version HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "current_version": "2.3",
    "supported_versions": [
      "1.0",
      "2.0",
      "2.1",
      "2.2",
      "2.3"
    ]
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **current_version** (*string*) – Current API version
- **supported_versions** (*list*) – List of supported API versions

5.6 RESTful API for Keylime (v3.0)

API version 3.0 introduces push-model attestation, where agents initiate connections to the verifier and proactively submit attestation evidence.

Unlike previous API versions where the agent exposed HTTP endpoints for the verifier to poll, in v3.0 the agent acts as a client. The v3.0 endpoints are served by the **verifier only**. The push-model agent does not expose an API.

For a conceptual overview of push-model attestation, see *Push-Model Attestation*.

> **Warning**
>
> Push-model attestation is currently experimental. The API may change in future releases.

5.6.1 Verifier

Push-Model Attestation Endpoints

These endpoints implement the two-phase push-model attestation protocol. Agents use these endpoints to submit attestation capabilities and evidence. Administrators can use the GET endpoints to view attestation results.

For details on authentication requirements, see *Authentication*.

POST /v3/agents/{agent_id}/attestations

Phase 1: Submit attestation capabilities and receive a challenge.

The agent sends its supported evidence types, cryptographic algorithms, and attestation key. The verifier selects parameters and returns a challenge nonce for TPM quote generation.

Parameters

- **agent_id** (*string*) – UUID of the agent

Authentication: PoP bearer token (agent-only)

Example request:

```
{
  "data": {
    "type": "attestation",
    "attributes": {
      "evidence_supported": [
        {
          "evidence_class": "certification",
          "evidence_type": "tpm_quote",
          "capabilities": {
            "signature_schemes": ["rsassa"],
            "hash_algorithms": ["sha256", "sha384", "sha512"],
            "available_subjects": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
↪ 15, 16, 17, 18, 19, 20, 21, 22, 23],
            "certification_keys": [
              {
                "key_class": "asymmetric",
                "key_algorithm": "rsa",
                "key_size": 2048,
                "server_identifier": "ak",
                "allowable_signature_schemes": ["rsassa"],
```

(continues on next page)

(continued from previous page)

```
                "allowable_hash_algorithms": ["sha256", "sha384", "sha512"],
                "public": "<base64-encoded AK public key>"
              }
            ],
            "component_version": "2.0",
            "evidence_version": "1.0"
          }
        },
        {
          "evidence_class": "log",
          "evidence_type": "ima_log",
          "capabilities": {
            "entry_count": 1024,
            "supports_partial_access": true,
            "appendable": true,
            "formats": ["text/plain"],
            "component_version": "1.0",
            "evidence_version": "1.0"
          }
        }
      ],
      "system_info": {
        "boot_time": "2024-01-15T10:30:00Z"
      }
    }
  }
}
```

Example response (201 Created):

```
{
  "data": {
    "type": "attestation",
    "id": "0",
    "attributes": {
      "stage": "awaiting_evidence",
      "evidence_requested": [
        {
          "evidence_class": "certification",
          "evidence_type": "tpm_quote",
          "chosen_parameters": {
            "challenge": "<base64-encoded nonce>",
            "signature_scheme": "rsassa",
            "hash_algorithm": "sha256",
            "selected_subjects": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,␣
↪15, 16, 17, 18, 19, 20, 21, 22, 23],
            "certification_key": {
              "key_class": "asymmetric",
              "key_algorithm": "rsa",
              "key_size": 2048,
              "server_identifier": "ak"
            }
```

(continues on next page)

(continued from previous page)

```
          }
        },
        {
          "evidence_class": "log",
          "evidence_type": "ima_log",
          "chosen_parameters": {
            "starting_offset": 0,
            "entry_count": 1024,
            "format": "text/plain"
          }
        }
      ],
      "system_info": {
        "boot_time": "2024-01-15T10:30:00Z"
      },
      "capabilities_received_at": "2024-01-15T10:30:00.123456Z",
      "challenges_expire_at": "2024-01-15T10:35:00.123456Z"
    },
    "links": {
      "self": "/v3/agents/{agent_id}/attestations/0"
    }
  }
}
```

Request JSON Object

- **data.type** (*string*) – Must be "attestation"
- **data.attributes.evidence_supported** (*array*) – List of evidence types the agent can produce
- **evidence_supported[].evidence_class** (*string*) – "certification" or "log"
- **evidence_supported[].evidence_type** (*string*) – "tpm_quote", "ima_log", or "uefi_log"
- **evidence_supported[].capabilities** (*object*) – Capabilities for this evidence type
- **data.attributes.system_info** (*object*) – System information (e.g. boot time)

Response JSON Object

- **data.id** (*string*) – Attestation index (auto-incremented per agent)
- **data.attributes.stage** (*string*) – "awaiting_evidence"
- **data.attributes.evidence_requested** (*array*) – Evidence the verifier wants the agent to provide
- **evidence_requested[].chosen_parameters.challenge** (*string*) – Base64-encoded challenge nonce for TPM quote
- **data.attributes.capabilities_received_at** (*string*) – ISO 8601 timestamp
- **data.attributes.challenges_expire_at** (*string*) – Deadline for evidence submission
- **data.links.self** (*string*) – URL to this attestation resource

Status Codes

- 201 Created – Attestation created, challenge issued
- 400 Bad Request – Invalid request body
- 403 Forbidden – Attestations disabled for this agent (timeout or previous failure)
- 404 Not Found – Agent not found
- 409 Conflict – Concurrent attestation creation attempt
- 422 Unprocessable Entity – Invalid capabilities data
- 429 Too Many Requests – Rate limited (attestation interval not elapsed). Includes `Retry-After` header
- 503 Service Unavailable – Previous attestation still being verified. Includes `Retry-After` header

PATCH /v3/agents/{agent_id}/attestations/latest

Phase 2: Submit attestation evidence for the latest attestation.

The agent sends the TPM quote, PCR values, and event logs generated using the challenge nonce from Phase 1. The verifier accepts the evidence and verifies it asynchronously.

Parameters

- **agent_id** (*string*) – UUID of the agent

Authentication: PoP bearer token (agent-only)

Example request:

```
{
  "data": {
    "type": "attestation",
    "attributes": {
      "evidence_collected": [
        {
          "evidence_class": "certification",
          "evidence_type": "tpm_quote",
          "data": {
            "subject_data": {
              "0": "<PCR 0 value>",
              "1": "<PCR 1 value>"
            },
            "message": "<base64-encoded TPM quote>",
            "signature": "<base64-encoded quote signature>"
          }
        },
        {
          "evidence_class": "log",
          "evidence_type": "ima_log",
          "data": {
            "entry_count": 512,
            "entries": "<base64-encoded or raw IMA log entries>"
          }
        }
      ]
    }
```

(continues on next page)

(continued from previous page)

```
  }
}
```

Example response (202 Accepted):

```
{
  "data": {
    "type": "attestation",
    "id": "0",
    "attributes": {
      "stage": "evaluating_evidence",
      "evidence": [
        {
          "evidence_class": "certification",
          "evidence_type": "tpm_quote",
          "capabilities": {},
          "chosen_parameters": {},
          "data": {
            "message": "<base64-encoded TPM quote>",
            "signature": "<base64-encoded quote signature>",
            "subject_data": {}
          }
        }
      ],
      "system_info": {
        "boot_time": "2024-01-15T10:30:00Z"
      },
      "capabilities_received_at": "2024-01-15T10:30:00.123456Z",
      "challenges_expire_at": "2024-01-15T10:35:00.123456Z",
      "evidence_received_at": "2024-01-15T10:31:00.123456Z"
    },
    "links": {
      "self": "/v3/agents/{agent_id}/attestations/0"
    }
  },
  "meta": {
    "seconds_to_next_attestation": 45
  }
}
```

Request JSON Object

- **data.type** (*string*) – Must be `"attestation"`
- **data.attributes.evidence_collected** (*array*) – List of evidence items
- **evidence_collected[].evidence_class** (*string*) – `"certification"` or `"log"`
- **evidence_collected[].evidence_type** (*string*) – Type of evidence (must match what was requested)
- **evidence_collected[].data** (*object*) – Evidence data (format depends on evidence type)

Response JSON Object

- **data.attributes.stage** (*string*) – `"evaluating_evidence"` (verification in progress)
- **data.attributes.evidence** (*array*) – Evidence items with capabilities, parameters, and data
- **data.attributes.evidence_received_at** (*string*) – ISO 8601 timestamp when evidence was received
- **meta.seconds_to_next_attestation** (*int*) – Suggested wait before starting the next attestation cycle

Status Codes

- 202 Accepted – Evidence accepted, verification in progress
- 400 Bad Request – Invalid evidence format
- 403 Forbidden – Evidence already submitted, attestation is not the latest, or challenges expired
- 404 Not Found – Agent or attestation not found
- 410 Gone – Attestation no longer exists
- 503 Service Unavailable – No available worker processes. Includes `Retry-After` header

`PATCH /v3/agents/{agent_id}/attestations/{index}`

Submit attestation evidence for a specific attestation by index.

Behaves identically to `PATCH /v3/agents/{agent_id}/attestations/latest` but targets a specific attestation index. Evidence can only be submitted for the latest attestation.

Parameters

- **agent_id** (*string*) – UUID of the agent
- **index** (*integer*) – Attestation index

Authentication: PoP bearer token (agent-only)

Status Codes

- 202 Accepted – Evidence accepted
- 403 Forbidden – Not the latest attestation, evidence already submitted, or challenges expired
- 404 Not Found – Agent or attestation not found

`GET /v3/agents/{agent_id}/attestations`

List all attestations for an agent.

Parameters

- **agent_id** (*string*) – UUID of the agent

Authentication: mTLS (admin) or PoP bearer token (own agent only)

Example response:

```
{
  "data": [
    {
      "type": "attestation",
      "id": "1",
```

(continues on next page)

(continued from previous page)


```
      "attributes": {
        "stage": "verification_complete",
        "evaluation": "pass",
        "evidence": [],
        "system_info": {
          "boot_time": "2024-01-15T10:30:00Z"
        },
        "capabilities_received_at": "2024-01-15T10:30:00.123456Z",
        "challenges_expire_at": "2024-01-15T10:35:00.123456Z",
        "evidence_received_at": "2024-01-15T10:31:00.123456Z",
        "verification_completed_at": "2024-01-15T10:32:00.123456Z"
      },
      "links": {
        "self": "/v3/agents/{agent_id}/attestations/1"
      }
    },
    {
      "type": "attestation",
      "id": "0",
      "attributes": {
        "stage": "verification_complete",
        "evaluation": "pass",
        "evidence": [],
        "system_info": {},
        "capabilities_received_at": "2024-01-15T10:25:00.123456Z",
        "challenges_expire_at": "2024-01-15T10:30:00.123456Z",
        "evidence_received_at": "2024-01-15T10:26:00.123456Z",
        "verification_completed_at": "2024-01-15T10:27:00.123456Z"
      },
      "links": {
        "self": "/v3/agents/{agent_id}/attestations/0"
      }
    }
  ]
}
```


Response JSON Object

- **data** (*array*) – List of attestation resources
- **data[].id** (*string*) – Attestation index
- **data[].attributes.stage** (*string*) – `"awaiting_evidence"`, `"evaluating_evidence"`, or `"verification_complete"`
- **data[].attributes.evaluation** (*string*) – `"pending"`, `"pass"`, or `"fail"`
- **data[].attributes.failure_reason** (*string*) – `"broken_evidence_chain"` or `"policy_violation"` (only when evaluation is `"fail"`)

Status Codes

- 200 OK – Success
- 404 Not Found – Agent not found

`GET /v3/agents/{agent_id}/attestations/latest`

Get the latest attestation for an agent.

Parameters

- **agent_id** (*string*) – UUID of the agent

Authentication: mTLS (admin) or PoP bearer token (own agent only)

Example response:

```
{
  "data": {
    "type": "attestation",
    "id": "1",
    "attributes": {
      "stage": "verification_complete",
      "evaluation": "pass",
      "failure_reason": null,
      "evidence": [
        {
          "evidence_class": "certification",
          "evidence_type": "tpm_quote",
          "capabilities": {},
          "chosen_parameters": {},
          "data": {
            "message": "<base64-encoded TPM quote>",
            "signature": "<base64-encoded signature>",
            "subject_data": {}
          }
        }
      ],
      "system_info": {
        "boot_time": "2024-01-15T10:30:00Z"
      },
      "capabilities_received_at": "2024-01-15T10:30:00.123456Z",
      "challenges_expire_at": "2024-01-15T10:35:00.123456Z",
      "evidence_received_at": "2024-01-15T10:31:00.123456Z",
      "verification_completed_at": "2024-01-15T10:32:00.123456Z"
    },
    "links": {
      "self": "/v3/agents/{agent_id}/attestations/1"
    }
  }
}
```

Response JSON Object

- **data.attributes.stage** (*string*) – Current stage of the attestation
- **data.attributes.evaluation** (*string*) – `"pending"`, `"pass"`, or `"fail"`
- **data.attributes.failure_reason** (*string*) – `null`, `"broken_evidence_chain"`, or `"policy_violation"`
- **data.attributes.evidence** (*array*) – Evidence items with full data

- **data.attributes.capabilities_received_at** (*string*) – When capabilities were received
- **data.attributes.challenges_expire_at** (*string*) – When challenges expire
- **data.attributes.evidence_received_at** (*string*) – When evidence was received (`null` if still awaiting)
- **data.attributes.verification_completed_at** (*string*) – When verification completed (`null` if still in progress)

Status Codes

- 200 OK – Success
- 404 Not Found – Agent not found or no attestations exist

`GET /v3/agents/{agent_id}/attestations/{index}`

Get a specific attestation by index.

Parameters

- **agent_id** (*string*) – UUID of the agent
- **index** (*integer*) – Attestation index

Authentication: mTLS (admin) or PoP bearer token (own agent only)

Response format is identical to `GET /v3/agents/{agent_id}/attestations/latest`.

Status Codes

- 200 OK – Success
- 404 Not Found – Agent or attestation not found

Session Endpoints

These endpoints manage PoP (Proof of Possession) authentication sessions for push-model agents. Sessions are required before an agent can submit attestations.

`POST /v3/sessions`

Create a new authentication session.

The verifier generates a challenge nonce that the agent must sign using its TPM attestation key to prove possession.

Authentication: None (public endpoint)

Example request:

```
{
  "data": {
    "type": "session",
    "attributes": {
      "agent_id": "d432fbb3-d2f1-4a97-9ef7-75bd81c00000",
      "authentication_supported": [
        {
          "authentication_class": "pop",
          "authentication_type": "tpm_pop"
        }
      ]
    }
```

(continues on next page)

(continued from previous page)

```
  }
}
```

Example response (200 OK):

```
{
  "data": {
    "type": "session",
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "attributes": {
      "agent_id": "d432fbb3-d2f1-4a97-9ef7-75bd81c00000",
      "authentication_requested": [
        {
          "authentication_class": "pop",
          "authentication_type": "tpm_pop",
          "chosen_parameters": {
            "challenge": "<base64-encoded nonce>"
          }
        }
      ],
      "created_at": "2024-01-15T10:30:00.123456Z",
      "challenges_expire_at": "2024-01-15T10:31:00.123456Z"
    }
  }
}
```

Request JSON Object

- **data.attributes.agent_id** (*string*) – UUID of the agent requesting a session
- **data.attributes.authentication_supported** (*array*) – Supported authentication methods

Response JSON Object

- **data.id** (*string*) – Session UUID
- **data.attributes.challenges_expire_at** (*string*) – Deadline for submitting the PoP response

Status Codes

- 200 OK – Session created
- 400 Bad Request – Missing or invalid agent_id
- 429 Too Many Requests – Rate limited. Includes `Retry-After` header

`PATCH /v3/sessions/{session_id}`

Submit Proof of Possession response to complete authentication.

The agent signs the challenge nonce from the session creation response using `TPM2_Certify` and submits the result. If valid, the verifier issues a bearer token for subsequent API calls.

Parameters

- **session_id** (*string*) – UUID of the session

Authentication: None (public endpoint; validates PoP internally)

Example request:

```
{
  "data": {
    "type": "session",
    "attributes": {
      "agent_id": "d432fbb3-d2f1-4a97-9ef7-75bd81c00000",
      "authentication_provided": [
        {
          "authentication_class": "pop",
          "authentication_type": "tpm_pop",
          "data": {
            "message": "<base64-encoded AK attest structure>",
            "signature": "<base64-encoded AK signature>"
          }
        }
      ]
    }
  }
}
```

Example response (200 OK, authentication passed):

```
{
  "data": {
    "type": "session",
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "attributes": {
      "agent_id": "d432fbb3-d2f1-4a97-9ef7-75bd81c00000",
      "evaluation": "pass",
      "token": "550e8400-e29b-41d4-a716-446655440000.<secret>",
      "authentication": [
        {
          "authentication_class": "pop",
          "authentication_type": "tpm_pop",
          "chosen_parameters": {
            "challenge": "<base64-encoded nonce>"
          },
          "data": {
            "message": "<base64-encoded AK attest>",
            "signature": "<base64-encoded AK signature>"
          }
        }
      ],
      "created_at": "2024-01-15T10:30:00.123456Z",
      "challenges_expire_at": "2024-01-15T10:31:00.123456Z",
      "response_received_at": "2024-01-15T10:30:30.123456Z",
      "token_expires_at": "2024-01-15T11:30:00.123456Z"
    }
  }
}
```

Response JSON Object

- **data.attributes.evaluation** (*string*) – "pass" or "fail"
- **data.attributes.token** (*string*) – Bearer token for subsequent requests (only on "pass")
- **data.attributes.token_expires_at** (*string*) – Token expiration time (only on "pass")

Status Codes

- 200 OK – PoP response processed (check `evaluation` field for result)
- 400 Bad Request – Missing or invalid request body
- 401 Unauthorized – PoP verification failed
- 404 Not Found – Session not found

Attestation Stages and Evaluations

Each attestation progresses through the following stages:

Stage	Description
`awaiting_evidence`	Capabilities received, challenge issued, waiting for evidence
`evaluating_evidence`	Evidence received, verification in progress
`verification_complete`	Verification finished, see `evaluation` for result

The `evaluation` field indicates the verification result:

Evaluation	Description
`pending`	Verification not yet complete
`pass`	Evidence verified successfully
`fail`	Evidence verification failed (see `failure_reason`)

When an attestation fails, the `failure_reason` field provides the cause:

Failure Reason	Description
`broken_evidence_chain`	TPM quote signature invalid or evidence integrity check failed
`policy_violation`	Evidence is valid but violates the configured attestation policy

5.6.2 Registrar

Note

API version 3.0 uses RFC 9110-compliant HTTP methods. Compared to v2.5:

- Agent registration uses `POST /v3.0/agents/` (collection-level) instead of `POST /v2.5/agents/{agent_id}`. The `agent_id` is now sent in the JSON request body.
- Agent activation uses `POST` instead of `PUT`.

- The legacy backwards-compatibility routes (`PUT /agents/{agent_id}/activate`, `PUT /agents/{agent_id}`, `POST /agents/{agent_id}`) are removed.
- A version root probe endpoint (`GET /v3.0/`) is available to check whether the server supports API version 3.0.

`GET /v3.0/`

Check whether the server supports API version 3.0. Returns `200` if the version is supported.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **`code`** (*int*) – HTTP status code
- **`status`** (*string*) – Status as string
- **`results`** (*object*) – Results as a JSON object (empty)

`GET /v3.0/agents/`

Get ordered list of registered agents

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "uuids": [
      "5e600bce-a5cb-4f5a-bf08-46d0b45081c5",
      "6dab10e4-6619-4ff9-9062-ee6ad23ec24d",
      "d432fbb3-d2f1-4a97-9ef7-75bd81c00000"
    ]
  }
}
```

Response JSON Object

- **`code`** (*int*) – HTTP status code
- **`status`** (*string*) – Status as string
- **`results`** (*object*) – Results as a JSON object
- **`uuids`** (*list*) – List of registered agents

`GET /v3.0/agents/{agent_id:UUID}`

Get EK certificate, AIK and optional contact ip and port of agent *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQDjZ4J2HO7ekIONAX/eYIzt7ziiVAqE/
↪1D7I9oEwIE88dIfqH0FQLJAg8u3+ZOgsJDQr9HiMhZRPhv8hRuia8ULdAomyOFA1cVzlBF+xcPUEemOIofbvcBNAoTY/
↪x49r8LpqAEUBBiUeOniQbjfRaV2S5cEAA92wHLQAPLF9Sbf3zNxCnbhtRkEi6C3NYl8/
↪FJqyu5Z9vvwEBBOFFTPasAxMtPm6a+Z5KJ4rDflipfaVcUvTKLIBRI7wkuXqhTR8BeIByK9upQ3iBo+FbYjWSf+BaN+wodMNg
↪",
    "ek_tpm": "AToAAQALAAMAsgAgg3GXZ0SEs/
↪gakMyNRqXXJP1S124GUgtk8qHaGzMUaaoABgCAAEMAEAgAAAAAAAEA0YwlPPIoXryMvbD5cIokN9OkljL2mV1oDxy7ETBXBe1
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQ==",
    "ekcert":
↪"MIIEGTCCAoGgAwIBAgIBBTANBgkqhkiG9w0BAQsFADAYMRYwFAYDVQQDEw1zd3RwbS1sb2NhbGNhMB4XDTIxMDQwOTEyNDAy
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQIDAQABo4HNMIHKMBAGA1UdJQQJMAcGBWeBB
↪wRIMEakRDBCMRYwFAYFZ4EFAgEMC2lkOjAwMDAxMDE0MRAwDgYFZ4EFAgIMBXN3dHBtMRYwFAYFZ4EFAgMMC2lkOjIwMTkxMD
↪wQCMAAwIgYDVR0JBBswGTAXBgVngQUCEDEOMAwMAzIuMAIBAAICAKIwHwYDVR0jBBgwFoAUaO+9FEi5yX/
↪GEnU+Vc6b3Si6JeAwDwYDVR0PAQH/BAUDAwcgADANBgkqhkiG9w0BAQsFAAOCAYEAaP/jI2i/
↪hXDrthtaZypQ8VUG5AWFnMDtgiMhDSaKwOBfyxiUiYMTggGYXLOXGIu1SJGBtRJsh3QSYgs2tJCnntWF9Jcpmk6kIW/
↪MC8shE+hdu/
↪gQZKjAPZS4QCLIldv+GVZdNYEIv2FYDsKl6Bq1qUsYhAb7z29Nu1itpdvja2qy7ODJ0u+ThccBuH60VGFclFdJg19dvVQMnfi
↪ZPTLNutJHmF0/Vk9W2pRym8SrUe8G6mwxVW8lP9M7fhovKTzoXVFW3gQWQeUxhvWOncXxtARFLp/
↪+f2mzGBRWxIslW17vpZ3QLlCdJ2C7P3U8x2tvkuyyDfz3/
↪pq+8ECupZhdSvpHlBnWvqs1tAWKW0qI9d0xNYjj3Kfl3Lfy7kqqe6FIkvbDlVhw3vnJlclW+M6D86jBulL9ze+3zyMxy2z8m7
↪",
    "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
    "ip": "127.0.0.1",
    "port": 9002,
    "regcount": 1
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **aik_tpm** (*string*) – base64 encoded AIK. The AIK format is TPM2B_PUBLIC from tpm2-tss.
- **ek_tpm** (*string*) – base64 encoded EK. When a *ekcert* is submitted it will be the public key of that certificate.
- **ekcert** (*string*) – base64 encoded EK certificate. Should be in *DER* format. Gets extracted from NV *0x1c00002*.
- **mtls_cert** (*string*) – Agent HTTPS server certificate. PEM encoded.

- **ip** (*string*) – IPv4 address for contacting the agent. Might be *null*.
- **port** (*integer*) – Port for contacting the agent. Might be *null*.

`POST /v3.0/agents/`

Register a new agent with the registrar. The `agent_id` is provided in the JSON request body (not in the URL path).

Example request:

```
{
  "agent_id": "d432fbb3-d2f1-4a97-9ef7-75bd81c00000",
  "ekcert":
↪"MIIEGTCCAoGgAwIBAgIBBTANBgkqhkiG9w0BAQsFADAYMRYwFAYDVQQDEw1zd3RwbS1sb2NhbGNhMB4XDTIxMDQwOTEyNDAy
↪gDAqXryb+F192IJLKShHYSN32LJjCYOKrvNX1lrmr377juICFSRClE4q+pCfzhNj0Izw/
↪eplaAI7gq41vrlnymWYGIEi4McErWG7qwr7LR9CXwiM7nhBYGtvobqoaOm4+f6zo3jQuks/
↪KYjk0BR3mgAec/Qkfefw2lgSSYaPNl/8ytg6Dhla1LK8f7wWy/
↪bv+3z7L11KLr8DZiFAzKBMiIDfaqNGYPhiFLKAMJ0MmJx63obCqx9z5BltV5YQIDAQABo4HNMIHKMBAGA1UdJQQJMAcGBWeBB
↪wRIMEakRDBCMRYwFAYFZ4EFAgEMC2lkOjAwMDAxMDE0MRAwDgYFZ4EFAgIMBXN3dHBtMRYwFAYFZ4EFAgMMC2lkOjIwMTkxMD
↪wQCMAAwIgYDVR0JBBswGTAXBgVngQUCEDEOMAwMAzIuMAIBAAICAKIwHwYDVR0jBBgwFoAUaO+9FEi5yX/
↪GEnU+Vc6b3Si6JeAwDwYDVR0PAQH/BAUDAwcgADANBgkqhkiG9w0BAQsFAAOCAYEAaP/jI2i/
↪hXDrthtaZypQ8VUG5AWFnMDtgiMhDSaKwOBfyxiUiYMTggGYXLOXGIu1SJGBtRJsh3QSYgs2tJCnntWF9Jcpmk6kIW/
↪MC8shE+hdu/
↪gQZKjAPZS4QCLIldv+GVZdNYEIv2FYDsKl6Bq1qUsYhAb7z29Nu1itpdvja2qy7ODJ0u+ThccBuH60VGFclFdJg19dvVQMnfi
↪ZPTLNutJHmF0/Vk9W2pRym8SrUe8G6mwxVW8lP9M7fhovKTzoXVFW3gQWQeUxhvWOncXxtARFLp/
↪+f2mzGBRWxIslW17vpZ3QLlCdJ2C7P3U8x2tvkuyyDfz3/
↪pq+8ECupZhdSvpHlBnWvqs1tAWKW0qI9d0xNYjj3Kfl3Lfy7kqqe6FIkvbDlVhw3vnJlclW+M6D86jBulL9ze+3zyMxy2z8m7
↪",
  "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQCg5mMzNFqdlUbW8uI/
↪GuMcIIvOXXTohHFTas59JlwrJQVed+5klWP+j7tI7492YPmCnoZvP4T4YdT1PN7tHHGfF81AeMnuw5GV5RkW/
↪QeSD+ssB4f6AfuzYJgBkc28zKmpRRHUbwN4rb/
↪HnJgRXdXsuIcnOqGcC39pD0kiu5TrN6hekjxTQtfAbIlQwwDwHCxKWdtH5x7avd15hqc6cBc2gjTQksXrk+OiMwOFTJ68n0qY
↪mVmd8XhPeYUoMlweXBOwc3e9zM9lZmMvregrFHKYc7CXChz",
  "mtls_cert": "-----BEGIN CERTIFICATE----- (...) -----END CERTIFICATE-----",
  "ip": "127.0.0.1",
  "port": "9002"
}
```

Request JSON Object

- **agent_id** (*string*) – UUID of the agent to register.
- **ekcert** (*string*) – base64 encoded EK certificate. Should be in *DER* format. Gets extracted from NV *0x1c00002*.
- **aik_tpm** (*string*) – base64 encoded AIK. The AIK format is TPM2B_PUBLIC from tpm2-tss.
- **mtls_cert** (*string*) – Agent HTTPS server certificate. PEM encoded.
- **ip** (*string*) – (Optional) contact IPv4 address for the verifier and tenant to use.
- **port** (*string*) – (Optional) contact port for the verifier and tenant to use.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "blob": "utzA3gAAAAEARAAgC/
↪w9LP1PKZ9thEk+GkMg4m+tkc9TkavcvFiFL6xbXM2q2fTRyKmQnxuCJc0tQdgsRXMftGiKJyA/
↪SUo8kGNVmcNfAQCs79kl9Ir49JJ8rfyMfDIqOuSVlu9PhxGUOeVzAdxyUmPxq5Qp0s431n/KeL/
↪5nUaVXC+qpOftF4bmVtXwLGTTUbKtyT3GG+9ujkjiwHCQhSKTQ8HiuARgXXh13ntFsJ75PBD5dWauLTuciYZI/
↪WQDVXAcgMnQNxodJUi9ir1GxJWz8zufjVQTVjrlgsgeBdOKbB6+H81K1d9prWhZaVLP+wIwO3YuWgtNHNi90E1z/
↪dah2pzfUpLvJo3lNZ4bJgrJUR507AokGKIFm7EfOf+5WWWAvGxGtgqTJB27vgE0CVBLEuDUHoRcLVBi1Np4GGNTByalxbulg8
↪"
  }
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **blob** (*string*) – base64 encoded blob containing the *aik_tpm* name and a challenge. Is encrypted with *ek_tpm*.

`DELETE /v3.0/agents/{agent_id:UUID}`

Remove agent *agent_id* from registrar.

Example response:


```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```


Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`POST /v3.0/agents/{agent_id:UUID}/activate`

Activate agent *agent_id* after registration.

Example request:


```
{
  "auth_tag":
↪"7087ba88746886262de743587ed97aea6b6e3f32755de5d85415c40feef3169bc58d38855ddb96e32efdd8745d0bdfe
↪"
}
```


Request JSON Object

- **auth_tag** (*string*) – hmac containing the challenge from *blob* and the *agent_id*.

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object (empty)

`GET /version`

Get current and supported API versions

Example request:

```
GET /version HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
{
  "code": 200,
  "status": "Success",
  "results": {
    "current_version": "2.5",
    "supported_versions": [
      "1.0",
      "2.0",
      "2.1",
      "2.2",
      "2.3",
      "2.4",
      "2.5",
      "3.0"
    ]
  }
}
```

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – Status as string
- **results** (*object*) – Results as a JSON object
- **current_version** (*string*) – Current API version
- **supported_versions** (*list*) – List of supported API versions

5.7 Changelog

5.7.1 Changes from v2.5 to v3.0

API version 3.0 introduces push-model attestation. Unlike previous versions where the verifier polls agents, in v3.0 agents initiate connections and submit attestation evidence to the verifier. The v3.0 endpoints are served by the verifier only; the push-model agent does not expose HTTP endpoints.

- **Added *POST /v3/agents/{agent_id}/attestations* endpoint to the verifier:**
  - Allows agents to submit attestation capabilities (Phase 1 of push protocol)
  - Returns challenge nonce for TPM quote generation
- **Added *PATCH /v3/agents/{agent_id}/attestations/latest* endpoint:**
  - Allows agents to submit attestation evidence (Phase 2 of push protocol)
  - Returns *202 Accepted* for asynchronous verification
- **Added *PATCH /v3/agents/{agent_id}/attestations/{index}* endpoint:**
  - Submit evidence for a specific attestation by index
- **Added *GET /v3/agents/{agent_id}/attestations* endpoint:**
  - Lists all attestation records for an agent
- **Added *GET /v3/agents/{agent_id}/attestations/latest* endpoint:**
  - Returns the most recent attestation for an agent, including verification status
- **Added *GET /v3/agents/{agent_id}/attestations/{index}* endpoint:**
  - Returns a specific attestation by its index
- **Added *POST /v3/sessions* endpoint:**
  - Creates a PoP authentication session and returns a challenge nonce for the agent
- **Added *PATCH /v3/sessions/{session_id}* endpoint:**
  - Completes PoP authentication by submitting the TPM-signed challenge response
- Introduced PoP (Proof of Possession) bearer token authentication for agent-to-verifier communication

API version 3.0 also introduces RFC 9110-compliant registrar routes:

- Added `GET /v3.0/` version root probe endpoint to the registrar, allowing clients to check whether the server supports API version 3.0.
- Agent registration now uses `POST /v3.0/agents/` (collection-level endpoint). The `agent_id` is sent in the JSON request body instead of the URL path.
- Agent activation now uses `POST /v3.0/agents/{agent_id}/activate` instead of `PUT`, following correct HTTP method semantics.
- **Removed legacy backwards-compatibility routes from the registrar:**
  - `POST /agents/{agent_id}` (use `POST /agents/` instead)
  - `PUT /agents/{agent_id}/activate` (use `POST /agents/{agent_id}/activate`)
  - `PUT /agents/{agent_id}` (use `POST /agents/{agent_id}/activate`)

5.7.2 Changes from v2.4 to v2.5

API version 2.5 was first implemented in Keylime 7.14.0.

- **Modified *POST /v2.5/verify/evidence* endpoint:**
  - Changed *valid* response field from integer (1/0) to boolean (true/false)
  - Added *claims* field to response containing verified claims
  - Added TEE (Trusted Execution Environment) verification support
- **Modified *GET /v2.5/quotes/integrity* endpoint:**
  - *enc_alg* field in agent responses were modified to return explicit bit-length formats (e.g., `rsa2048`, `rsa3072`, `ecc256`, `ecc384`)
- **Server-side automatic normalization ensures backward compatibility:**
  - `rsa` → `rsa2048`
  - `ecc` → `ecc256`
- **Enhanced *GET /version* endpoint to support API version negotiation:**
  - Added *supported_versions* field containing an array of all API versions the agent supports
  - Retained *supported_version* field for backward compatibility (contains latest version)
  - Tenant and verifier now negotiate to use the highest mutually supported API version instead of blindly using the agent's latest version
  - Prevents compatibility issues when newer agents communicate with older tenants/verifiers
- **Added attestation monitoring fields to *GET /v2.5/agents/{agent_id}* response:**
  - *attestation_status*: Current attestation status (`"PASS"`, `"FAIL"`, or `"PENDING"`)
  - *attestation_period*: Configured attestation interval derived from `quote_interval`
  - *maximum_attestation_interval*: Maximum time allowed between attestations in PUSH mode

5.7.3 Changes from v2.3 to v2.4

API version 2.4 was first implemented in Keylime 7.13.0.

- **Added *POST /v2.4/verify/evidence* experimental endpoint to the verifier:**
  - Allows 3rd party verification of TPM attestation evidence (quotes, IMA logs, measured boot) against policies
  - Returns *valid* field (integer: 1 for valid, 0 for invalid) and *failures* array with validation errors

5.7.4 Changes from v2.2 to v2.3

API version 2.3 was first implemented in Keylime 7.12.0.

- Added *GET /v2.3/mbpolicies/{name}* endpoint to the verifier
- Added *POST /v2.3/mbpolicies/{name}* endpoint to the verifier
- Added *PUT /v2.3/mbpolicies/{name}* endpoint to the verifier
- Added *DELETE /v2.3/mbpolicies/{name}* endpoint to the verifier
- Added *GET /version* endpoint to the registrar

5.7.5 Changes from v2.1 to v2.2

API version 2.2 was first implemented in Keylime 7.11.0.

- Added *GET /v2.2/verify/identity* endpoint to the verifier
- Added *GET /v2.2/agent/info* endpoint to the agent

5.7.6 Changes from v2.0 to v2.1

API version 2.1 was first implemented in Keylime 6.4.0.

- Added *ak_tpm* field to *POST /v2.1/agents/{agent_id:UUID}* in verifier.
- Added *mtls_cert* field to *POST /v2.1/agents/{agent_id:UUID}* in verifier.
- Removed *vmask* parameter from *GET /v2.1/quotes/integrity* in agent

This removed the requirement for the verifier to connect to the registrar.

5.7.7 Changes from v1.0 to v2.0

API version 2.0 was first implemented in Keylime 6.3.0.

- Added mTLS authentication to agent endpoints.
- Added *supported_version* field to *POST /v2.0/agents/{agent_id:UUID}* in verifier.
- Added *mtls_cert* field to *POST/GET /v2.0/agents/{agent_id:UUID}* in registrar.
- Added */version* endpoint to agent. Note that this endpoint is not implemented by all agents.
- Dropped zlib encryption for *quote* field data in *GET /v2.0/quotes/integrity*/*GET /v2.0/quotes/identity*.

RESTful API for Keylime

Keylime API is versioned. More information can be found here: https://github.com/keylime/enhancements/blob/master/45_api_versioning.md

> **Warning**
>
> API version 1.0 will no longer be officially supported starting with Keylime 6.4.0.

5.7.8 General responses

ANY /

Generic fields in responses

Response JSON Object

- **code** (*int*) – HTTP status code
- **status** (*string*) – textual context of that status
- **results** (*object*) – Holds the actual data.

CHAPTER

SIX

KEYLIME DEVELOPMENT

6.1 Contributing

When contributing any keylime repository, please first discuss the change you wish to make via an issue in the relevant repository for your change or ask the community in the `#keylime` channel on the CNCF Slack workspace

6.1.1 Pull Request Process

1. Create an issue outlining the fix or feature.
2. Fork the keylime repository to your own github account and clone it locally.
3. Hack on your changes.
4. Update the README.md or documentation with details of changes to any interface, this includes new environment variables, exposed ports, useful file locations, CLI parameters and configuration values.
5. Add and commit your changes with some descriptive text on the nature of the change / feature in your commit message. Also reference the issue raised at [1] as follows: *Fixes #45*. See the following link for more message types
6. Ensure that CI passes, if it fails, fix the failures.
7. Every pull request requires a review from the core keylime team
8. If your pull request consists of more than one commit, please squash your commits as described in see *Squash Commits*.

Commit Message Guidelines

We follow the commit formatting recommendations found on Chris Beams' How to Write a Git Commit Message article.

Well formed commit messages not only help reviewers understand the nature of the Pull Request, but also assists the release process where commit messages are used to generate release notes.

A good example of a commit message would be as follows:

```
Summarize changes in around 50 characters or less

More detailed explanatory text, if necessary. Wrap it to about 72
characters or so. In some contexts, the first line is treated as the
subject of the commit and the rest of the text as the body. The
blank line separating the summary from the body is critical (unless
you omit the body entirely); various tools like `log`, `shortlog`
and `rebase` can get confused if you run the two together.
```

(continues on next page)

(continued from previous page)

```
Explain the problem that this commit is solving. Focus on why you
are making this change as opposed to how (the code explains that).
Are there side effects or other unintuitive consequences of this
change? Here's the place to explain them.

Further paragraphs come after blank lines.

- Bullet points are okay, too

- Typically a hyphen or asterisk is used for the bullet, preceded
  by a single space, with blank lines in between, but conventions
  vary here

If you use an issue tracker, put references to them at the bottom,
like this:

Resolves: #123
See also: #456, #789
```

Note the *Resolves #123* tag, this references the issue raised and allows us to ensure issues are associated and closed when a pull request is merged.

Please refer to the github help page on message types for a complete list of issue references.

Squash Commits

Should your pull request consist of more than one commit (perhaps due to a change being requested during the review cycle), please perform a git squash once a reviewer has approved your pull request.

A squash can be performed as follows. Let's say you have the following commits:

```
initial commit
second commit
final commit
```

Run the command below with the number set to the total commits you wish to squash (in our case 3 commits):

```
git rebase -i HEAD~3
```

You default text editor will then open up and you will see the following:

```
pick eb36612 initial commit
pick 9ac8968 second commit
pick a760569 final commit

# Rebase eb1429f..a760569 onto eb1429f (3 commands)
```

We want to rebase on top of our first commit, so we change the other two commits to *squash*:

```
pick eb36612 initial commit
squash 9ac8968 second commit
squash a760569 final commit
```

After this, should you wish to update your commit message to better summarise all of your pull request, run:

```
git commit --amend
```

You will then need to force push (assuming your initial commit(s) were posted to github):

```
git push origin your-branch --force
```

Docker Test Environment

Python Keylime with a TPM emulator can be deployed using Docker. Since this docker configuration uses a TPM emulator, it should only be used for development or testing and NOT in production.

Please see either the Dockerfiles or our local CI script which will automate the build and pull of Keylime.

6.2 Updating Configurations

When configuration changes are introduced, a new mapping and configuration templates should be created (at least once per release where configuration changes were introduced).

When only adding new configuration options, the configuration file is still compatible with the previous version since all the options that existed before are still present. In this case, the configuration minor version number should be bumped.

When an option is removed or renamed, the configuration file is no longer compatible with the previous version. In this case, the major version number should be bumped.

6.2.1 Adding upgrade template files

For each configuration version, it should exist a corresponding directory under the `templates`. For example, for the configuration version `2.0`, the directory `templates/2.0` was added.

For each new version, the following files should be created:

- A directory for the new version number should be created in the `templates` directory
- The `mapping.json` file that specify the transformations from the previous configuration version to the new version should be added. See the format in *Mapping file format*
- For each existing component, the corresponding configuration file jinja2 template should be added. For example, for the `verifier`, the `verifier.j2` should be created in the directory for the new version
- If special transformations that cannot be expressed through the simple operations in the `mapping.json` file, the `adjust.py` script can be added. See below the requirements for the `adjust.py` in *Adjust script requirements*

For example, when adding templates for a new version `X.Y`, the following directory tree should be added:

```
templates
└── X.Y
    ├── adjust.py (optional)
    ├── agent.j2
    ├── ca.j2
    ├── logging.j2
    ├── mapping.json
    ├── registrar.j2
    ├── tenant.j2
    ├── test.md
    └── verifier.j2
```

6.2.2 Mapping file format

For each configuration version, a mapping from the previous version to the new version is required. The mapping is provided as a JSON file, with the following fields:

- `version`: The mapping version, in the `MAJOR.MINOR` format. The value assigned to this field should match the template directory name. For example, for the mapping in the `templates/2.0` directory, the `version` field should be set as `2.0`.
- `type`: The mapping type. The supported values are `update` and `full`. See the requirements for each type below in *Update mapping* and *Full mapping*, respectively.
- `components`: The components to be modified by the mapping. Depending on the mapping type, the contents of the `components` field are different. See the requirements for each type below in *Update mapping* and *Full mapping*.
- `subcomponents`: Only present in the `full` mapping type, this field is required to associate sections to their parent components.
- Full mapping: The `type` field in the `mapping.json` file should be set as `full` for the mapping to be processed as a full mapping. In this type of mapping, all the options for all the components are treated as replacements of options. If an option is omitted, it means the option is removed in the new configuration version.

6.2.3 Update mapping

For the update mapping, the `type` field in the `mapping.json` file should be set as `update`. The update mapping is used to create a new configuration version by applying changes to the existing options through operations and preserve all the other options. This mapping type is the recommended way to introduce small changes to the configuration files.

In the update mapping, the `components` field list the components that are modified from the previous version, and the operations applied to them. See the format of the components dictionary below in the *Update mapping components format*

The update mapping does not use or need the `subcomponents` field.

Update mapping components format

For the update mapping, the `components` field should be set as a dictionary which maps the sections to be modified to the operations to be applied to them. Only the sections that are modified need to be present in the dictionary, all the omitted components are preserved as they were in the previous version.

The supported operations to modify the sections are:

- `add`: This operation adds a new option to the section. It should be assigned as a dictionary mapping the new option name to its default value.

  For example, the following mapping file adds 2 new options to the `[comp_a]` section:

```
{
    "version": "3.1",
    "type": "update",
    "components": {
        "comp_a": {
            "add": {
                "new_option": "value",
                "new_option2": "value2"
            }
        }
    }
}
```

- `remove`: This operation removes options that exists in the previous configuration version. An array of options to be removed should be assigned to the `remove` field for the section/component to be modified.

  For example, the following mapping file removes 2 options from the `[comp_a]` section:

```
{
    "version": "3.1",
    "type": "update",
    "components": {
        "comp_a": {
            "remove": ["unused_option", "another_unused_option"]
        }
    }
}
```

- `replace`: This operation replaces an option that exists in the previous configuration version with another in the new version. During the processing of the mapping file, the value found in the replaced option will be preserved, and assigned to the new option in the output. If the option is not found in the input configuration file, the default value is used instead.

  The `replace` field should be set as a dictionary mapping the option to be replaced to the parameters for the new option. The dictionary should have the following fields:

  - `section`: The section to which the new option should be added
  - `option`: The new option name
  - `default`: The default value to be used in case the old option is not found in the input configuration.

  For example, the following mapping file replaces two options from the `[comp_a]` section:

```
{
    "version": "3.1",
    "type": "update",
    "components": {
        "comp_a": {
            "replace": {
                "old_option_to_replace": {
                    "section": "new_section",
                    "option": "new_option",
                    "default": "value"
                },
                "old_value": {
                    "section": "other_section",
                    "option": "other_new_option",
                    "default": "value"
                }
            }
        }
    }
}
```

6.2.4 Full mapping

In the full mapping, al the options of the new configuration version should be declared. If an option is omitted, it means the option is removed.

The format of the fields in the full mapping file are:

- `version`: Should be in the `MAJOR.MINOR` format. The version should match the directory name
- `type`: Should be set as `full`. If omitted, the mapping will be treated as as full mapping
- `components`: Should be set as a dictionary which maps each component to a dictionary of options. See the option dictionary format below in *Full mapping components format* section.
- `subcomponents`: Should be set as a dictionary mapping subcomponents to its main component. This is necessary to create a relationship between the sections of the files that are not components (e.g. The `[revocations]` section in the `verifier.conf` file should be declared in the `subcomponents` dictionary as `"revocations": "verifier"`). See the format below in *Subcomponents format*

Full mapping components format

For each component, the options transformations should be declared through dictionaries that map the **new option** name to the **option it is replacing**.

The upgrade script will search the options to be replaced in the old configuration, in the section provided in the `section` field. If the option is found, the value is preserved in the news configuration, otherwise the value provided in the `default` field is used instead.

As an example, follows an excerpt of the `templates/2.0/mapping.json` file:

```
"components": {
    "agent": {
        "version": {
            "section": "agent",
            "option": "version",
            "default": "2.0"
        },
        "revocation_notification_ip": {
            "section": "general",
            "option": "receive_revocation_ip",
            "default": "127.0.0.1"
        },
}
```

In the excerpt above, in the `agent` component two options are declared, `version` and `revocation_notification_ip`.

The new option `revocation_notification_ip` will receive the value from the `receive_revocation_ip` from the `general` section in the old configuration file. If the option is not found, the value `127.0.0.1` provided in the `default` field is used instead.

Subcomponents format

The configuration file for some of the components (e.g. the `verifier.conf`) have more than one section. The main section is named after the component (e.g. `[verifier]` section of the `verifier.conf` file). The other sections are considered subsections, or subcomponents, by the configuration upgrade script. The subsections are associated with their main section in the `Subcomponents` dictionary, which maps a subsection to the associated main section.

For example, the following excerpt from the `templates/2.0/mapping.json` file:

```
"subcomponents": {
    "revocations": "verifier",
    "loggers": "logging",
    "handlers": "logging",
    "formatters": "logging",
    "formatter_formatter": "logging",
    "logger_root": "logging",
    "handler_consoleHandler": "logging",
    "logger_keylime": "logging"
}
```

In the excerpt, the `[revocations]` section is declared as a subsection (or subcomponent) of the `[verifier]` section.

6.2.5 Adjust script requirements

The optional `adjust.py` script can perform complex operations that cannot be expressed using the `mapping.json` file. For example, deciding the value of an option depending on the presence of another option in the configuration file.

The `adjust.py` script is processed after the `mapping.json` is applied. For this reason, when writing the `adjust.py` script, the author should consider the input to be the output of the processing of the associated `mapping.json` file.

The only requirement for the `adjust.py` script is to implement the `adjust()` function, defined as the following:

```
def adjust(
    config: RawConfigParser, mapping: Dict, logger: Logger = logging.getLogger(__name__)
) -> None:
```

The `config` parameter is the result after applying the transformations defined by the `mapping.json` file.

The `mapping` parameter is the dictionary read from the `mapping.json` JSON file.

The optional `logger` parameter will receive the logger from the `keylime_upgrade_config` script, so that all the log messages are in a single place. It is recommended to keep the default assigned as `logging.getLogger(__name__)` so that the `keylime_upgrade_config` can set the log level accordingly.

The `adjust()` function should make the changes to the parser received through the `config` parameter directly.

CHAPTER

SEVEN

SECURING KEYLIME

Warning

This page is still under development and not complete. It will be so until this warning is removed.

7.1 System Hardening

7.2 TLS configuration

7.3 Reporting an issue

Please contact us directly at security@keylime.groups.io for any bug that might impact the security of this project. Do not use a github issue to report any potential security bugs.

CHAPTER

EIGHT

KEYLIME_TENANT

8.1 Keylime tenant management tool for agent provisioning and policy management

Manual section
1

Author
Keylime Developers

Date
July 2025

8.1.1 SYNOPSIS

keylime_tenant [*OPTIONS*] [*COMMAND*]

(Most operations require root privileges, use with sudo)

8.1.2 DESCRIPTION

keylime_tenant is the primary command-line interface for managing Keylime agents and policies. It allows users to provision agents with TPM-based attestation, manage runtime policies, measured boot policies, and interact with Keylime registrar and verifier services.

The tenant can add, delete, update, and monitor agents, as well as manage various types of policies including runtime policies (for IMA/EVM attestation) and measured boot policies (for boot-time attestation). It supports both push and pull models for agent communication.

You can run keylime_tenant on the same system as the Keylime registrar or verifier, or on a separate system.

8.1.3 COMMANDS

-c, –command ***COMMAND***
Specify the command to execute. Valid commands are:

- **add**: Add a new agent to the system (default)
- **delete**: Remove an agent from the system
- **update**: Update an existing agent's configuration
- **regstatus**: Show agent status from registrar
- **cvstatus**: Show agent status from cloud verifier
- **status**: Show combined agent status

- **reglist**: List all agents in registrar
- **cvlist**: List all agents in cloud verifier
- **reactivate**: Reactivate a failed agent
- **regdelete**: Delete agent from registrar only
- **bulkinfo**: Get bulk information about agents
- **addruntimepolicy**: Add a runtime policy (requires –runtime-policy or –allowlist)
- **showruntimepolicy**: Display a runtime policy (requires –runtime-policy-name)
- **deleteruntimepolicy**: Remove a runtime policy (requires –runtime-policy-name)
- **updateruntimepolicy**: Update a runtime policy (requires –runtime-policy-name)
- **listruntimepolicy**: List all runtime policies
- **addmbpolicy**: Add a measured boot policy (requires –mb-policy-name)
- **showmbpolicy**: Display a measured boot policy (requires –mb-policy-name)
- **deletembpolicy**: Remove a measured boot policy (requires –mb-policy-name)
- **updatembpolicy**: Update a measured boot policy (requires –mb-policy-name)
- **listmbpolicy**: List all measured boot policies

8.1.4 OPTIONS

-h, –help
Show help message and exit

–push-model
Enable push model (avoid requests to keylime-agent)

-t, –targethost ***AGENT_IP***
The IP address of the host to provision

-tp, –targetport ***AGENT_PORT***
The port of the host to provision

-r, –registrarhost ***REGISTRAR_IP***
The IP address of the registrar where to retrieve the agents data from

-rp, –registrarport ***REGISTRAR_PORT***
The port of the registrar

–cv_targethost ***CV_AGENT_IP***
The IP address of the host to provision that the verifier will use (optional). Use only if different than argument to option -t/–targethost

-v, –cv ***VERIFIER_IP***
The IP address of the cloud verifier

-vp, –cvport ***VERIFIER_PORT***
The port of the cloud verifier

-vi, –cvid ***VERIFIER_ID***
The unique identifier of a cloud verifier

-nvc, –no-verifier-check
Disable the check to confirm if the agent is being processed by the specified verifier. Use only with -c/–command delete or reactivate

-u, –uuid *AGENT_UUID*
UUID for the agent to provision

-f, –file *FILE*
Deliver the specified plaintext file to the provisioned agent

–cert *CA_DIR*
Create and deliver a certificate using a CA created by ca-util. Pass in the CA directory or use "default" to use the standard directory

-k, –key *KEYFILE*
An intermediate key file produced by user_data_encrypt

-p, –payload *PAYLOAD*
Specify the encrypted payload to deliver with encrypted keys specified by -k

–include *INCL_DIR*
Include additional files in provided directory in certificate zip file. Must be specified with –cert

–runtime-policy *RUNTIME_POLICY*
Specify the file path of a runtime policy

–runtime-policy-checksum *RUNTIME_POLICY_CHECKSUM*
Specify the SHA-256 checksum of a runtime policy

–runtime-policy-sig-key *RUNTIME_POLICY_SIG_KEY*
Specify the public key file used to validate the runtime policy signature

–runtime-policy-url *RUNTIME_POLICY_URL*
Specify the URL of a remote runtime policy

–runtime-policy-name *RUNTIME_POLICY_NAME*
The name of the runtime policy to operate with

–mb-policy *MB_POLICY*
The measured boot policy to operate with

–mb-policy-name *MB_POLICY_NAME*
The name of the measured boot policy to operate with

–tpm_policy *TPM_POLICY*
Specify a TPM policy in JSON format. Example: {"15":"0000000000000000000000000000000000000000"}

–verify
Block on cryptographically checked key derivation confirmation from the agent once it has been provisioned

–supported-version *SUPPORTED_VERSION*
API version that is supported by the agent. Detected automatically by default

8.1.5 DEPRECATED OPTIONS

The following options are deprecated and may be removed in future versions:

–allowlist *ALLOWLIST*
DEPRECATED: Migrate to runtime policies for continued functionality. Specify the file path of an allowlist

–allowlist-url *ALLOWLIST_URL*
DEPRECATED: Migrate to runtime policies for continued functionality. Specify the URL of a remote allowlist

–allowlist-name *ALLOWLIST_NAME*
DEPRECATED: Migrate to runtime policies for continued functionality. The name of allowlist to operate with

–exclude *IMA_EXCLUDE*
: **DEPRECATED**: Migrate to runtime policies for continued functionality. Specify the location of an IMA exclude list

–mb_refstate *MB_POLICY*
: **DEPRECATED**: Use –mb-policy instead. Specify the location of a measured boot reference state

–signature-verification-key *IMA_SIGN_VERIFICATION_KEYS*
: **DEPRECATED**: Provide verification keys as part of a runtime policy for continued functionality. Specify an IMA file signature verification key

8.1.6 EXAMPLES

Add a new agent:

```
sudo keylime_tenant -c add -t 192.168.1.100 -u agent-001
```

Add an agent with runtime policy:

```
sudo keylime_tenant -c add -t 192.168.1.100 -u agent-001 --runtime-policy /path/to/
↪policy.json
```

Check agent status:

```
sudo keylime_tenant -c status -u agent-001
```

Delete an agent:

```
sudo keylime_tenant -c delete -u agent-001
```

List all agents:

```
sudo keylime_tenant -c cvlist
```

Add a runtime policy:

```
sudo keylime_tenant -c addruntimepolicy --runtime-policy-name my-policy --runtime-policy␣
↪/path/to/policy.json
```

Add a measured boot policy:

```
sudo keylime_tenant -c addmbpolicy --mb-policy-name my-mb-policy --mb-policy /path/to/mb-
↪policy.json
```

Provision agent with certificate delivery:

```
sudo keylime_tenant -c add -t 192.168.1.100 -u agent-001 --cert default
```

Provision agent with custom verifier:

```
sudo keylime_tenant -c add -t 192.168.1.100 -u agent-001 -v 192.168.1.200 -vp 8881
```

8.1.7 FILES

/etc/keylime/tenant.conf
: Default configuration file for keylime_tenant. Contains all tenant related settings.

8.1.8 PREREQUISITES

- Keylime verifier service running (default: 127.0.0.1:8881)
- Keylime registrar service running (default: 127.0.0.1:8891)
- Root privileges (use sudo)
- Network connectivity to registrar and verifier services
- Valid TLS configuration in /etc/keylime/tenant.conf

8.1.9 SEE ALSO

keylime_verifier(8), **keylime_registrar**(8), **keylime_agent**(8)

For more information about Keylime, visit: https://keylime.dev

8.1.10 BUGS

Report bugs to the Keylime project at: https://github.com/keylime/keylime/issues

CHAPTER

NINE

KEYLIME_VERIFIER

9.1 Keylime verifier service for agent attestation

Manual section
8

Author
Keylime Developers

Date
September 2025

9.1.1 SYNOPSIS

keylime_verifier

(Most operations require root privileges, use with sudo)

9.1.2 DESCRIPTION

The verifier is a long-running service that attests registered agents. It accesses the registrar database to obtain agent data, and optionally performs measured boot evaluation and durable attestation. The service does not accept command-line options; its behavior is configured via configuration files and environment variables, and it is managed by keylime tenant.

9.1.3 CONFIGURATION

Primary configuration is read from `/etc/keylime/verifier.conf` (or an override via env). All options are under the `[verifier]` section.

Essentials: - **mode**: Attestation mode (`pull` or `push`). Default: `pull` - **uuid**: Unique identifier for this verifier instance - **ip**, **port**: Bind address and HTTP port. Use a concrete address to limit

the verifier to one interface, `0.0.0.0` to listen on all IPv4 interfaces, or `::` to listen on all IPv6 interfaces (which also accepts IPv4 on dual-stack hosts).

- **registrar_ip**, **registrar_port**: Registrar endpoint
- **enable_agent_mtls**: Enable mTLS with agents and tenant
- **tls_dir**: TLS material location
  - `generate`: auto-generate CA, client and server keys/certs under `$KEYLIME_DIR/cv_ca`
  - `default`: use existing materials under `$KEYLIME_DIR/cv_ca`
- **server_key**, **server_key_password**, **server_cert**: Server TLS files

- **client_key**, **client_key_password**, **client_cert**: Client TLS files
- **trusted_client_ca**, **trusted_server_ca**: CA lists
- **database_url**: SQLAlchemy URL; value `sqlite` maps to `$KEYLIME_DIR/cv_data.sqlite`
- **database_pool_sz_ovfl**: Pool size, overflow (non-sqlite)
- **auto_migrate_db**: Apply DB migrations on startup
- **num_workers**: Number of worker processes (`0` = CPU count)
- **max_workers**: Maximum worker processes; actual count is `min(cpu_count, max_workers)` (`0` = no limit, default `16`)
- **exponential_backoff**, **retry_interval**, **max_retries**: Retry behavior for agent comm
- **quote_interval**: Time between integrity checks (seconds)
- **max_upload_size**: Upload size limit (bytes)
- **request_timeout**: Agent request timeout (seconds)
- **shutdown_drain_timeout**: Max time (seconds) to wait for in-flight operations during shutdown
- **measured_boot_policy_name**, **measured_boot_imports**, **measured_boot_evaluate**: measured boot policy settings
- **severity_labels**, **severity_policy**: revocation severity config
- **ignore_tomtou_errors**: handle ToMToU IMA entries (bool)
- **durable_attestation_import** and related **persistent_store_url**, **transparency_log_url**, **time_stamp_authority_url**, **time_stamp_authority_certs_path**, **persistent_store_format**, **persistent_store_encoding**, **transparency_log_sign_algo**, **signed_attributes**: durable attestation
- **require_allow_list_signatures**: require signed allowlists (bool)

9.1.4 ENVIRONMENT

- **KEYLIME_VERIFIER_CONFIG**: Path to verifier.conf (highest priority)
- **KEYLIME_LOGGING_CONFIG**: Path to logging.conf
- **KEYLIME_DIR**: Working directory (default: `/var/lib/keylime`)
- **KEYLIME_TEST**: `on/true/1` enables testing mode (looser checks; WORK_DIR becomes CWD)

9.1.5 FILES

- `/etc/keylime/verifier.conf`
- `/etc/keylime/logging.conf`
- `$KEYLIME_DIR/cv_data.sqlite` (when `database_url = sqlite`)
- `$KEYLIME_DIR/cv_ca` (when `tls_dir = default` or `generate`)
- systemd unit: `keylime_verifier.service`

9.1.6 RUNTIME

Start from system install:

```
sudo keylime_verifier
```

Start as a systemd service:

```
systemctl enable --now keylime_verifier
```

Open firewall ports (adjust if you changed ports):

```
firewall-cmd --add-port 8881/tcp
firewall-cmd --runtime-to-permanent
```

9.1.7 NOTES

- Verifier initializes measured boot components on startup.
- With `tls_dir = generate`, the verifier creates CA/keys/certs in `$KEYLIME_DIR/cv_ca` used by other components.

9.1.8 SEE ALSO

keylime_registrar(8), **keylime_tenant**(1), **keylime_agent**(8), **keylime_push_model_agent**(8)

9.1.9 BUGS

Report bugs at https://github.com/keylime/keylime/issues

CHAPTER

TEN

KEYLIME_REGISTRAR

10.1 Keylime registrar service for agent registration

Manual section
8

Author
Keylime Developers

Date
September 2025

10.1.1 SYNOPSIS

keylime_registrar

(Most operations require root privileges, use with sudo)

10.1.2 DESCRIPTION

The registrar is a long-running service used by agents. It maintains its own database where it stores data of registered agents. The service does not accept command-line options; behavior is configured via configuration files and environment variables, and is managed by keylime tenant.

10.1.3 CONFIGURATION

Primary configuration is read from `/etc/keylime/registrar.conf` (or an override via env). All options are under the `[registrar]` section.

Essential configuration options:

ip
Bind address

port
HTTP port

tls_port
HTTPS port

tls_dir
TLS material location (`generate` for auto-generate CA, keys, certs under `$KEYLIME_DIR/reg_ca`, `default` for shared verifier CA under `$KEYLIME_DIR/cv_ca`)

server_key, server_key_password, server_cert, trusted_client_ca
TLS files

database_url
SQLAlchemy URL; value `sqlite` maps to `$KEYLIME_DIR/reg_data.sqlite`

database_pool_sz_ovfl
Pool size, overflow (non-sqlite)

max_workers
Maximum worker processes; actual count is `min(cpu_count, max_workers)` (`0` = no limit, default `16`)

auto_migrate_db
Apply DB migrations on startup

max_upload_size
Request body limit (bytes)

tpm_identity
Allowed identity (`default`, `ek_cert_or_iak_idevid`, `ek_cert`, `iak_idevid`)

malformed_cert_action
`warn` (default), `reject`, or `ignore`

durable_attestation_import (optional)
Python import path to enable Durable Attestation

10.1.4 ENVIRONMENT

KEYLIME_REGISTRAR_CONFIG
Path to registrar.conf (highest priority)

KEYLIME_LOGGING_CONFIG
Path to logging.conf

KEYLIME_DIR
Working directory (default: `/var/lib/keylime`)

KEYLIME_TEST
`on`/`true`/`1` enables testing mode (looser checks; WORK_DIR becomes CWD)

10.1.5 FILES

`/etc/keylime/registrar.conf`
Registrar configuration file

`/etc/keylime/logging.conf`
Logging configuration

`$KEYLIME_DIR/reg_data.sqlite`
Database file when `database_url = sqlite`

`$KEYLIME_DIR/reg_ca`
TLS certificates when `tls_dir = generate`

`$KEYLIME_DIR/cv_ca`
Shared verifier certificates when `tls_dir = default`

10.1.6 RUNTIME

Start from system install:

```
sudo keylime_registrar
```

Start as a systemd service:

```
systemctl enable --now keylime_registrar
```

Open firewall ports (adjust if you changed ports):

```
firewall-cmd --add-port=8890/tcp --add-port=8891/tcp
firewall-cmd --runtime-to-permanent
```

10.1.7 NOTES

- HTTPS is required for routes unless explicitly allowed insecure by the service.
- With `tls_dir = default`, start the verifier before the registrar so the shared CA/certs exist in `$KEYLIME_DIR/cv_ca`.
- The service forks worker processes (default: CPU count, capped by `max_workers`).
- Registrar and verifier may run on the same host or on separate hosts.

10.1.8 SEE ALSO

keylime_verifier(8), **keylime_tenant**(1), **keylime_agent**(8)

10.1.9 BUGS

Report bugs at https://github.com/keylime/keylime/issues

CHAPTER

ELEVEN

KEYLIME_AGENT

11.1 Keylime agent service for TPM-based attestation

Manual section
8

Author
Keylime Developers

Date
September 2025

11.1.1 SYNOPSIS

keylime_agent

(Most operations require root privileges, use with sudo)

11.1.2 DESCRIPTION

The agent is a long-running service that runs on systems to be attested. It communicates with the TPM to generate quotes, collects IMA and measured boot event logs, and provides secure payload functionality. The service does not accept command-line options; behavior is configured via TOML configuration files.

11.1.3 CONFIGURATION

Primary configuration is read from `/etc/keylime/agent.conf` (or an override via env). Configuration uses TOML format. All options are under the `[agent]` section.

Drop-in overrides: files in `/etc/keylime/agent.conf.d/` are applied in lexicographic order.

Essential configuration options:

uuid
Agent identifier (`generate`, `hash_ek`, `environment`, `dmidecode`, `hostname`, or explicit UUID)

ip, port
Bind address and port (default: 9002)

contact_ip, contact_port
External contact address (optional)

registrar_ip, registrar_port
Registrar endpoint

enable_agent_mtls
Enable mTLS communication

tls_dir
TLS material location (`generate` for auto-generate under `$KEYLIME_DIR/cv_ca`, `default` for `$KEYLIME_DIR/secure`)

server_key, server_key_password, server_cert
TLS files (self-signed cert)

trusted_client_ca
Trusted client CA list

enc_keyname
Payload encryption key file name

dec_payload_file
Decrypted payload file name

secure_size
tmpfs partition size for secure storage

tpm_ownerpassword
TPM owner password (`generate` for random)

extract_payload_zip
Auto-extract zip payloads (bool)

enable_revocation_notifications
Listen for revocation via ZeroMQ (bool)

revocation_notification_ip, revocation_notification_port
ZeroMQ endpoint

revocation_cert
Certificate to verify revocation messages

revocation_actions
Python scripts to run on revocation

payload_script
Script to run after payload extraction

enable_insecure_payload
Allow payloads without mTLS (insecure)

measure_payload_pcr
PCR to extend with payload (-1 to disable)

exponential_backoff, retry_interval, max_retries
TPM communication retry

tpm_hash_alg, tpm_encryption_alg, tpm_signing_alg
TPM algorithms

ek_handle
EK handle (`generate` or explicit handle like `0x81000000`)

enable_iak_idevid
Enable IAK/IDevID usage (bool)

iak_idevid_template, iak_idevid_asymmetric_alg, iak_idevid_name_alg
IAK/IDevID config

idevid_password, idevid_handle, iak_password, iak_handle
Persistent key handles

iak_cert, idevid_cert
Certificate file names

run_as
User:group to drop privileges to

ima_ml_path
IMA measurement log path (default: `/sys/kernel/security/ima/ascii_runtime_measurements`)

measuredboot_ml_path
Measured boot log path (default: `/sys/kernel/security/tpm0/binary_bios_measurements`)

11.1.4 ENVIRONMENT

KEYLIME_AGENT_CONFIG
Path to agent.conf (highest priority)

KEYLIME_LOGGING_CONFIG
Path to logging.conf

KEYLIME_DIR
Working directory (default: `/var/lib/keylime`)

KEYLIME_AGENT_UUID
UUID when `uuid = environment`

KEYLIME_AGENT_IAK_CERT
Override iak_cert path

KEYLIME_AGENT_IDEVID_CERT
Override idevid_cert path

KEYLIME_TEST
`on/true/1` enables testing mode

11.1.5 FILES

`/etc/keylime/agent.conf`
TOML format configuration file

`/etc/keylime/agent.conf.d/`
Drop-in snippets; read in lexicographic order

`/etc/keylime/logging.conf`
Logging configuration

`$KEYLIME_DIR/secure/`
Secure tmpfs mount for keys/payloads

`$KEYLIME_DIR/cv_ca/`
TLS certificates when `tls_dir = generate`

`$KEYLIME_DIR/tpmdata.yml`
TPM state persistence

11.1.6 RUNTIME

Start from system install:

```
sudo keylime_agent
```

Start as a systemd service:

```
sudo systemctl enable --now keylime_agent
```

Open firewall port:

```
sudo firewall-cmd --add-port=9002/tcp
sudo firewall-cmd --runtime-to-permanent
```

11.1.7 PREREQUISITES

- Root privileges (use sudo)
- TPM 2.0 available (verify with `tpm2_pcrread`)
- IMA enabled in kernel
- Network connectivity to registrar

11.1.8 NOTES

- Agent uses TOML configuration format (unlike other Keylime components).
- The Rust agent is the current implementation; Python agent is deprecated.
- Agent generates self-signed certificates for mTLS if not provided.

11.1.9 SEE ALSO

keylime_verifier(8), **keylime_registrar**(8), **keylime_tenant**(1)

11.1.10 BUGS

Report bugs at https://github.com/keylime/rust-keylime/issues

CHAPTER

TWELVE

KEYLIME_PUSH_MODEL_AGENT

12.1 Keylime push-model agent for TPM-based remote attestation

Manual section
8

Author
Keylime Developers

Date
February 2026

12.1.1 SYNOPSIS

keylime_push_model_agent [*OPTIONS*]

(Most operations require root privileges, use with sudo)

12.1.2 DESCRIPTION

The push-model agent is a long-running service that runs on systems to be attested. Unlike the standard Keylime agent which acts as a server and waits for the verifier to poll it, the push-model agent initiates connections to the verifier and proactively submits attestation evidence.

The agent registers with the registrar, authenticates with the verifier using Proof of Possession (PoP), and performs periodic attestation cycles consisting of capabilities negotiation and evidence submission.

This agent uses API version 3.0 and requires the verifier to be configured in push mode (`mode = push`).

12.1.3 OPTIONS

–verifier-url ***URL***
URL of the verifier (must use HTTPS). Default: `https://localhost:8881`

–registrar-url ***URL***
URL of the registrar. Default: `http://127.0.0.1:8888`

–agent-identifier ***ID***
Agent UUID. Overrides the `uuid` configuration option.

–attestation-interval-seconds ***SECONDS***
Interval between attestation cycles. Default: `60`

–ca-certificate ***PATH***
CA certificate file for verifying the verifier's TLS certificate. Overrides `verifier_tls_ca_cert`.

–api-version *VERSION*
API version to use. Default: `v3.0`

–timeout *MILLISECONDS*
HTTP request timeout. Default: `5000`

–insecure
Accept invalid TLS certificates. For testing only.

–avoid-tpm
Use a mock TPM instead of hardware TPM. For testing only.

–json-file *FILE*
JSON file for payload data.

–attestation-index *INDEX*
Attestation index value. Default: `1`

–session-index *INDEX*
Session index value. Default: `1`

–message-type *TYPE*
Message type (Attestation, EvidenceHandling, Session). Default: `Attestation`

–method *METHOD*
HTTP method. Default: `POST`

12.1.4 CONFIGURATION

Primary configuration is read from `/etc/keylime/agent.conf` (TOML format). All options are under the `[agent]` section. Command-line arguments override configuration file values.

Drop-in overrides: files in `/etc/keylime/agent.conf.d/` are applied in lexicographic order.

Push-model specific options:

verifier_url
URL of the verifier. Must use HTTPS. Default: `https://localhost:8881`

verifier_tls_ca_cert
Path to CA certificate for verifying the verifier's TLS certificate. Relative paths are resolved from `keylime_dir`. Default: `cv_ca/cacert.crt`

attestation_interval_seconds
Interval in seconds between attestation cycles. Default: `60`

api_versions
API versions to use. Default: `3.0`

certification_keys_server_identifier
Server identifier for attestation key certification. Default: `ak`

uefi_logs_evidence_version
UEFI logs evidence format version. Default: `2.1`

exponential_backoff_initial_delay
Initial retry delay in milliseconds. Default: `10000`

exponential_backoff_max_retries
Maximum number of retry attempts. Default: `5`

exponential_backoff_max_delay
Maximum retry delay in milliseconds. Default: `300000`

Shared options (same as standard agent):

uuid
Agent identifier. Default: auto-generated UUID.

registrar_ip, registrar_port
Registrar endpoint. Default: `127.0.0.1:8890`

registrar_tls_enabled
Enable TLS for registrar communication. Default: `false`

registrar_tls_ca_cert
CA certificate for registrar TLS verification. Default: `cv_ca/cacert.crt`

tpm_hash_alg, tpm_encryption_alg, tpm_signing_alg
TPM algorithms. Defaults: `sha256`, `rsa`, `rsassa`

keylime_dir
Working directory. Default: `/var/lib/keylime`

run_as
User:group to drop privileges to. Default: `keylime:tss`

enable_iak_idevid
Enable IAK/IDevID usage. Default: `false`

12.1.5 ENVIRONMENT

KEYLIME_AGENT_CONFIG
Path to agent.conf (highest priority)

KEYLIME_DIR
Working directory (default: `/var/lib/keylime`)

RUST_LOG
Log level configuration. Default in systemd service: `keylime_push_model_agent=info,keylime=info`

All configuration options can be overridden via environment variables in the form `KEYLIME_AGENT_<OPTION_NAME>` (e.g. `KEYLIME_AGENT_VERIFIER_URL`).

12.1.6 FILES

`/etc/keylime/agent.conf`
TOML format configuration file (shared with standard agent)

`/etc/keylime/agent.conf.d/`
Drop-in configuration snippets

`/var/lib/keylime/cv_ca/cacert.crt`
Default CA certificate for verifier TLS verification

`/var/lib/keylime/agent_data.json`
Persisted agent TPM data

12.1.7 RUNTIME

Start directly:

```
sudo keylime_push_model_agent --verifier-url https://verifier.example.com:8881
```

Start as a systemd service:

```
sudo systemctl enable --now keylime_push_model_agent
```

Check service status:

```
sudo systemctl status keylime_push_model_agent
sudo journalctl -u keylime_push_model_agent -f
```

12.1.8 PREREQUISITES

- Root privileges (use sudo)
- TPM 2.0 available (verify with `tpm2_pcrread`)
- Verifier configured with `mode = push`
- Network connectivity from agent to verifier and registrar
- Verifier CA certificate available on agent machine

12.1.9 NOTES

- This service conflicts with `keylime_agent.service`. Only one agent type can run on a machine at a time.
- The push-model agent does not expose any listening ports.
- Push-model attestation is currently experimental.
- Authentication uses PoP bearer tokens, not mTLS client certificates.

12.1.10 SEE ALSO

keylime_agent(8), **keylime_verifier**(8), **keylime_registrar**(8), **keylime_tenant**(1)

12.1.11 BUGS

Report bugs at https://github.com/keylime/rust-keylime/issues

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- search

HTTP ROUTING TABLE

/v2.5

/v3

/v3.0

/version