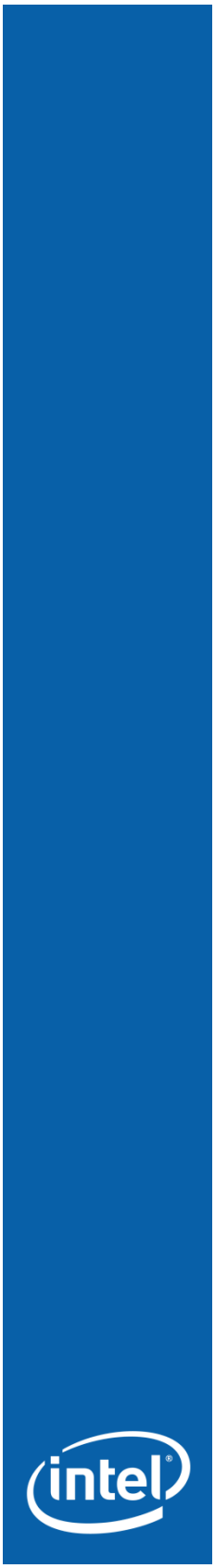

Remote Attestation for Multi-Package Platforms using Intel® SGX Datacenter Attestation Primitives (DCAP)

Rev 1.1
September, 2021

Remote Attestation for Multi-Package Platforms using Intel® SGX Datacenter Attestation Primitives (DCAP)

© Intel Corporation

Remote Attestation for Multi-Package Platforms using Intel® SGX Datacenter Attestation Primitives (DCAP)

Table of Contents

1. Introduction

Intel® SGX remote attestation on multi-package platforms requires SGX instructions to use platform keys that are shared between CPU packages. These platform keys are generated in the field when a platform is assembled and distributed to each CPU package on the platform. The BIOS stores the platform keys for each CPU package that is encrypted by respective unique hardware key of each CPU package. SGX attestation keys are certified with signing keys derived from the platform keys rather than a unique hardware key of a SGX CPU package. This differs from a single package SGX platforms where the attestation keys are certified with signing keys that are derived from unique hardware key of the CPU package.

Because the platform keys are derived at platform assembly, the provisioning keys that are derived from the platform keys are not recognized by the attestation infrastructure and they must be registered. A registration authority service evaluates whether each CPU package on the platform is a genuine SGX package in good standing using its database of hardware key certificates. Then the registration authority service provides platform identity certificates to the attestation infrastructure. Once when the platform is registered, it can perform remote attestation the same way the single package SGX platforms perform remote attestation is described in other Intel® SGX Data Center Attestation Primitive (DCAP) documentation.

This document provides information on the released Intel® SGX DCAP platform software and tools that support multi-package registration and a brief overview of the multi-package boot flows and components.

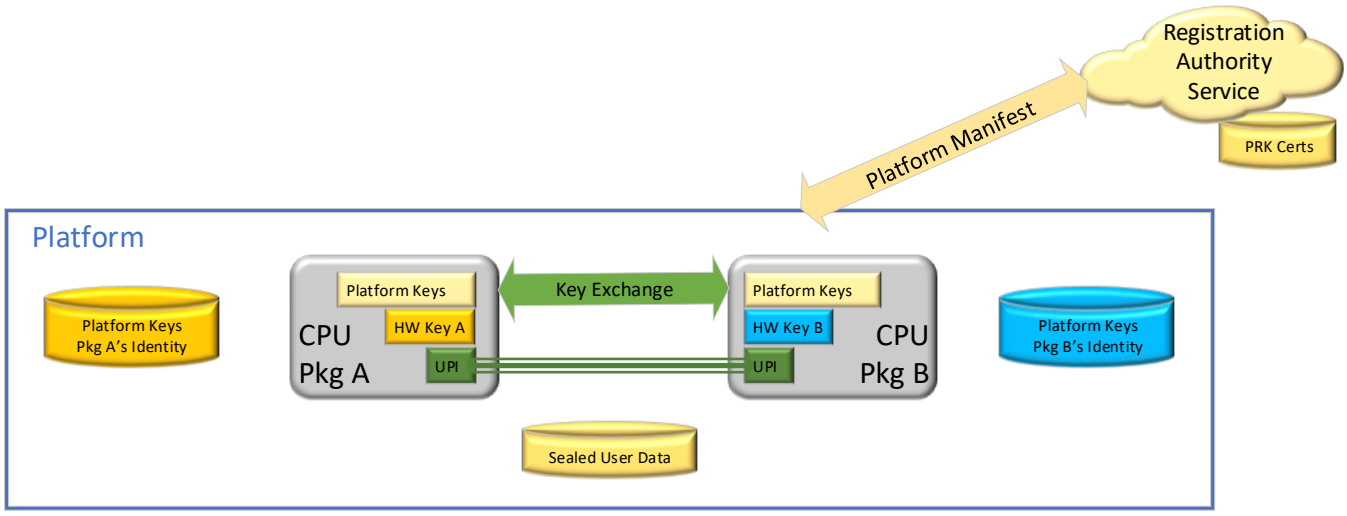

Figure 1: Multi-Package Server Overview

1.1.Terminology

Intel® SGX DCAP	Intel® Software Guard Extensions Data Center Attestation Primitives

Registration Authority Service	The Registration Authority Service is the foundation for provisioning and attesting multi-package platforms. The PCK Certificate Provisioning Service & attestation verifiers rely on the assertion of the Registration Server that this platform is trustworthy. Intel hosts Intel® Registration Service for this purpose.
Registration Service Authentication Key (RSAK)	Key that the registration authority service uses to sign authorizations to add new packages to the platform and its self-signed Registration Server ID structure.
Registration Service Encryption Key (RSEK)	The registration authority service's 3072-bit RSA key used to encrypt/decrypt the platform keys.
Registration Service Name (RSNAME)	Self-selected public ID of the Registration Authority Service. Frequently the hash of the service domain name.
Security Version Number (SVN)	Version number that indicates when security relevant updates have occurred. New versions can have increased functional versions without incrementing the SVN.
Platform Key	This 128-bit key is the foundation of the provisioning key derivations in a processor. Multi-package platforms negotiate platform keys in the field. They are delivered to the registration authority service encrypted with the RSAK. They are stored by each CPU package on the platform in the sealed and encrypted key blobs using its respective unique hardware key. SGX Sealing Keys use the platform key in conjunction with another value that is unique to the platform instance. The platform key alone is not enough to unseal enclave-sealed user data.
Hardware Key	This is the unique key that is available to the each SGX-capable CPU package. It is derived directly from fuses and used to derive PRK signatures and key blob sealing keys.
Key Blob	Data Structure that stores the platform keys for each CPU device on the platform. Each device uses a unique sealing key to encrypt the platform keys in the key blob. Used by BIOS to determine the state of the platform keys.
Platform Manifest	The platform manifest allows the registration authority service to evaluate whether the platform and its components CPU packages are suitable for being

	certified as an SGX platform. It contains the shared platform keys that are encrypted using the Registration Server's Encryption Key (RSEK).
Provisioning Registration ID (PRID)	Unique 128-bit ID for each CPU package that is used for registration.
Platform Registration Key (PRK)	Unique 3072-bit RSA key pair for each processor package that is used for Initial Platform Establishment, TCB Recoveries and Add Package boot flows. This key is TCB-specific. It is used to sign the platform manifests. It is used in the protocol for establishing protected sessions between processors. It is derived from the HW key.
Provisioning Certification Enclave (PCE)	Intel® SGX architectural enclave that uses a Provisioning Certification Key (PCK) to sign REPORT structures for Provisioning or Quoting Enclaves. These signed REPORTS contain the ReportData indicating that attestation keys or provisioning protocol messages are created on genuine hardware.
Platform Provisioning ID (PPID)	Provisioning ID for a platform instance. PPID is not TCB dependent. The PCE generates the PPID.
Platform Registration Key Certificate (PRK Cert)	Binary certificates issued and signed by Intel for each multi-package CPU device. A new PRK Certificate is released for each CPU package when a microcode patch with a new SVN is released.
Platform Membership Certificate	Issued/signed by the Registration Service's Authentication Key (RSAK). They indicate that the specified package, running the specified CPUSVN, is authorized to access keys that are owned by the indicated Platform Instance at a specific SVN. It includes the Platform Registration ID (PRID), Platform Info, and Registration Server of that package.
Platform Security Version Numbers (PSVN)	The set of SVNs for all components in the Intel® SGX provisioning Trusted Computing Base (TCB) including the PCE's SVN.
Provisioning Certification Key (PCK)	Signing key that is available to Provisioning Certification Enclave for signing certificate-like QE REPORT structures. The key is unique to the processor package or platform instance, the HW TCB, and the PCE version (PSVN).
Provisioning Certification Key Certificate (PCK Cert)	The x.509 Certificate chain that is signed and distributed by the Registration Service for every SGX enabled multi-package platform. It matches the private key generated by the Provisioning Certification Enclave (PCE).

Intel® SGX Registration Service	Intel hosts a registration authority service called the Intel® SGX Registration Service. The PCK Certificate Provisioning Service & attestation verifiers rely on the Intel® SGX Registration Server's assertion that this platform is trustworthy.
Intel® SGX Provisioning TCB	The Trusted Computing Base of Intel® SGX provisioning. Include the platform HW TCB and the PCE's SVN.
PCEID	Identifies the version of the PCE that is used to generate the PPID and PCK signing key.
SGX Quote	Data structure that is used to provide evidence to an off-platform entity that an application enclave runs with Intel® SGX protections on a trusted Intel® SGX-enabled platform.
Quoting Enclave (QE)	The enclave that generates the attestation key used to sign SGX Quotes.
QE_ID	The default platform identifier that is used by the PCK Cert ID Retrieval Tool and the run-time PCK certificate requests. It is not a hardware identifier, and it is generated by the Quoting Enclave.
Intel® SGX Provisioning Certification Service (Intel® PCS)	Service hosted by Intel on the Internet that offers APIs for retrieving the Provisioning Certification Key (PCK) certificates and other Intel endorsements for generating and verifying SGX Quotes.
Reference Provisioning Certification Caching Service (PCCS)	A reference caching server to allow a CSP or a datacenter to cache PCK Certificates and other endorsements from the Intel® SGX Provisioning Certification in their local network.

Table 1-1: Terminology

2. Overview

2.1. SGX Multi-Package Attestation Components

The infrastructure for supporting SGX multi-package registration includes the microcode of CPU packages, the BIOS, the registration platform software, and a registration authority service (Intel® SGX Registration Service).

2.1.1. Microcode

Microcode is responsible for verifying that each CPU package is SGX-capable and making sure that each package has a consistent view of SGX-protected memory. It also generates the common platform keys that each package uses for SGX key derivation. Microcode is the only component that can generate the Provisioning Registration Key's (PRK) private key based on the CPU package's hardware key. The PRK private key is unique to each CPU package and to the security version number (SVN) of the loaded microcode patch.

Microcode generates a data structure called key blob for each CPU package in the platform. They contain shared platform keys and other information about the platform. Each key blob is encrypted and MAC'd with the respective hardware key. The key blobs are made available to the BIOS for persistent storage. When BIOS presents a set of valid and matching key blobs to the microcode on subsequent boot flows, the platform can be booted with SGX enabled and without performing registration, and it can use the previously generated platform keys.

Microcode also generates a data structure called the platform manifest that contains information about all CPU packages in the platform, signatures from all CPU packages using the PRK private keys, and the platform keys encrypted with the Registration Server's Encryption Key (RSEK).

Microcode can also share the platform keys with an added package once it has been certified by the registration authority service.

2.1.2. BIOS

BIOS determines the boot flow based on the state of the available key blobs and on the security version number (SVN) of the microcode patch present at platform reset. BIOS selects the value of SVN to use in the boot flow. It can be equal to or lower than the value of the actual SVN of the loaded microcode patch, but it cannot be higher. BIOS also collects the information about the registration authority service (SgxRegistrationServerID) and loads the microcode patch.

Once a platform has been established, BIOS stores the generated key blobs and uses them on subsequent boots. This way the platform can boot with previously generated platform keys.

When the Add Package boot flow is selected (see Add Package (Replace Package) for more details), BIOS generates the add request structure. It also delivers the resulting platform membership certificate to the microcode to complete the Add Package flow.

BIOS provides and consumes information between the OS software layer using the UEFI variables that the BIOS Multi-Package UEFI Variables appendix describes.

2.1.3. Platform Software

The multi-package platform software essentially acts as network transport layer between the BIOS and the registration authority service. It also acts as management software for the various boot flows. The multi-package platform software runs in the OS context. It can either run as an executable that starts up on every boot, or it can be a set of tools used by datacenter and CSP managers. The way the software is used depends on the SGX server environment.

The bulk of this document describes the platform software delivered by Intel.

2.1.4. Registration Authority Service

The registration authority service has three main functions:

1. Validate platform manifests for the Initial Platform Establishment and TCB Recovery boot flows.
2. Process add requests and generate platform membership certificates.
3. Generate Platform Certification Key (PCK) certificates for registered platforms.

Each platform instance is related to only one registration authority service. If a platform needs to register with a different registration authority service, new platform keys need to be generated.

When the registration authority service receives a platform manifest for a new platform, it makes sure all PRK signatures from all participating CPU packages come from valid SGX-enabled CPU packages in good standing. The registration authority service contains a database of all PRK certificates generated by Intel. If a PRK certificate is not found or the signature check fails, the registration authority service sends an error back to the platform software/BIOS, and that platform is not recorded in the service database. If any of the PRK certificates are revoked, the service does not record the platform and responds with the appropriate error. The service also checks that all platform CPU packages are compatible with one another and that the CPU package topology is valid.

If the platform is not recorded in the registration service, it cannot generate PCK certificates for that platform, so it prevents the platform from running remote attestation. Note that the platform still has SGX enabled since the microcode and BIOS boot flow succeeded. It just cannot run remote attestation. If it succeeds, the platform is recorded in the registration service database and PCK certificate generation is possible.

When the registration authority service receives an add request, it verifies that a PRK certificate exists for the package's PRID and CPUSVN. If it exists, it verifies that it is not revoked. Then it verifies that the package is consistent with the existing platform. If all these checks pass, the registration authority service constructs the platform membership certificate and sends it back in the response. The registration authority service does not check any signature on the add request since the add request is generated by BIOS, which does not have access to the PRK private key. The platform manifest certificate contains the PRK public key and the platform information all signed by the Registration Server's Authorization Key (RSAK). When microcode completes Add Package flow, it uses the PRK public key to verify a PRK signature from the new package before providing it with the platform keys.

Software needs to communicate the platform manifests and the add requests to the registration authority service and process the responses. Responses to platform manifests contain status information and the platform's Platform Provisioning ID (PPID). Responses to add requests contain status information and may also contain platform membership certificates.

To support multi-package platform remote attestation for SGX, the registration authority service needs to be capable of generating PCK certificates. Once the encrypted platform keys for multi-package platforms are delivered to the registration authority service, the registration authority service uses its knowledge of the Provisioning Certification Enclave (PCE) identity to generate the PCK public key.

Intel hosts a registration authority service called the Intel® SGX Registration Service (Intel® SGX RS). For more information on the Intel® SGX RS, see
https://api.portal.trustedservices.intel.com/

2.2. SGX Multi-Package States

The SGX platform keys for a multi-package platform have a lifecycle depending on the state of the boot flow. This section provides a brief overview of each state.

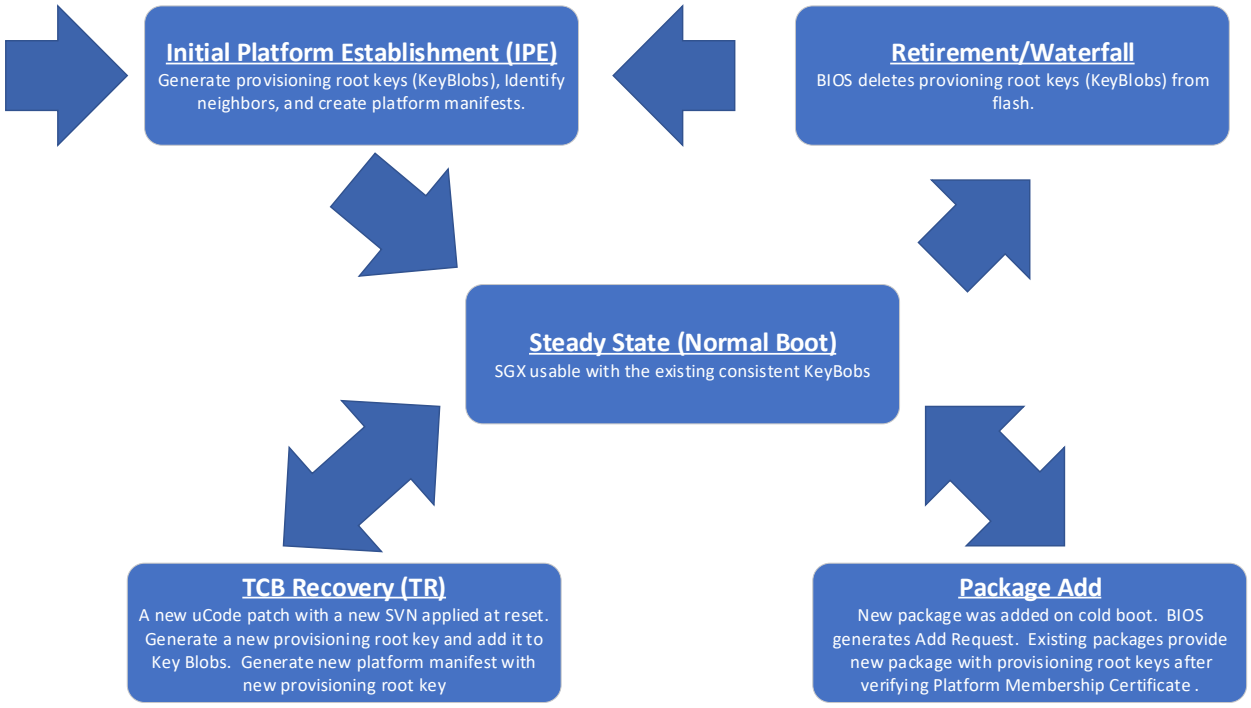

Figure2: Platform Key Life Cycle

The BIOS uses data structures called key blobs to determine the state of the platform. The key blobs contain the platform keys that are negotiated for the platform. Each CPU package needs to have a consistent key blob structure to boot successfully with SGX enabled. BIOS stores the key blobs in persistent storage (FLASH).

2.2.1. Initial Platform Establishment (IPE)

During the Initial Platform Establishment boot flow, the BIOS checks the key blobs for each CPU package to verify that they are all consistent with each other and the platform. BIOS deletes the inconsistent key blobs. If there are no key blobs or none of the key blobs are consistent with the current platform, BIOS provides no key blobs to microcode. Microcode sees that there are no key blobs provided and generates new platform keys. If BIOS selects to boot with microcode patch SVN that is equal to or less than the actual value of the SVN of the microcode patch loaded at reset, microcode generates platform keys for the provided SVN and all lower SVNs. The new platform keys are randomly generated for this platform instance. Each CPU package uses its HW key to encrypt the shared platform keys and generate a key blob. BIOS stores the key blobs in flash for future boots. Microcode also generates a new platform manifest for the new platform instance. BIOS provides the new platform manifest to software via the SGXRegistrationServerRequest UEFI variable and indicates that a registration flow is required using the SGXRegistrationStatus UEFI variable. The registration authority service must evaluate the new platform manifest before it can generate any PCK Certificates for the new platform.

2.2.2. Normal

During normal boot, the BIOS checks the key blobs for each CPU package to verify they are all consistent with each other and the platform. If the key blobs indicate that they contain the key for the SVN of the microcode patch loaded at reset, BIOS provides them to microcode. Microcode verifies the key blobs for each package using the CPU packages' respective HW key. If microcode successfully verifies the key blobs, the registration authority service has nothing to verify, and it does not request the BIOS/software to do anything. In this case, the platform uses the platform keys stored in the key blobs.

2.2.2.1. Microcode Downgrade

This is a special case of a Normal boot flow. The BIOS checks the key blobs for each CPU package to verify that they are all consistent with each other and the platform. In this case, the SVN of the microcode patch that is loaded at reset is lower than the highest platform key in the key blob. This is supported, and the platform boots with the existing platform keys. Microcode does not have access to the platform keys that are higher than the SVN of the microcode patch that is loaded at reset.

2.2.2.2. Removing CPU Packages

Removing CPU packages has no impact to the existing CPU packages or the platform keys. This is a Normal boot flow.

2.2.2.3. Re-arranging CPU Packages

If the same set of CPU packages is in different sockets, BIOS and microcode recognize that the same CPU packages make up the platform and the platform keys are not recreated. Rearranging the CPU packages is considered to be a Normal boot flow.

2.2.3. TCB Recovery (TR/TCB-R)

During a TCB Recovery boot, the BIOS checks the key blobs to verify that they are all consistent with each other and the platform. In this flow, the BIOS detects that the platform loaded a microcode patch

at reset with a higher SVN than the highest platform key in any of the key blobs. BIOS requests microcode boot at this new SVN. Microcode sees that its SVN is greater than the highest SVN key in the key blobs and generates a new platform key for that SVN (and any platform keys between the highest in the key blob and the booting SVN). These platform key(s) are shared with all the packages on the platform and added to each of key blobs. Microcode also creates a new platform manifest that has the new platform key(s) added. BIOS provides the platform manifest with the new platform key(s) to software via the SGXRegistrationServerRequest UEFI variable and indicates that a registration flow is required using the SGXRegistrationStatus UEFI variable. The registration authority service must evaluate the new platform manifest before it can generate a PCK Certificate(s) at the new SVN level(s).

2.2.4. Add Package (Replace Package)

During an Add Package flow, BIOS checks the key blobs to verify that they are consistent with each other and the platform. In this flow, BIOS detects that at least one of the packages does not have a key blob or that its key blob is not consistent with the platform (BIOS deletes the inconsistent key blobs). BIOS then generates an add package request and disables SGX. Since SGX is disabled, the microcode does not evaluate the existing key blobs. BIOS provides the add package structure with the identity of the new package(s) to the platform software via the SGXRegistrationServerRequest UEFI variable and indicates that an Add Package flow is required using the SGXRegistrationStatus UEFI variable. The registration authority must evaluate the add package structure before the microcode can share the existing platform keys to the new CPU package. The registration authority service evaluates the add package request and generates a signed platform membership certificate(s) verifying that the CPU package(s) is authenticate and in good standing. The platform membership certificate includes the PRK public key of the CPU package, and it is signed by the registration authority service's RSAK.

The platform software then provides the platform membership certificate back to BIOS via the SGXRegistrationServerResponse UEFI variable. The platform must then be rebooted. Upon reboot, BIOS sees that a package is missing a key blob, and instead of disabling SGX, it provides the existing key blobs and the platform membership certificate(s) for the new CPU package(s) to microcode loader. Microcode evaluates the key blobs and the platform membership certificate(s). If everything verifies properly, the new CPU package is provided the platform keys and microcode creates a key blob for the new CPU package. The platform then boots with SGX enabled.

If a you do not want to support the Add Package flow, you can perform an SGX factory reset to erase the key blobs and force an Initial Platform Establishment flow when a new package is added or replaced. You can also use the BIOS User Interface (UI) 'SGX APB Support' setting to disable the Add Package boot flow. With this disabled in BIOS, BIOS automatically enters the IPE flow instead of the Add Package flow when BIOS detects a new CPU package.

See the add package API supported by the Intel® Registration Service at https://api.portal.trustedservices.intel.com/.

2.2.5. Retirement/Waterfall

The same collection of CPU packages can conduct IPE flows multiple times on the same physical platform. This is useful for waterfalling/recycling. Each IPE flow generates new platform keys and

creates a new 'platform instance'. Each platform instance has its own platform keys and platform manifests.

There is a BIOS setting that allows the platform owner to perform a factory reset on the SGX state and settings. This deletes all the key blobs for that platform and forces a new IPE flow.

2.3. UEFI Variables and Tboot

When Intel® TXT is enabled and using TBOOT, the TBOOT loader will launch the Linux kernel with the 'noefi' parameter. This will disable the UEFI Runtime Services in the OS. TBOOT does this because any component that runs before TBOOT executes the GETSEC[SENTER] instruction is not in the TCB (BIOS, MBR, GRUB bootloader, etc). Any measurements made by these components are not trusted. Since TXT does not measure the UEFI Runtime Services, TBOOT boots the Linux kernel with the 'noefi' parameter.

As a result of the 'noefi' parameter, the UEFI variables used to enable the flows described in SGX Multi-Package States are not available. This means the platform cannot complete the IPE, TR and Add Package flows when TBOOT is run.

When the platform needs to perform the IPE, TR and Add Package flows with TBOOT support, the kernel needs to boot without the 'noefi' parameter, perform the operation, and reboot with TBOOT and the 'noefi' parameter.

The impact of this restriction can be limited:

1. IPE flow:
   - The TBOOT provisioning stage requires a non-TBOOT OS flow and a reboot.
   - The IPE provision stage can happen during the same OS boot flow by reading the Platform Manifest from the UEFI variable

2. The Add Package
   - Requires a boot to the OS with SGX disabled.
   - During this boot, TBOOT can be disabled or boot without 'noefi'.
   - Add Package requires a reboot to complete.
   - Next boot, TBOOT can be enabled and linux booted with 'noefi'.

3. TCB Recovery
   - The platform needs to reboot to perform a TCB Recovery since it requires a new uCode patch to be loaded at reset.
   - *Without TBOOT, an additional reboot would not be required.*
   - With TBOOT, TCB Recovery requires access to the UEFI variables.
     - After the uCode has been updated, you need to boot with out TBOOT or boot without 'noefi' and collect the Platform Manifest from the UEFI variable.
     - *Another reboot is required to re-enable TBOOT.*
   - TCB Recoveries are limited to 2 times a year.

2.4. SGX Multi-Package Registration Modes

Intel's registration authority service (Intel® SGX Registration Service) supports two mechanisms for registering a platform. This section describes these methods.

Note that SGX can be enabled and available even when the platform is not registered. However,the platform cannot perform remote attestation until it is successfully registered.

2.4.1. Direct Registration

In direct registration, the platform manifests are sent directly to the Intel® SGX Registration Service's register API (https://api.portal.trustedservices.intel.com/). When you use this interface to register a platform, you implicitly give permission to store the encrypted platform keys in the Registration Service's database. When the Registration Service later generates a PCK Certificate, it sets the 'Cached Keys' OID to 'true' to indicate that the Registration Service is caching the encrypted platform keys. You may use the 'Indirect Registration' method later, but it does not affect the value of the 'Cached Keys' flag. Once the 'Cached Keys' flag is set in the Intel® SGX Registration Service, it cannot be reversed for that platform instance.

Registering the platform using direct registration allows requesting a PCK Certificate with just the platform's Platform Provisioning ID (PPID) of the platform. The PPID is derived from the platform keys and both the Registration Service and the Intel® SGX PCE can generate the PPID. The PCK requester does not need to store the platform manifest once it is registered with the Registration Service.

2.4.2. Indirect Registration

In indirect registration, the platform manifests are not sent directly to the Intel® SGX Registration Service. Instead, the PCK Certificate is generated using the platform manifest as input, and the Registration Service only uses encrypted platform keys of the platform manifest long enough to generate the PCK Certificate. It does not store them permanently. It only stores platform metadata. The PCK Certificates that is generated this way sets the 'Cached Keys' OID to 'false' to indicate that the Registration Service is not caching the encrypted platform keys. If a platform manifest from the same platform instance is later sent to the Registration Service's direct registration API, the Registration Service returns an error to maintain the consistency of the 'Cached Keys' flag policy. Once the 'Cached Keys' flag is set in the Intel® SGX Registration Service, it cannot be reversed for that platform instance.

When you use this method, PCK Certificates cannot be requested using the PPID since the Registration Service does not have the platform keys required to generate PPID. Instead, the platform manifest must be provided to generate the PCK Certificates. You must maintain a copy of the platform manifests.

2.4.2.1. Intel® SGX Provisioning Certification Service (Intel® PCS)

The Intel® SGX Provisioning Certification Service offers APIs for retrieving PCK Certificates. This service was introduced to support third party attestation for single package platforms. It has been expanded to include support for multi-package platforms. See https://api.portal.trustedservices.intel.com for more information.

The new APIs support indirect registration by allowing the platform owner to request PCK Certificates using the platform manifest for multi-package platforms.

2.4.3. Registration Environments

The registration environment, where the SGX platform runs, depends on the SGX server deployment flow that you use. In some cases, the platforms are released as an end-user system and registration occurs in the customer environment with access to the internet. The environments are mostly used for server workstations and some small enterprises. For others, there are distinctly different operating environments for the provisioning phase and for the run-time phase of the platform lifetime. For both environments, access to the internet is typically controlled, and therefore access to the registration authority service is not available to the SGX platform.

2.4.3.1. Single-Stage Registration

Typically, the registration authority service (such as the Intel® SGX Registration Service) is hosted on the Internet. For platforms that have access to the Internet, the platform can simply register directly with registration when it first boots. There is no need to proxy the registration, and registration can happen in a single stage. In the single-stage environment, a software agent supporting server registration flows runs automatically at boot time. It can check the BIOS UEFI interface for registration request data, send it to the registration authority service and then process the responses returned by the registration authority service accordingly.

Single-stage registration can satisfy small enterprises or server workstations that want a simpler registration flow and do not want to make large investments into registration infrastructure. Platform owners can also use the single-stage environment to support a simple validation environment. See using the Multi-Package Registration Agent (MPA) for more information on supporting this environment.

2.4.3.2. Dual-Stage Registration

In the dual-stage environment, the platform requiring registration does not have access to the Internet (and subsequently an externally hosted registration authority service). Instead, there is a proxy that performs communication with the registration authority service on its behalf. The platform can be moved to the run-time environment once the proxy successfully delivers the platform manifest to the registration service either directly or indirectly.

Currently, only the platform manifest retrieval is supported in the dual-stage environment since it does not require a response from the registration authority service to complete the flow. The Add Package flow requires a response from the registration authority service (the platform membership certificate) to complete the flow and enable SGX. Add Package flows may not be suitable for the dual-stage environment unless you can get immediate responses from the registration authority service or the platform can be provisioned with the platform membership certificate later.

See PCK Cert ID Retrieval Tool for more information on supporting this environment.

3. Multi-Package Registration Platform Software Tools

The Intel® SGX Data Center Attestation Primitives (DCAP) release has added new tools and extended existing tools to support multi-package platform registration. These tools use the Multi-Package Registration Libraries described in a later section. These tools are created to quickly support multi-package platform registration in their datacenter or cloud service provider environments. Customers can modify or design their own tools using the multi-package libraries. These DCAP tools and libraries are released in binary format for Linux* (https://download.01.org/intel-sgx/latest/dcap-latest/linux/tools/) and for Windows* (https://download.01.org/intel-sgx/latest/dcap-latest/windows/tools/). For the open source for these tools, see

https://github.com/intel/SGXDataCenterAttestationPrimitives/tree/master/tools/SGXPlatformRegistration.

3.1. Multi-Package Registration Agent (MPA)

The Multi-Package Registration Agent (MPA) is an executable that launches as a daemon/service automatically after boot. Upon OS boot, it examines the state of the BIOS Multi-Package UEFI Variables to determine if there is any registration action to take. This tool requires Internet access. It also requires access to the UEFI SGX variable interface, which is only available to bare-metal platforms or to the Host VM. So, this tool may not be suitable for datacenters or Cloud Service Providers (CSP's) that have a separate provisioning environment and customer workload environment where customer workloads run in guest VMs and platform provisioning happens without an Internet connection. This tool may be most useful for validation purposes or workstations with Internet connection (Single Stage Registration). This tool is available as open source, and you can use it as a reference for communicating to both the SGX MP UEFI variables and the Intel® SGX Registration Service (SGX MP Network).

BIOS resets the SgxRegistrationStatus.Status.SgxRegistrationComplete to 0 when a registration request is available for processing. BIOS clears this bit when there is a successful IPE boot flow, TCB Recovery boot flow, and the Add Package boot flow. When this flag is 0, the BIOS puts data in the SgxRegistrationServerRequest UEFI variable that needs processing. For the IPE boot flow and the TCB Recovery boot flow, the SGXRegistrationServerRequest UEFI variable contains the platform manifest structure. For the Add Package boot flow, the SGXRegistrationServerRequest UEFI variable contains the add package structure. The MPA treats the platform manifest and add package structure as blobs and only parses the header to determine which of the data structures BIOS provided.

The MPA first checks if BIOS has reported any errors by scaning the SgxRegistrationStatus.Errorcode. Then it checks the SgxRegistrationStatus.Status.SgxRegistrationComplete flag to see if it has been reset to 0. If both the SgxRegistrationStatus.Errorcode is 'success' and the SgxRegistrationStatus.Status.SgxRegistrationComplete flag is 0, the MPA reads the SgxRegistrationServerRequest UEFI variable and parses the data structure header to determine what type of data structure needs processing (currently, only platform manifest and add package structures are supported). The MPA sends the data structure to the Intel® SGX Registration Service. When the MPA successfully communicates to the Registration Service or it encounters a non-recoverable error, it sets the SgxRegistrationStatus.Status.SgxRegistrationComplete to 1 to indicate that BIOS does not need to provide the data structure again on a subsequent boot. If the MPA receives an error that can be recoverable, it does not set the

SgxRegistrationStatus.Status.SgxRegistrationComplete to 1 to indicate that BIOS should provide the same structure on a subsequent boot so the MPA can retry processing the structure again.

The MPA sends the requests to the URL provided by the SgxRegistrationConfiguration UEFI variable using the API defined by the Intel® SGX Registration Service. Once complete, the MPA stops its service/daemon.

3.1.1. Platform Manifest Handling

BIOS provides platform manifests on IPE boot flows and TR boot flows. It also provides them in Normal boot flow when the SgxRegistrationComplete flag is 0 (indicating a retry). There is no response data from the Registration Service after sending the platform manifest. The MPA treats some server response codes as terminal when no amount of retries fixes the problem. In this case, the MPA sets the SgxRegistrationComplete flag to 1 so that BIOS does not provide the same platform manifest on a subsequent boot.

The list of response codes from the Intel® SGX Registration Service response codes that terminate the registration process:

- 201 - Created (The platform instance is created or updated in the Registration Service database)
- 400 - Invalid Platform Manifest (Client should not repeat the request without modifications)
    - ErrorCode = InvalidRequestSyntax
    - ErrorCode = InvalidRegistrationServer
    - ErrorCode = InvalidOrRevokedPackage
    - ErrorCode = PackageNotFound
    - ErrorCode = IncompatiblePackage
    - ErrorCode = InvalidPlatformManifest
    - ErrorCode = CachedKeyPolicyViolation

The list of response codes from the Intel® SGX Registration Service response codes that do not terminate the registration process:

- 401 - Failed to authenticate or authorize the request
- 415 - MIME type specified in the request is not supported by the server.
- 500 - Internal server error occurred
- 503 - Server is currently unable to process the request.

There are also internal MPA errors that are considered terminal. See section MPA Error Codes for a list of these errors.

3.1.2. Add Package Handling

Handling an Add Package flow is more complicated than platform manifests. The MPA expects a response from the Registration Service. This response contains the platform membership certificate(s) for the new CPU package(s). If the response does not contain the platform membership certificate, the MPA reports a failure. This flow requires internet connection to directly interact with the Registration Service. The Intel® SGX Registration Service requires that the user subscribes for an API key to use the add package API.

BIOS provides an add package structure in the Add Package boot flow. It also provides them in a Normal boot flow when the SgxRegistrationComplete flag is 0 indicating a retry. The MPA treats

some server response codes as terminal, and retries do not fixe the problem. In this case, the MPA sets the SgxRegistrationComplete flag to 1 so that BIOS does not provide the same add package structure on a subsequent boot.

The list of Intel® SGX Registration Service response codes that terminates the add package process:

- 200 - OK
- 400 - Invalid Platform Manifest (Client should not repeat the request without modifications)
    - ErrorCode = InvalidRequestSyntax
    - ErrorCode = PlatformNotFound
    - ErrorCode = InvalidOrRevokedPackage
    - ErrorCode = PackageNotFound
    - ErrorCode = InvalidAddRequest

The list of Intel® SGX Registration Service response codes that does not terminate the add package process:

- 401 - Failed to authenticate or authorize the request
- 415 - MIME type specified in the request is not supported by the server.
- 500 - Internal server error occurred
- 503 - Server is currently unable to process the request.

There are also internal MPA errors that are considered terminal. See section MPA Error Codes for a list of these errors.

3.1.3. Configuration

The MPA can be configured with the following settings. Linux configurations are provided in a configuration file. Windows configurations are provided by registry keys.

- **Subscription Key** – Only required for Add Package flows. Provided by the Intel® SGX Registration Service upon subscribing.
    - Config Location:
        - Linux: /etc/mpa_registration.conf
        - Windows:
            - KEY_LOCAL_MACHINE\SOFTWARE\Intel\SGX_RA\RASubscriptionKey
            - Add the following String key name "token"
                - <64byte-hex-value>
                - e.g: 7a963d696ff94b7d82df4cbe924b1574

- **Proxy Setting** – Modify the proxy settings used by the MPA
    - Values:

Proxy Type	Linux Value	Windows Value
Use the configuration in your operating system (**default value**)	default	0
Direct access to the internet	direct	1
Set the proxy URL directly - Supports authenticated proxy.	manual	2

- Proxy URL uses standard format: user:password@proxy:port		

- Config Location:
  - Linux: /etc/mpa_registration.conf
  - Windows:
    - HKEY_LOCAL_MACHINE\SOFTWARE\Intel\SGX_RA\RAProxy
    - Add the following DWORD key name "type" and set it to one of the Windows Values from the table above.
      - If you set it to '2' (manual), then also add the String key "url" and set the URL in the standard format described in the table above.

- **Log Level**
  - Values:

Logging Level	Linux Value	Windows Value
None	none	0
Functional	func	1
Error (**default value**)	error	2
Verbose	info	3

  - Config Location:
    - Linux: /etc/mpa_registration.conf
    - Windows:
      - HKEY_LOCAL_MACHINE\SOFTWARE\Intel\SGX_RA\RALog
      - Add the the DWORD key "level" and set it to one of the Windows Values from the table above.
  - Logging output:
    - Linux - /var/log/mpa_registration.log
    - Windows –
      - C:\Windows\System32\Winevt\Logs\Application.evtx
      - Open Application.evtx (from another platform with Windows GUI)
      - Look for "IntelMPAService" records in the "source" column
- **UEFI path**
  - Values:
    - `/sys/firmware/efi/efivars` **(Default)**
  - Config Location:
    - Linux: /etc/mpa_registration.conf
    - Windows: Not available for Windows

3.1.3.1. BIOS Registration Authority Service Configuration

The BIOS uses a default URL for the Intel® Registration Service in the SgxRegistrationServerConfiguration UEFI variable. The platform owner can modify the registration service configuration using the SgxRegistrationServerConfiguration UEFI variable but only when SGX is disabled.

The BIOS also has a BIOS User Interface (UI) ('SGX Auto MP Registration') setting that allows the platform owner to enable/disable the MPA from running automatically at OS boot. By default, the MPA does not automatically run at boot.

3.1.4. Error Codes

The MPA writes an error code to the ErrorCode field of the SgxRegistrationServerStatus UEFI variable when it completes. These are the possible ErrorCode values produced by the MPA (the MPA error codes always have the MSBit of the ErrorCode field):

MPA_SUCCESS (0x00)

Completed without any errors

MPA_AG_UNEXPECTED_ERROR (0x80)

Unexpected internal error

MPA_AG_OUT_OF_MEMORY (0x81)

Out-of-memory error

MPA_AG_NETWORK_ERROR (0x82)

Proxy detection or network communication error

MPA_AG_INVALID_PARAMETER (0x83)

Invalid parameter in input

MPA_AG_INTERNAL_SERVER_ERROR (0x84)

Internal server error occurred

MPA_AG_SERVER_TIMEOUT (0x85)

Server communication timeout

MPA_AG_BIOS_PROTOCOL_ERROR (0x86)

BIOS UEFI protocol error

MPA_AG_UNAUTHORIZED_ERROR (0x87)

The client is unauthorized to access the registration server

MPA_RS_INVALID_REQUEST_SYNTAX (0xA0)

Server could not understand request due to malformed syntax

MPA_RS_PM_INVALID_REGISTRATION_SERVER (0xA1)

Server rejected request because it is intended for different registration server (Registration Server Authentication Key (RSAK) mismatch)

MPA_RS_INVALID_OR_REVOKED_PACKAGE (0xA2)

Server rejected request due to invalid or revoked CPU package

MPA_RS_PACKAGE_NOT_FOUND (0xA3)

Server could not recognize at least one of the CPU packages

MPA_RS_PM_INCOMPATIBLE_PACKAGE (0xA4)

Server detected at least one of the CPU packages is incompatible with rest of the CPU packages on the platform

MPA_RS_PM_INVALID_PLATFORM_MANIFEST (0xA5)

Server could not process the platform manifest structure

MPA_RS_AD_PLATFORM_NOT_FOUND (0xA6)

Server rejected add package request because the platform has not been registered

MPA_RS_AD_INVALID_ADD_REQUEST (0xA7)

Server could not process the add package structure

MPA_RS_UNKOWN_ERROR (0xA8)

Server rejected request for unknown reason (Probably means MPA needs to be updated with newly defined server response errors)

3.2. PCK Cert ID Retrieval Tool

The PCK Cert ID Retrieval Tool is an executable that collects the platform information that is necessary for retrieving PCK Certs from the Intel® SGX Provisioning Certificate Service (PCS). The DCAP releases for single-package platforms already include the PCK Cert ID Retrieval Tool, but it has been expanded to include support for multi-package platforms.

The main expansion to the tool is its ability to retrieve the platform manifest from the SgxRegistrationServerRequest UEFI variable. By retrieving the platform manifest, this tool supports Indirection Registration better by allowing to store the platform manifest and use it later for retrieving PCK Certificates. The registration authority service does not need to persistently store the platform keys when the platform owner maintains a copy of the platform manifest for retrieving PCK Certificates.

Unlike the PCK Cert ID Retrieval Tool support for single package platforms, the tool needs to run in the host VM or on baremetal to get access to the UEFI variables. The SGX UEFI variables are not exposed to guest VMs. This tool is expected to run in the platform deployment environment when a platform is assembled or when a TCB Recovery event occurs requiring a new microcode patch applied at reset. It is not expected to run in a guest VM. For nore information, see Dual Stage Registration.

By default, this tool loads enclaves to retrieve the platform identification data. This is required for single package platforms. For multi-package platforms, it can be configured to get limited platform identification information without loading any enclaves. For more information, see Platform ID Without Enclave Loading.

Unlike the MPA, the PCK Cert ID Retrieval Tool does not require internet connection. The PCK Cert ID Retrieval Tool can output the platform ID information to a file, or it can be configured to send the data to the DCAP's reference Caching Service (See PCK Certification Caching Service (PCCS)for more information).

The tool links with the SGX Multi-Package UEFI Variables Access Library.

3.2.1. Platform Manifest Handling

BIOS provides platform manifests on IPE boot flows and TR boot flows. It also provides them in Normal boot flow when the SgxRegistrationComplete flag is 0 indicating a retry. This tool always checks the SgxRegistrationServerRequest UEFI variable directly and ignores the value of the SgxRegistrationComplete flag. If it finds the platform manifest, it adds it to its platform ID output.

Although this tool does not check the SgxRegstrationComplete flag to be 0 before reading the SgxRegistrationServer Request, it writes 1 to the SgxRegstrationComplete flag if it successfully processes the SgxRegistrationServerRequest UEFI variable.

3.2.2. Add Package Handling

The PCK Cert ID Retrieval Tool does not support Add Package flows. If it encounters an add package structure in the SgxRegistrationServerRequest UEFI variable, it generates an error. You should use other means to support Add Package flows.

3.2.3. Configuration

The PCK Cert ID Retrieval Tool has two major operating modes:

1. Output platform ID information to a comma-separated values (CSV) file
2. Send platform ID information to the reference design PCCS on the local network
    a. Network configuration can be specified by an XML file.
    b. Network configuration can be specified on the command line.

Valid Command line Parameters:

Command Line Parameter	Description
-f <filename>	Output the platform ID information to the "filename". The output is a comma-separated value (CSV) file with base-16 encoding.
-url <cache_server_address>	Reference PCCS's URL. *Only needed when using the network to communicate to the PCCS.*
-user_token <token_string>	User token to access the PCCS. *Only needed when using the network to communicate to the PCCS.*
-proxy_type <proxy_type>	Proxy setting when accessing the PCCS. *Only needed when using the network to communicate to the PCCS.*
-proxy_url <proxy_server_address>	Proxy server address. *Only needed when using the network to communicate to the PCCS.*
-use_secure_cert <[true | false]>	Accept secure/insecure https cert.

	Only needed when using the network to communicate to the PCCS. *Default value is true*
-platform_id <platform_id>	When provided, no enclaves are loaded. You need to provide a unique platform id that can be used to identify the platform at run-time.
-?	Show command help
-h	Show command help
-help	Show command help

3.2.3.1. Outputting to a CSV File

The PCK Cert ID Retrieval tool can output the platform identification information using the '-f' command line parameter.

Note: All integer fields are in little-endian format.

```
<EncryptedPPID (384 byte array)>,
<PCE_ID (16 bit integer)>,
<CPUSVN (16 byte array)>,
<PCE ISVSVN (16 bit integer)>,
<PLATFORM_ID (16 byte array)>,
<PLATFORM_MANIFEST (variable byte array)>
```

This output is the same as it was for the single-package platforms with the addition of the platform manifest. The platform manifest is an optional output to the CSV file, and it is not present on single-package platforms or on multi-package platform boot flows that do not provide a platform manifest.

Because the EncryptedPPID cannot be used as a platform identifier due the randomness entropy of the encryption algorithm, the PLATFORM_ID is used as platform identifier for the platform. If the -platform_id parameter is not provided, the DCAP Quoting Enclave (QE) generates a QE_ID as the platform_id. If the PLATFORM_ID is used as input, the Quote Provider Library must use the same PLATFORM_ID when retrieving the PCK Certificates from the PCCS.

3.2.3.2. Outputting to the Reference Provisioning Certification Caching Service (PCCS)

The PCK Cert ID Retrieval Tool can be configured to send the platform ID information to the Intel reference PCCS. The PCCS exposes a REST API for accepting the platform ID information. See PCK Certification Caching Service (PCCS) for more information. The network configuration can be provided on the command line or by entering the information into an xml configuration file (network_configuration.conf). The command line configurations take precedence. This is convenient if the multi-package platform can reach the local network in the provisioning environment.

3.2.3.3. Platform Identity Without Enclave Loading

The PCK Cert ID Retrieval Tool may run in a constrained environment during platform provisioning. It may not have OS support for loading enclaves or all of the DCAP software packages installed, or you may have a specific need to choose/supply their own platform ID. To support these

environments better, the PCK Cert ID Retrieval Tool supports a mode that only retrieves the platform manifest and does not retrieve the platform ID information that requires loading any enclaves. The output of the CSV file or the data sent to the PCCS only contains the PCEID, the platform manifest and a user-supplied run-time platform identifier. The run-time platform identifier replaces the QE_ID in the platform ID information. This platform identifier can be used during run-time for requesting PCK Certificates because the platform manifest is not available to guest VMs. Providing the '-platform_id' command line parameter selects this mode of operation

3.3. Multi-Package Management Tool

The DCAP release for multi-package platforms also includes a management tool to provide status and configuration to the platform from the OS software. Like the PCK Cert ID Retrieval Tool, it links with the SGX Multi-Package UEFI Variables Access Library.

Command Line Parameter	Description
-get_platform_manifest <file_name>	Reads the SgxRegistrationServerRequest UEFI variable to get the platform manifest when present. It outputs the platform manifest in binary form to the file specified in <file_name>.
-get_key_blobs <file_name>	Reads SgxRegistrationPackageInfo UEFI variable to get the key blobs when present. It outputs the key blobs in binary form to the file specified in <file_name>.
-set_server_info <file_name> <hex_flags> <URL>	Used to change the registration authority service. <file_name> contains the self-signed SgxRegistrationServerID from the registration authority service. <hex_flags> indicates the value of 'Flags' in SgxRegistrationConfiguration UEFI variable. <URL> the URL of the registration authority service.
-get_registration_status	Reports whether it is completed or in progress. This is the reporting the value of the 'SgxRegistrationComplete' flag in the SgxRegistrationStatus UEFI variable.
-get_last_registration_error_code	Reports the registration error code. It is the value of the 'Status.ErrorCode' field in the SgxRegistrationStatus UEFI variable. The error code can be from the BIOS or from the MPA.
-get_sgx_status	Reports the status of SGX.
-v	Produce verbose output.
-h	Show command help.

3.3.1. Changing the Registration Authority Service

BIOS provides a default registration authority service configuration. For the Intel reference BIOS, the default registration authority service is the Intel® SGX Registration Service. The SGX multi-package platforms support modifying the registration authority service with the SgxRegistrationConfiguration UEFI variable. This variable becomes writable only when the platform owner disables SGX via the BIOS configuration settings. The registration authority service does not need to generate a self-signed SgxRegistrationServerID structure. This combined with the registration authority service URL is the SgxRegistrationServerInfo structure. The platform owner then writes the SgxRegistrationServerInfo to the SgxRegistrationConfiguration UEFI variable.

On the next SGX enabled boot, key blobs generated for a different registration authority service are deleted and BIOS forces an Initial Platform Establishment boot flow. This results in new key blobs and a new platform manifest.

3.3.2. Handling Key Blobs

Each CPU package on the platform has a key blob when SGX is enabled on a multi-package platform. BIOS provides a mechanism for retrieving the key blobs. Platform owners may want to maintain a copy of the key blobs in case they need to be restored after they are deleted from BIOS persistent store (e.g. the FLASH was erased or SGX was reset). The SgrRegistrationPackageInfo UEFI variable provides the key blobs. By default, BIOS does not present the key blobs to the software. The platform owner needs to 'opt-in' using the BIOS configuration setting ('SGX Package Info In-band Access') before BIOS provides the key blobs.

3.3.3. Registration Error Codes

The error codes written to the 'Status.ErrorCode' field in the SgxRegistrationStatus UEFI variable can come from one of two sources. BIOS writes to this field when an error occurrs during a boot flow. The MSBit of the ErrorCode is 0 when generated by BIOS. The software can also use this field to report any errors in processing the data from BIOS. The MSBit of the ErrorCode is 1 when generated by software. Software should not overwrite the ErrorCode if BIOS writes a non-zero value.

The software error codes generated by the MPA are defined in MPA Error Codes. The BIOS error codes are defined as follows:

RS_PREMEM_OTHER	0x10
RS_PREMEM_NOMEM	0x11
RS_PREMEM_SYS_NOT_CAPABLE	0x12
RS_PREMEM_NO_VALID_PRMRR	0x13
RS_PREMEM_HW_NOT_CAPABLE	0x14
RS_PREMEM_TME_DISABLED	0x15
RS_PREMEM_SGX_DISABLED	0x16
RS_PREMEM_INVALID_PRRMR_SIZE	0x17

RS_PREMEM_PRMRR_NOT_SECURED	0x18
RS_PREMEM_MEM_TOPOLOGY_ERR	0x19
RS_POSTMEM_OTHER	0x20
RS_POSTMEM_NOMEM	0x21
RS_POSTMEM_SYSHOST_NOTFOUND	0x22
RS_POSTMEM_MMAP_HOST_NOTFOUND	0x23
RS_POSTMEM_VSPPI_NOTFOUND	0x24
RS_POSTMEM_MRCHCSPPI_NOTFOUND	0x25
RS_POSTMEM_SVN_ERR	0x26
RS_POSTMEM_REGVARS_ERR	0x27
RS_POSTMEM_KEYBLOBS_RES_ERR	0x28
RS_POSTMEM_PRID_UNLOCK_ERR	0x29
RS_POSTMEM_DETERMINE_BOOT_ERR	0x2A
RS_POSTMEM_FIRSTBOOT_ERR	0x2B
RS_POSTMEM_WARMRESET_ERR	0x2C
RS_LATEINIT_OTHER	0x30
RS_LATEINIT_TRIGCALLBACK_ERR	0x31
RS_LATEINIT_HOBLIST_NOTFOUND	0x32
RS_LATEINIT_MPSVC_ERR	0x33
RS_LATEINIT_INITDATAHOB_RES	0x34
RS_LATEINIT_UPDTCAPAB_ERR	0x35
RS_LATEINIT_UPDTPRMRR_ERR	0x36
RS_LATEINIT_CRDIMM_ERR	0x37
RS_LATEINIT_UPDTLEWR_ERR	0x38
RS_LATEINIT_SYS_NOT_CAPABLE	0x39
RS_LATEINIT_SGX_DISABLED	0x3A
RS_LATEINIT_FACTORY_RESET_ERR	0x3B
RS_LATEINIT_NVSAREA_ERR	0x3C
RS_LATEINIT_GET_NVVAR_ERR	0x3D
RS_LATEINIT_EXPOSE_PROTO_ERR	0x3E

RS_LATEINIT_LOCKVARS_ERR	0x3F
RS_LATEINIT_VAR_ROTO_ERR	0x40
RS_LATEINIT_CALLBACK_OTHER	0x50
RS_LATEINIT_CALLBACK_NOMEM	0x51
RS_LATEINIT_CALLBACK_BIOSPARAM_ERR	0x52
RS_LATEINIT_CALLBACK_MICROCODE_LAUNCH_ERR	0x53
RS_LATEINIT_CALLBACK_UPDT_TIMESTMP_ERR	0x54
RS_LATEINIT_CALLBACK_UPDT_PKG_INFO_ERR	0x55
RS_LATEINIT_CALLBACK_LAUNCHCTRL_ERR	0x56
RS_LATEINIT_CALLBACK_UPDT_KEYBLOBS_ERR	0x57
RS_LATEINIT_CALLBACK_TCBRECOVERY_ERR	0x58
RS_LATEINIT_CALLBACK_STORPLATMANIF_ERR	0x59
RS_LATEINIT_CALLBACK_LEGACYVARS_ERR	0x5A
RS_LATEINIT_CALLBACK_REGSTATE_VAR_ERR	0x5B

3.3.4. SGX Status

The possible values reported for SGX on multi-package platforms are:

- SGX is enabled
- A reboot is required to finish enabling SGX
- SGX is disabled and a Software Control Interface is not available to enable it
- SGX is not enabled on this platform. More details are unavailable
- SGX is disabled, but can be enabled manually in the BIOS setup
- SGX is not supported by this CPU

4.Multi-Package Registration Libraries

4.1. SGX Multi-Package UEFI Variables Access Library

This library provides a set of C-like APIs that allow applications to interface with the multi-package SGX UEFI variables used to communicate with BIOS. The Multi-Package Registration Agent, the PCK Cert ID Retrieval Tool and the Multi-Package Management Tool all link to this library. You can develop your own tools using this library to suit your SGX attestation infrastructure.

4.1.1. Initialize the Multi-Package UEFI Library (MP UEFI Library)

Description

Provides the UEFI variable directory path and the logging level that the UEFI library uses in the other functions. Only the Linux version of the library uses the 'path' input, and it is ignored in the Windows version of the library. You must call this function before using the other APIs provided by this library.

Syntax

```
MpResult mp_uefi_init(
                    const char* path,
                    const LogLevel logLevel);
```

Parameters

path [In]

Linux absolute path to the UEFI variables directory. For Linux, if the value is NULL, the default UEFI path of /sys/firmware/efi/efivars/ is used. For Windows, this parameter is ignored.

logLevel [In]

Set the logging level. Logging messages default to stdout. You can create an auxiliary logging function and link with the MP UEFI Library to change the output location.

- Linux:

```
void log_message_aux(
                    LogLevel level,
                    const char *format,
                    va_list argptr)
```

- Windows:

```
void uefi_log_message_aux(
                    LogLevel glog_level,
                    LogLevel level,
                    const char* format,
                     ...)
```

Return Values

MP_SUCCESS:

The MP UEFI library successfully initialized.

MP_REDUNDANT_OPERATION:

The MP UEFI library was already initialized.

MP_MEM_ERROR:
Failed to initialize the MP UEFI library.

4.1.2. Retrieve the Registration Request Type

Description

Returns the type of data structure in the SgxRegistrationServerRequest UEFI variable. Currently, the library only supports platform manifest and add package structures.

Syntax

```
MpResult mp_uefi_get_request_type(
                        MpRequestType *type);
```

Parameters

type [Out]
Holds the pending request type or MP_REQ_NONE.

Return Values

MP_SUCCESS:
The API either found the SgxRegistrationServerRequest UEFI variable and 'type' contains the request type or the API could not find the SgxRegistrationServerRequest UEFI variable and 'type' contains MP_REQ_NONE.

MP_INVALID_PARAMETER:
The parameter type is NULL.

MP_UEFI_INTERNAL_ERROR:
The request structure header in the SgxRegistrationServerRequest UEFI variable has an invalid version, invalid size, or unrecognized GUID.

MP_NOT_INITIALIZED:
The MP UEFI library was not initialized.

4.1.3. Retrieve the BIOS Registration Server Request

Description

Returns the contents of the SgxRegistrationServerRequest UEFI variable. It also returns the required size of the 'request' structure in the parameter 'request_size' if you pass in NULL for the 'request' parameter.

Syntax

```
MpResult mp_uefi_get_request(
                        uint8_t *request,
                        uint16_t *request_size);
```

Parameters

request [Out]

Holds the request buffer to be populated. When this value is NULL but 'request_size' is not NULL, the API will return the size of the request in the SgxRegistrationServerRequest UEFI variable in 'request_size'.

request_size [In/Out]

If 'request' is not NULL, it contains the size in bytes of buffer pointed to by 'request'. Upon a successful execution, the API sets it to the number of bytes written to 'request'. If 'request' is NULL or the inputted 'request_size' is too small to contain the request (return value is MP_USER_INSUFFICIENT_MEM), the API sets it to the number of bytes required to contain the 'request' data.
Must not be NULL.

Return Values

MP_SUCCESS:

Successfully read the contents of the SgxRegistrationServerRequest UEFI variable if 'request' is not NULL or 'request_size' contains the required buffer size when 'request' is NULL.

MP_INVALID_PARAMETER:

The parameter 'request_size' is NULL.

MP_NO_PENDING_DATA:

The API could not find the SgxRegistrationServerRequest UEFI variable.

MP_USER_INSUFFICIENT_MEM:

The size of the request exceeds the size of the inputted 'request'.

MP_UEFI_INTERNAL_ERROR:

The request structure header in the SgxRegistrationServerRequest UEFI variable has an invalid version or invalid size.

MP_NOT_INITIALIZED:

The MP UEFI library was not initialized.

4.1.4. Provide to BIOS the Registration Server Response

Description

The Registration Service may generate responses to the data provided in the SgxRegistrationServerRequest UEFI variable. This API allows software to provide those server responses to BIOS via the SgxRegistrationServerResponse UEFI variable. Currently, only the Add Package (Replace Package) boot flow generates a response data from the Registration Service.
If the SgxRegistrationServerResponse UEFI variable is not already available, this API creates it.

Syntax

```
MpResult mp_uefi_set_server_response(
                        const uint8_t *response,
                        uint16_t *response_size);
```

Parameters

response [In]

Contains the response from the registration authority service.

response_size [In]

Size of response buffer in bytes.

Return Values

MP_SUCCESS:

Successfully wrote the inputted data to SgxRegistrationServerResponse UEFI variable.

MP_INVALID_PARAMETER:

Either 'response' or 'response_size' is NULL.

MP_UEFI_INTERNAL_ERROR:

Error encountered when writing to the UEFI variable.

MP_NOT_INITIALIZED:

The MP UEFI library was not initialized.

4.1.5. Retrieve Platform Information from BIOS

Description

This API reads data from the SgxRegistrationPackageInfo UEFI variable. Currently, BIOS uses this variable to provide software with the key blobs generated for each CPU package. The platform owner needs to enable a BIOS configuration ('SGX Package Info In-band Access') before it provides this information. This data is not provided to the software by default.

Syntax

```
MpResult mp_uefi_get_key_blobs(
                    uint8_t *blobs,
                    uint16_t blobs_size);
```

Parameters

blobs [Out]

Holds the package info buffer to be populated. When this value is NULL but 'blobs_size' is not NULL, the API returns the size of the data in the SgxRegistrationPackageInfo UEFI variable in 'blobs_size'.

blobs_size [In/Out]

If 'blobs' is not NULL, it contains the size in bytes of the buffer pointed to by 'blobs'. Upon a successful execution, the API sets it to the number of bytes written to the 'blobs' buffer.
If 'blobs' is NULL or the inputted 'blobs_size' is too small to contain the package info data (return value is MP_USER_INSUFFICIENT_MEM), the API sets it to the number of bytes required to contain the package info data.
Must not be NULL.

Return Values

MP_SUCCESS:

Successfully read the contents of the SgxRegistrationPackageInfo UEFI variable if 'blobs' is not NULL or 'blobs_size' contains the required buffer size when 'blobs' is NULL.

MP_INVALID_PARAMETER:
The parameter 'blobs_size' is NULL.

MP_UEFI_INTERNAL_ERROR:
The request structure header in the SgxRegistrationPackageInfo UEFI variable has an invalid version or invalid size.

MP_NO_PENDING_DATA:
SgxRegistrationPackageInfo UEFI variable is not provided by BIOS.

MP_USER_INSUFFICIENT_MEM:
The size of the package info exceeds the size of the inputted 'blobs'.

MP_NOT_INITIALIZED:
The MP UEFI library was not initialized.

4.1.6. Retrieve Registration Status

Description

This API reads the SgxRegistrationStatus UEFI variable and returns the registration, package info, and error code information.

Syntax

```
MpResult mp_uefi_get_registration_status(
                    MpRegistrationStatus *status);
```

Parameters

status [Out]
Holds the registration status. Must not be NULL.

Return Values

MP_SUCCESS:
Successfully read the SgxRegistrationStatus UEFI variable.

MP_INVALID_PARAMETER:
The parameter 'status' is NULL.

MP_UEFI_INTERNAL_ERROR:
The request structure header in the SgxRegistrationStatus UEFI variable has an invalid version, invalid size or the variable was not found.

MP_NOT_INITIALIZED:
The MP UEFI library was not initialized.

4.1.7. Set the Registration Status

Description

This API allows to write to the SgxRegistrationStatus UEFI variable. This variable can only be written under certain circumstances. See the definition of SgxRegistrationStatus UEFI variable for

more information. You can use this API to modify the registration and package info complete bits. It also allows to set an error code that any SW encountered during processing the data provided by BIOS or the registration service infrastructure. This API overwrites the contents of the UEFI variable.

Syntax

```
MpResult mp_uefi_set_registration_status(
                    MpRegistrationStatus *status);
```

Parameters

status [In]
Holds the desired registration status.

Return Values

MP_SUCCESS:
Successfully wrote the inputted data to the SgxRegistrationStatus UEFI variable.

MP_INVALID_PARAMETER:
The parameter 'status' is NULL.

MP_UEFI_INTERNAL_ERROR:
Encountered an error while writing the SgxRegistrationStatus UEFI variable. Check logs for more information.

MP_NOT_INITIALIZED:
The MP UEFI library was not initialized.

4.1.8. Retrieve the Registration Service Configuration

Description

This API reads the SgxRegistrationConfiguration UEFI variable. This variable contains the information that the software uses for contacting the registration infrastructure services.

Syntax

```
MpResult mp_uefi_get_registration_server_info(
                                    uint16_t flags,
                                    string *server_address,
                                    uint8_t *server_id,
                                    uint16_t *server_id_size);
```

Parameters

flags [Out]
Holds the retrieved registration flags in the SgxRegistrationConfiguration UEFI variable.

server_address [Out]
Holds the registration server address.

server_id [Out]
Address of server_id buffer to be populated (SgxRegistrationServerID).

server_id_size [In/Out]

If both 'server_id' and 'server_id_size' are not NULL, it contains the size in bytes of the buffer pointed to by 'server_id'. Upon a successful execution, the API sets it to the number of bytes written to the 'server_id' buffer. If the inputted 'server_id_size' not NULL but the number of bytes is too small to contain the server_id (return value is MP_USER_INSUFFICIENT_MEM), the API sets it to the number of bytes required to contain the server id data.
If 'server_id' is NULL and 'server_id_size' is not NULL, the API sets it to the number of bytes required to contain the server id data.
If 'server_id_size' is NULL, no 'server_id' information is returned.

Return Values

MP_SUCCESS:

Successfully read the contents of the SgxRegistrationConfiguration UEFI variable.

MP_INVALID_PARAMETER:

Either 'flags' or 'response_size' is NULL.
The version of the SgxRegistrationServerInfo in the SgxRegistrationConfiguration UEFI variable is not supported.

MP_USER_INSUFFICIENT_MEM:

The size of the server id read exceeds the size of the inputted 'server_id'.

MP_UEFI_INTERNAL_ERROR:

The request structure header in the SgxRegistrationConfiguration UEFI variable has an invalid version or the variable was not found.

MP_NOT_INITIALIZED:

The MP UEFI library was not initialized.

4.1.9. Set the Registration Service Information

Description

This API allows software to modify the registration authority service information in the SgxRegistrationConfiguration UEFI variable. This includes the registration authority service URL and the SgxRegistrationServerInfo. This UEFI variable is only writable when SGX is disabled. It first reads the UEFI variable then modifies the contents and writes it back. The URL is optional and keeps the existing value, but the server_id is not optional.

Syntax

```
MpResult mp_uefi_set_registration_server_info(
                                    const uint16_t flags,
                                    const string *server_address,
                                    const uint8_t *server_id,
                                    const uint16_t server_id_size);
```

Parameters

flags [In]

Holds the registration flags to write to the SgxRegistrationConfiguration UEFI variable.

server_address [In]
Holds the registration server address.

server_id [In]
Address of SgxRegistrationServerID buffer to be written.

server_id_size [In]
Size in bytes of the data stored in the 'server_id' buffer.

Return Values

MP_SUCCESS:
Successfully wrote the inputted data to the SgxRegistrationConfiguration UEFI variable.

MP_INVALID_PARAMETER:
The 'server_id' parameter is NULL, the size of the URL string is too long, or the URL is an invalid value.

MP_UEFI_INTERNAL_ERROR:
The request structure header in the SgxRegistrationConfiguration UEFI variable has an invalid version, or the variable was not found.

MP_UNEXPECTED_ERROR:
The API encountered an unexpected error. Check logs for more information.

MP_MEM_ERROR:
Insufficient memory

MP_NOT_INITIALIZED:
The MP UEFI library was not initialized.

4.1.10. Exit the Multi-Package UEFI Library

Description

Free any resources used by the MP UEFI Library.

Syntax

```
MpResult mp_uefi_terminate();
```

Parameters

None

Return Values

MP_SUCCESS:
Successfully terminated the MP UEFI library.

MP_REDUNDANT_OPERATION:
The MP UEFI library was not initialized or has been terminated.

4.2. SGX Multi-Package Registration Service Network Library (MP Network Library)

This library provides a set of C-like APIs that allows applications to communicate with the REST APIs defined by the Intel® SGX Registration Service APIs. The Multi-Package Registration Agent links to this library. You can develop your own tools using this library to suit your SGX attestation infrastructure.

4.2.1. Initialize the Multi-Package Network Library

Description

Provides the configuration data needed by the library to communicate to the registration authority service providing the REST APIs.

Syntax

```
MpResult mp_network_init(
                    const char *server_address,
                    const char *subscription_key,
                    const ProxyConf *proxy,
                    const LogLevel logLevel);
```

Parameters

server_address [In]
: Server URL that exposes the REST APIs.

subscription_key [In]
: Some REST APIs may require a subscription key. Currently, only the add package API requires a subscription key.

proxy [In]
: Desired proxy configurations of the platform communicating to the registration service.

logLevel [In]
: Set the logging level. Logging messages default to stdout. You can create an auxiliary logging function and link with the MP Network Library to change the output location.

- Linux:

```
void log_message_aux(
                    LogLevel level,
                    const char *format,
                    va_list argptr)
```

- Windows:

```
void uefi_log_message_aux(
                    LogLevel glog_level,
                    LogLevel level,
                    const char* format,
                     ...)
```

Return Values

MP_SUCCESS:
: Successfully initialized the network library.

MP_INVALID_PARAMETER:

Either the 'server_address', 'subscription_key' or the 'proxy' parameter is NULL
The size of the URL string is too long or the URL is an invalid value.

MP_REDUNDANT_OPERATION:
The MP Network library was already initialized.

MP_MEM_ERROR:
Failed to initialize the MP Network library.

4.2.2. Send a Request to the Registration Server

Description

Sends a supported request type to the registration service and returns the response.

Syntax

```
MpResult mp_send_binary_request(
                    const MpRequestType *request_type,
                    const uint8_t *request,
                    const uint16_t request_size,
                    const uint8_t *response,
                    const uint16_t *response_size,
                    HttpStatusCode *status_code,
                    RegistrationErrorcode *error_code);
```

Parameters

request_type [In]
The type of request to be sent. Currently, only the platform manifest and the add package request types are supported.

request [In]
Request buffer to send to the service.

request_size [In]
Size in bytes of the 'request' buffer.

response [In/Out]
Buffer that contains the response from the service.

response_size [In]
Size in bytes of the 'response' buffer.

status_code [Out]
HTTPS status code returned by the service.

error_code [In]
The error code generated by the registration service. See the **MPA_RS_*** error codes defined in MPA Error Codes.

Return Values

MP_SUCCESS:

Successfully sent the request.

MP_INVALID_PARAMETER:
Either the 'request', 'response_size', 'status_code' or the 'error_code' parameter is NULL.
The 'response' parameter is not NULL, but the 'response_size' value is 0.
The 'request_size' value is 0.
The 'request_type' value is not supported.
The 'request_type' is an add package, but the network library was not initialized with a valid sized subscription key.

MP_NETWORK_ERROR:
Failed to set up the network connection, proxy or other failure sending request to the server.

MP_UNEXPECTED_ERROR:
The API encountered an unexpected error. Check logs for more information.

MP_MEM_ERROR:
Error allocating memory.

MP_USER_INSUFFICIENT_MEM:
The 'response_size' parameter value is too small to contain the server response.

MP_NOT_INITIALIZED:
The MP UEFI library was not initialized.

4.2.3. Exit the Multi-Package Network Library

Description

Free any resources used by the MP Network Library.

Syntax

```
MpResult mp_uefi_terminate();
```

Parameters

None

Return Values

MP_SUCCESS:
Successfully terminated the MP Network library.

MP_REDUNDANT_OPERATION:
The MP Network library was not initialized or has been terminated.

A. Data Structures

A.1. Common Data Structures

```
/** Result codes returned by the API's in the UEFI and Network libraries */
typedef enum {
    MP_SUCCESS = 0,
    MP_NO_PENDING_DATA = 1,
    MP_ALREADY_REGISTERED = 2,
    MP_MEM_ERROR = 3,
    MP_UEFI_INTERNAL_ERROR = 4,
    MP_USER_INSUFFICIENT_MEM = 5,
    MP_INVALID_PARAMETER = 6,
    MP_SGX_NOT_SUPPORTED = 7,
    MP_UNEXPECTED_ERROR = 8,
    MP_REDUNDANT_OPERATION = 9,
    MP_NETWORK_ERROR = 10,
    MP_NOT_INITIALIZED = 11,
    MP_INSUFFICIENT_PRIVILEGES = 12
} MpResult;

/** Supported logging levels */
typedef enum _LogLevel
{
    MP_REG_LOG_LEVEL_NONE = 0,
    MP_REG_LOG_LEVEL_FUNC = 1,
    MP_REG_LOG_LEVEL_ERROR = 2,
    MP_REG_LOG_LEVEL_INFO = 3,
    MP_REG_LOG_LEVEL_MAX_VALUE
}LogLevel;

/* Supported types of data structures in the SgxRegistrationServerRequest UEFI
variable */
typedef enum {
    MP_REQ_REGISTRATION = 0,
    MP_REQ_ADD_PACKAGE = 1,
    MP_REQ_NONE = 2
} MpRequestType;

/** These are the possible error codes reported by the agent in the
SgxRegsitrationStatus UEFI variable in the ErrorCode field */
typedef enum _RegistrationErrorCode
{
    MPA_SUCCESS = 0x00,
    MPA_AG_UNEXPECTED_ERROR = 0x80,        ///< Unexpected agent internal error.
    MPA_AG_OUT_OF_MEMORY = 0x81,           ///< Out-of-memory error.
    MPA_AG_NETWORK_ERROR = 0x82,           ///< Proxy detection or network
                                            ///< communication error
    MPA_AG_INVALID_PARAMETER = 0x83,       ///< Invalid Parameter passed in
    MPA_AG_INTERNAL_SERVER_ERROR = 0x84,   ///< Internal server error occurred.
```

```
    MPA_AG_SERVER_TIMEOUT = 0x85,          ///< Server timeout reached
    MPA_AG_BIOS_PROTOCOL_ERROR = 0x86,     ///< BIOS Protocol error

    /* Registration Server HTTP 400 Response Error details */
    MPA_RS_INVALID_REQUEST_SYNTAX = 0xA0,  ///<The request could not be understood
                                               by the server due to malformed
                                               syntax.
    MPA_RS_PM_INVALID_REGISTRATION_SERVER = 0XA1 ///< RS rejected request because
                                                    it is intended for
                                                    different Registration
                                                    Server (Registration Server
                                                    Authentication Key
                                                    mismatch).
    MPA_RS_INVALID_OR_REVOKED_PACKAGE = 0xA2, ///< RS rejected request due to
                                                 invalid or revoked CPU
                                                 package.
    MPA_RS_PACKAGE_NOT_FOUND = 0xA3,       ///< RS rejected request as at least
                                               one of the CPU packages could not
                                               be recognized by the server.
    MPA_RS_PM_INCOMPATIBLE_PACKAGE = 0xA4, ///< RS rejected request as at least
                                               One of the CPU packages is
                                               incompatible with rest of the
                                               packages on the platform.
    MPA_RS_PM_INVALID_PLATFORM_MANIFEST = 0xA5,  ///< RS rejected request due to
                                                    invalid platform
                                                    configuration.
    MPA_RS_AD_PLATFORM_NOT_FOUND = 0xA6,   ///< RS rejected request as provided
                                               platform instance is not
                                               recognized by the server.
    MPA_RS_AD_INVALID_ADD_REQUEST = 0xA7,  ///< RS rejected request as the Add
                                               Package payload was invalid.
    MPA_RS_UNKOWN_ERROR = 0xA8,            ///< RS rejected request for unknown
                                               reason.  Probably means software
                                               needs to be updated with newly
                                               defined RS errors
} RegistrationErrorCode;
```

A.2. Multi-Package UEFI Library Data Structures

```
/** Body definition of the SgxRegistrationStatus UEFI Variable*/
typedef struct {
    union {
        uint16_t status;
        struct {
            uint16_t registrationStatus:1;
            uint16_t packageInfoStatus:1;
            uint16_t reservedStatus:14;
        };
    };
    RegistrationErrorCode errorCode;
} MpRegistrationStatus;
```

A.3. Multi-package Network Library Data Structures

```
/** Proxy type definition specified in the configuration file and network library
initialization function. Initialize the Multi-Package Network Library */
typedef enum _ProxyType
{
    MP_REG_PROXY_TYPE_DEFAULT_PROXY = 0,    ///< Use the configuration in your
                                            ///< operating system
    MP_REG_PROXY_TYPE_DIRECT_ACCESS = 1,    ///< Direct access to the internet
    MP_REG_PROXY_TYPE_MANUAL_PROXY = 2,     ///< Set the proxy URL directly
    MP_REG_PROXY_TYPE_MAX_VALUE
} ProxyType;

/** Expands the ProxyType structure with a URL to support
MP_REG_PROXY_TYPE_MANUAL_PROXY */
typedef struct _ProxyConf{
    ProxyType proxy_type;
    char proxy_url[MAX_PATH_SIZE];
} ProxyConf;

/** These are the possible HTTP status codes return from the registration service
*/
typedef enum _HttpResponseCode
{
    MPA_RS_PLATFORM_CREATED = 201,     ///< 201 Operation successful - a new
                                       ///< platform instance has been registered
                                       ///< in the registration service's
                                       ///< database.
    MPA_RS_PACKAGES_BEEN_ADDED = 200, ///< 200 Operation successful – packages
                                       ///< have been added to an existing
                                       ///< platform instance.
    MPA_RS_BAD_REQUEST = 400,          ///< Invalid payload. The client should not
                                       ///< repeat the request without
                                       ///< modifications.
    MPA_RS_INTERNAL_SERVER_ERROR = 500, ///< Internal server error occurred.
    MPA_RS_SERVICE_UNAVAILABLE = 503, ///< Server is currently unable to process
                                       ///< the request. The client should try to
                                       ///< repeat its request after some time.
    MPA_RS_UNSUPPORTED_MEDIA_TYPE = 415, ///< MIME type specified in the request
                                          ///< is not supported by the server.
} HttpStatusCode;
```

B. BIOS Multi-Package UEFI Variables

The UEFI variables used for communication between BIOS and software enable a protocol to control the availability of data structure on boot flows. The data provided by BIOS and the registration authority service are considered privacy sensitive. They contain hardware identifiers that a platform owner may want to restrict to software. For this reason, the protocol includes the ability for software to indicate to BIOS that it finished processing the structures. This increases the complexity of the UEFI protocol implementation. If a platform owner does not need to implement the privacy protection provided by the protocol, they can consider simplifying the protocol and always provide the structures to software as read only values, and BIOS does not use the 'complete' flags to remove the variables from a subsequent boot flow.

UEFI variable are not exposed to guest VMs by default – VMM must implement the exposure of the variables to a guest VM. This may satisfy some platform owners that want to prevent the privacy sensitive information from guest workloads.

B.1. SgxRegistrationConfiguration

BIOS creates this variable to communicate the registration authority service URL to software. Software reads this variable when there is data in SgxRegistrationServerRequest and SgxRegistrationStatus.SgxRegistrationComplete bit is 0. The platform owner must disable SGX to allow software to write to this variable. When this variable is writeable by software, the software can modify which registration authority service the platform to use. When the registration authority service changes, any current platform registration is invalid. On writes to this variable, the BIOS clears all platform registration data (KEY_BLOBS and PLATFORM_MANIFESTS). Any attempts to write to this variable when it is read-only are ignored.

BIOS provides this variable to software on all boot flows.

The 'Flags' field contains a flag indicating when the platform owner allows the registration authority service to store the platform keys. Software uses this flag to determine if direct or indirect registration is enabled for the platform. The flag can be modified using the BIOS UI's 'SGX Auto MP Registration' knob.

<table>
<tr><td></td><td>SgxRegistrationConfiguration</td></tr>
<tr><td>GUID</td><td>18b3bc81-e210-42b9-9ec8-2c5a7d4d89b6</td></tr>
<tr><td>Size</td><td>1514</td></tr>
<tr><td>Attributes</td><td>Read-only when SGX is enabled. Read-Write when SGX is disabled.</td></tr>
<tr><td>Description</td><td>BIOS creates this variable during all boot flows. Software can use it to modify the registration authority service.

Contains the following fields:

<table>
<tr><th>Name</th><th>Size</th><th>Type</th><th>Description</th></tr>
<tr><td>Version</td><td>2</td><td>LE Integer</td><td>1</td></tr>
</table>
</td></tr>
</table>

Size	2	LE Integer	Size in bytes of data below
Flags	2	LE Integer	BIT 0: RS Encrypted Keys 0: Registration Server saves platform keys 1: Registration Server does not save platform keys Bits 1:15: Reserved MBZ
SgxRegServerInfo	1514	Mix	As defined in MP SGX_REGISTRATION_SERVER_INFO

Note: The above data can be part of BIOS setup configuration variable

B.1.1. Header

The Header is the first field of multi-package data structures that is shared between components.

Name	Size	Type	Description
GUID	16	Byte Array	GUID uniquely identifying the data structure.
SIZE	2	LE Integer	Data structure size excluding the size of this header.
VERSION	2	LE Integer	Structure version.
RESERVED	12	N/A	Reserved: This field is 0.

B.1.2. PubKey

This structure represents an RSA3072 public key. It does not contain a HEADER since it is never used as an "upper-layer" structure.

Name	Size	Type	Description
MODULUS	384	LE Integer	RSA key pair modulus (N)
PUBEXP	4	LE Integer	RSA public exponent (E)

B.1.3. SGX Registration Server ID

This structure represents the identity of the registration authority service. BIOS provides it to microcode so it can properly generate platform manifests and key blobs.

The self-signed REGISTRATION_SERVER_ID structure contains all the keys the registration authority service uses for authorizing and decrypting the platform keys. The structure includes two 3072-bit RSA keys. The Registration Server Authorization Key (RSAK) is used to sign PLATFORM_MEMBERSHIP_CERTS and this structure. The Registration Service Encryption Key (RSEK) is used by microcode for encrypting the platform keys in the platform manifest.

Name	Size (bytes)	Data Type	Description
Header	32	Mix	GUID: 31A12AFE-0720-4EBC-B64E-C4B3C7F8BC0F. Version: 1
RSNAME	32	Byte Array	Registration Server self-selected public ID. Frequently the hash of the server's domain name or something to this effect
RSAK	PubKey (388)	Mix	Registration Server's RSA Authorization Key.
RSEK	PubKey (388)	Mix	Registration Server's RSA public key used to encrypt platform keys.
Signature	384	LE Integer	This entire structure is self-signed using the Registration Server's RSAK.

B.1.4. SGX Registration Server Info

This structure links the registration authority service URL with the signed REGISTRATION_SERVER_ID structure.

Name	Size (bytes)	Data Type	Description
HEADER	32	Mix	GUID: 212FE183-6B1A-42A1-A7A9-DA3AB6B7BD02 Version: 1
URL_SIZE	2	LE Integer	Number of bytes in the URL.

URL	256	Byte Array	ASCII representation of the URL name. Does not contain '\0' NULL terminator.
SgxRegistrationServerID	sizeof (SgxRegistrationServerID)	Mix	Registration authority services's RSA public keys and RSNAME

B.2. SGX Registration Server Request

BIOS exposes this variable when data needs to be sent to the registration authority service. Its contents depend on the registration boot flow. For Initial Platform Establishment and TCB Recovery boot flows, it contains the platform manifests. For the Add Package flow, it contains add package structure. BIOS only generates this variable when there is data to send. The platform manifest and add package structures contain privacy sensitive information and should only be exposed to software until registration completes. Software indicates that registration is complete by setting the SgxRegistrationStatus.SgxRegistrationComplete bit to 1. BIOS clears the SgxRegistrationStatus.SgxRegistrationComplete bit to 0 when there is data to process, and software expects this variable to be available. Software processes its contents and sets the SgxRegistrationStatus.SgxRegistrationComplete bit to 1 to indicate whether the registration flow completes successfully. Software also sets the SgxRegistrationStatus.SgxRegistrationComplete to 1 on terminal errors received from the server as an indication that no retries resolve the error. If the registration does not complete and the software does not set the SgxRegistrationStatus.SgxRegistrationComplete bit to 1, BIOS provides the same data in this variable on the next boot for software to retry processing the data. Otherwise, BIOS does not present this same data on a subsequent boot.

Any errors encountered by software are reported with an error code in SgxRegistrationStatus.ErrorCode.

<table>
<tr><td>Name</td><td>SgxRegistrationServerRequest</td></tr>
<tr><td>GUID</td><td>304e0796-d515-4698-ac6e-e76cb1a71c28</td></tr>
<tr><td>Size</td><td>N/A</td></tr>
<tr><td>Attributes</td><td>Read-Only</td></tr>
<tr><td>Description</td><td>This variable is created by BIOS when SgxRegistrationStatus.SgxRegistrationComplete is 0.

Contains several self-signed data structures based on boot scenario.
<table>
<tr><th>Boot Scenario</th><th>Contents</th><th>Size</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td>Version</td><td>2</td><td>LE Integer</td><td>2 – When content is PLATFORM_MANIFST
1 or 2 – When content is ADD_REQUEST</td></tr>
</table></td></tr>
</table>

	Size	2	LE Integer	Size in bytes of data below (after trimming)
Initial Platform Establishment/TCB Recovery	PLATFORM_MANIFEST	Variable (PM Header size will always be untrimmed size)	Mix	Contains 2 PLATFORM_MANIFESTS. The first PLATFORM_MANIFEST is from the IPE flow and the second is for TCB Recovery (for the IPE boot flow, the TCB Recovery PLATFORM_MANIFEST will be all zeros and will be trimmed the same as the IPE PLATFORM_MANIFEST). Data Header: **GUID:** 178E874B-49E4-4AA5-99BB-3057170925B4. **Version:** 1
Add package	ADD_REQUEST	211	Mix	Contains the ADD_REQUEST structure. Data Header: **GUID:** 696519ca-73c1-4785-a0f6-4d289d37e995 **Version:** 1

B.3. SGX Registration Server Response

This variable is created by software when it successfully receives response data from the registration authority service. Once successfully completed, the software sets the SgxRegistrationStatus.SgxRegistrationComplete bit to 1 to indicate to BIOS that the software does not require the same SGX Registration Server Request data on the subsequent boot flow. Currently, the only response data from the registration authority service platform membership certificates in response to a successful add request.

You shouls clear the data in this variable once it is consumed by BIOS to protect privacy sensitive data on the next boot.

Name	SgxRegistrationServerResponse
GUID	89589c7b-b2d9-4fc9-bcda-463b983b2fb7
Size	4 + 8*sizeof(PLATFORM_MEMBERSHIP_CERT)
Attributes	Read-Write

<table>
<tr><td>Description</td><td>This variable is created by OS/SW using data it received from the registration authority server.

Contains response data from the registration server.
<table>
<tr><th>Name</th><th>Size</th><th>Type</th><th>Description</th></tr>
<tr><td>Version</td><td>2</td><td>LE Integer</td><td>1</td></tr>
<tr><td>Size</td><td>2</td><td>LE Integer</td><td>Size in bytes of data below</td></tr>
<tr><td>Platform Member Ship Certs[8]</td><td>8 * sizeof(PLATFORM_MEMBERSHIP_CERT)</td><td>Mix</td><td>Array of platform memberships certs returned by the registration server.

Empty array elements are all 0x00s.

BIOS clears the data once it has read it.</td></tr>
</table>
</td></tr>
</table>

B.4. SGX Registration Package Info

Currently, this variable contains the key blob structures for each package in the platform. BIOS only generates this variable when new or modified key blobs are available. Microcode may generate new key blobs in any boot flow. The software can use this variable to store the key blobs off-platform if the key blobs stored by BIOS are lost.

BIOS clears the SgxRegistrationStatus.SgxPackageInfoComplete bit to 0 when key blobs are available. The key blobs contain privacy sensitive information and should only be exposed until software reads them. Once software reads the data out for backup storage, it should set SgxRegistrationStatus.SgxPackageInfoComplete bit to 1 to indicate to BIOS that it should not expose this same data on a subsequent boot flow. If the software does not set the SgxRegistrationStatus.SgxPackageInfoComplete bit to 1, BIOS provides the same data in this variable on the subsequent boot (unless microcode generates new key blobs).

By default, this UEFI variable is not provided to software when microcode generates new key blobs. Platform owners that need to store key blobs off-platform must opt-in via a BIOS configuration. For platforms that do not opt-in for key blob storage, BIOS always sets the SgxRegistrationStatus.SgxPackageInfoComplete bit to 1 before booting to the OS.

To restore the key blobs, software can create/write to this UEFI variable so BIOS can use the key blobs on the next boot. This variable is writeable only if SGX is disabled.

Name	SgxRegistrationPackageInfo
GUID	ac406deb-ab92-42d6-aff7-0d78e0826c68
Size	8*sizeof(KEY_BLOB)
Attributes	Read-only when SGX enabled. Read-Write when SGX disabled

Description	This variable is created by BIOS using data it received from microcode. It can be created and written to by software when SGX is disabled.

Name	Size	Type	Description
Version	2	LE Integer	1
Size	2	LE Integer	Size in bytes of data below
KEY_BLOB[8]	8 * sizeof(KEY_BLOB)	Mix	Array of KEY_BLOB generated or modified by the microcode loader. Empty array elements are all 0x00s.

B.5. SGX Registration Status

This variable is created by BIOS and is always available to software for reading on all boot flows (including when SGX is disabled). When BIOS needs to communicate to the software that some action needs to take place, BIOS resets one of the defined Status bits to 0. When software completes the action, it can set the relevant status bit to 1 to prevent BIOS from requesting the same action on a subsequent boot and to limit exposure of privacy sensitive data. When all valid status bits are set to 1, BIOS makes this variable read-only. Software can also use this variable to query the status of an action on future boot flows, so the status bits need to be preserved by BIOS across boots (until a new action is required). If one of the status bits remains 0 on a subsequent boot, BIOS provides the data necessary to allow the software to retry that action.

BIOS communicates to software any errors that occurred during a boot flow by setting the Error Code to a non-zero value (with the MSBit reset to 0). If BIOS reports an error, there is no action for software to take. Software can also write to the Error Code when it encounters an error with the MSBit set to 1). To allow a software Error Code to persist across boot flows, BIOS should not overwrite a non-zero software ErrorCode on a subsequent successful normal boot flow.

Name	SgxRegistrationStatus
GUID	f236c5dc-a491-4bbe-bcdd-88885770df45
Size	2
Attributes	Read-Write then Read-only

Description

BIOS creates this variable whenever communication to the registration authority service is required or whenever a key blob backup is required.

Name	Size	Type	Description
Version	2	LE Integer	1
Size	2	LE Integer	Size in bytes of data below
Status	2	Little Endian	BIT[0]: SgxRegistrationComplete 0: SGX Registration is in progress. SgxRegistrationServerRequest is accessible. 1: SGX Registration is complete. SGXRegistrationRespons is available when ErrorCode is 0. SgxPlatformServerRequest is not accessible on next boot. BIT[1]: SgxRegistrationPackageInfo read complete 0: RegistrationPackageInfo backup in process. SgxRegistrationPackageInfo accessible. 1: RegistrationPackageInfo backup is complete SgxRegistrationPackageInfo is not accessible on next boot. BIT[15:2]: Reserved
Error Code	1	N/A	Registration Error Code. • BIOS errors have MSBit reset. • SW errors have MSBit set.