Hash Unified API SUpport Support in UEFI BIOS

Design Document

November 2019

Revision History

Date	Revision	Description
November 2019	0.1	Initial Release

§

1.0 Introduction

This is a High Level Design document that describes the implementation of a library that will perform the following:

- Based on user input, decide the highest level of signing support (SHA and RSA)
- Interfaces with openssl library to call the required SHA/RSA operation
- Provide a generic API that can be called by application such as Image Verification Lib to perform signing operation

1.1 Terminology

Term	Description
SHA	Secure Hash Algorithm
RSA	Rivest-Shamir-Adleman Key Infrastructure
PKCS	Public Key Cryptographic Standards
HDD	Hard Disk Drive
FV	Firmware Volume
CSME	Converged Security and manageability Engine
IBB	Intel Boot Block
OBB	OEM Boot Block
ACM	Authenticated Code Module
PCD	Platform Configuration Desciptor

Table 1: Acronyms

§

2.0 Objective

The main objective of this feature is to expand the capability of Secure Boot, authentication mechanism in UEFI BIOS to also support SHA384 and RSA3072 as hashing algorithm and signing key strength, respectively.

As part of this support the following requirements should be also met:

Provide a single library to make the decision on the level of signing support as opposed to the current implementation where a lot of the code is hard-coded, such as, calling into SHA256 library directly and no way to change to other algorithms such as SHA384 or SHA512. This can be achieved using a PCD.

§

3.0 Detailed Description and Resolution

The following figure provides details about signing support currently available in UEFI.

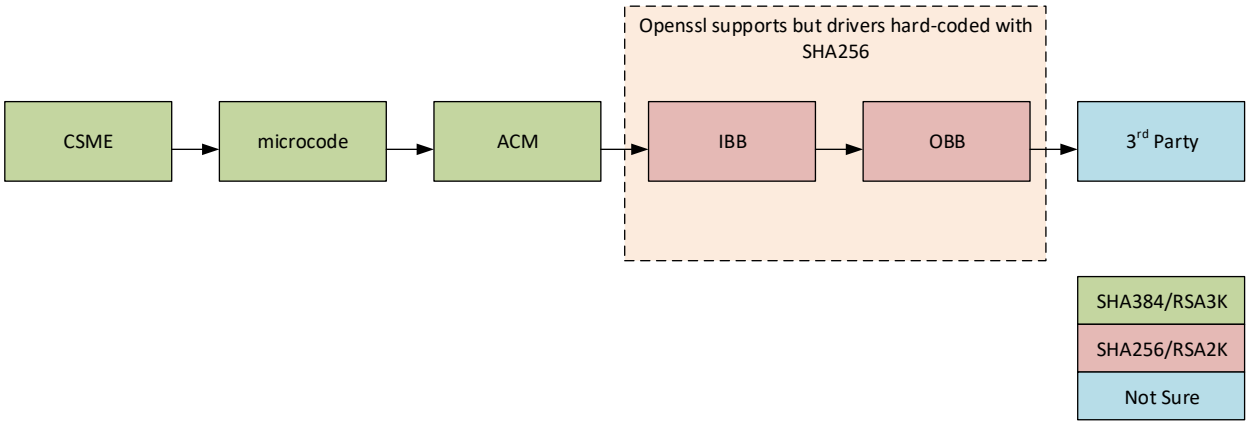

Figure 1: Current Signing Support in UEFI Secure Boot

3.1 Hard-coded Hashing API with Duplicate Code

3.1.1 Current Implementation

Most of the drivers (except TPM and Measured Boot functionality), use hard-coded APIs that interface with hashing libraries in CryptoPkg directly. The following figure explains the interface in the current implementation (using PeiBootGuardEventLog driver as an example):

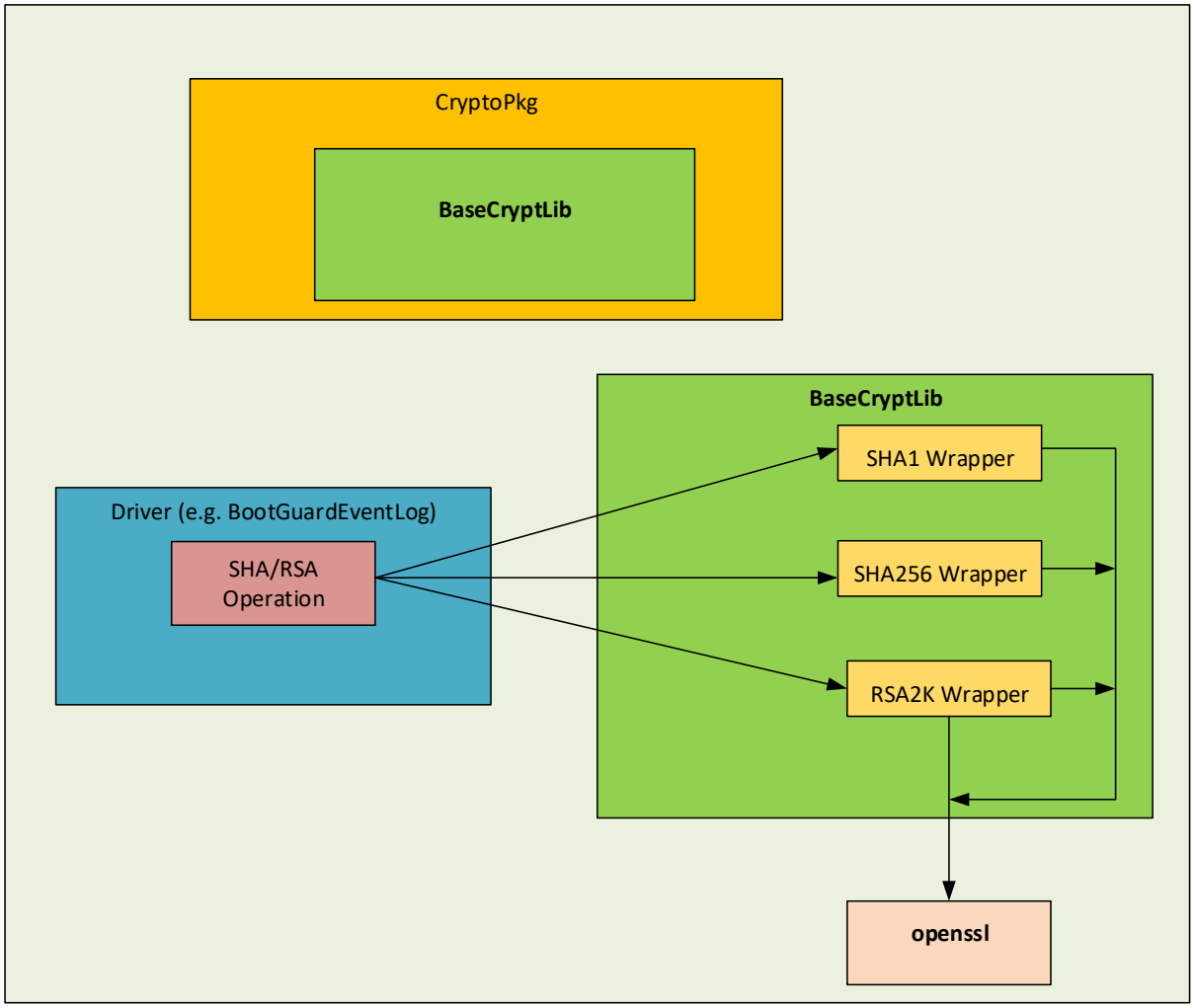

Figure 2: Current Interface to BaseCryptLib

The corresponding pseudo-code in PeiBootGuardEventLog driver is:

```
/**
Calculate DetailPCR extend value

@param[out] SHA1 DetailPCR digest
@param[out] SHA256 DetailPCR digest
**/
VOID
CalculateDetailPCRExtendValue (
```

```
   OUT UINT8 *Sha1Digest,
   OUT UINT8 *Sha256Digest,
            OUT UINT8 *sha384Digest
)
{
.
.
   /* Currently only one interface is supported build instance */
      CreateSha1Hash ((UINT8 *)&DetailPcrData, sizeof(DetailPcrData),
(UINT8 *)Sha1Digest); OR
      CreateSha256Hash ((UINT8 *)&DetailPcrData, sizeof(DetailPcrData),
(UINT8 *)Sha256Digest);
}
```

As we see above, the hashing API is hard-coded and cannot be changed without modifying the code.

3.1.2 Proposed Solution

Introduce a unifiedHashAPI library which will provide a unified API to the driver such as PeiBootGuardEventLog. The selection of the hashing algorithm will be based on the PCD, PcdSystemHashPolicy. In addition, HashAlgoInstace library is also introduced. This library will provide an interface between hash libraries implemented in BaseCryptLib in CryptoPkg and UnifiedHashApi library. The idea behind HashAlgoInstance library is that it will register a particular hash algorithm library with UnifiedHashApi library only if it is enabled and needed, thus, keeping the memory footprint manageable during build-time and run-time.

The interface will be implemented as shown in the following figure:

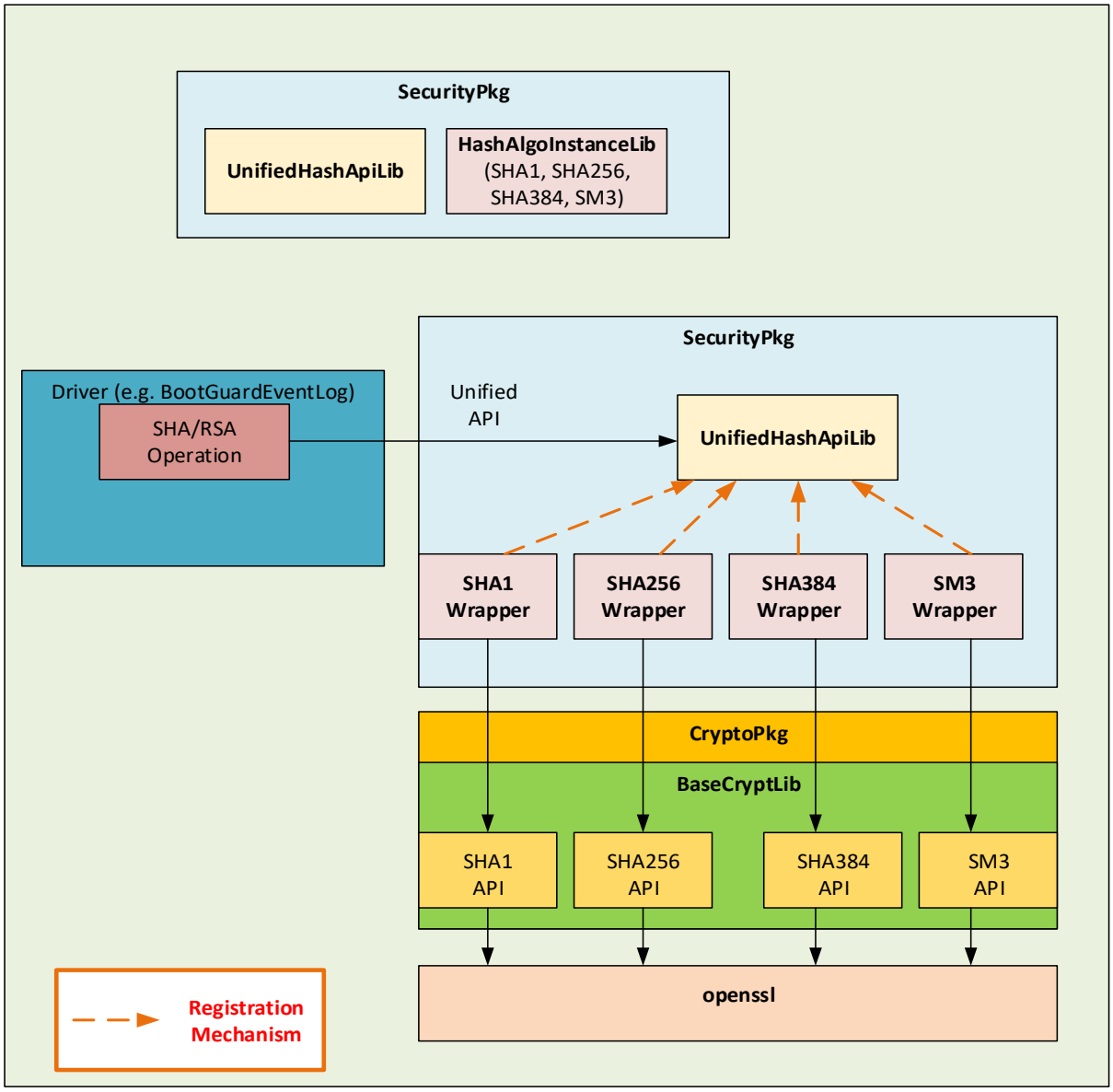

Figure 3: Proposed Unified Hash API Library Implementation

§

3.2 Appendix

3.2.1 Structure Used to Register a Hash Interface Instance

Hash registration interface.

```
typedef struct {
  EFI_GUID    HashGuid;
  HASH_INIT   HashInit;
  HASH_UPDATE HashUpdate;
  HASH_FINAL  HashFinal;
} HASH_INSTANCE_UNIFIED_API;
```

HASHGuid	Description
HASH_ALGORITHM_SHA1_GUID	SHA1 hash GUID
HASH_ALGORITHM_SHA256_GUID	SHA256 hash GUID
HASH_ALGORITHM_SHA384_GUID	SHA384 hash GUID
HASH_ALGORITHM_SHA512_GUID	SHA512 hash GUID
HASH_ALGORITHM_SM3_256_GUID	SM3_256 hash GUID

The callback to register API is as follows:

```
EFI_STATUS
EFIAPI
RegisterHashLib (
  IN HASH_INSTANCE_UNIFIED_API *HashInterface
);
Registers the API for specified SHA algorithm
HashInterface: Instance of HASH_OPERATIONS structure for specified SHA
algorithm
```

3.2.2 API Exposed to Drivers for Hash Operations

```
EFI_STATUS
EFIAPI
HashApiInit (
  IN UINT32 HashType,
  OUT HASH_HANDLE *HashHandle
)
```

Gets the Context size for specified SHA algorithm
HashType: Hash Algorithm (defined by PcdSystemHashPolicy or user-specified Hash Algorithm)
HashHandle: Hash Context Handle.

```
EFI_STATUS
EFIAPI
HashApiUpdate (
  IN UINT32 HashType,
  IN HASH_HANDLE HashHandle,
  IN VOID *Data,
  IN UINTN DataSize
)
```

Updates the HASHcontext.
HashType: Hash Algorithm (defined by PcdSystemHashPolicy or user-specified Hash Algorithm)
HashHandle: Hash Context Handle.
Data: Pointer to data used the update the SHA context
DataSize: Size of the data used to update the SHA context

```
EFI_STATUS
EFIAPI
HashApiFinal (
  IN UINT32 HashType,
  IN HASH_HANDLE HashHandle,
  OUT UINT8 *Digest
)
```

Updates the HASH context.
HashType: Hash Algorithm (defined by PcdSystemHashPolicy or user-specified Hash Algorithm)
HashHandle: Hash Context Handle.
Digest: Final Hash Value

§