CHAPTER 3

Base Platform Security Hardware Building Blocks

> *Every distraction is a possibility, Every downfall is an opportunity.*
>
> —Ria Cheruvu

Historically, the attacks on platforms have been transitioning from application-level software (SW) to user mode SW to kernel mode SW to firmware (FW) and now hardware (HW). The frequency of HW- and FW-level vulnerabilities increased substantially from 2003 to 2019 and therefore reinforces a concrete need for HW-based security to harden the platform. This is evident from the data cataloged in the National Vulnerability Database (NVD) organized as CVEs; more information about NVD can be found at https://nvd.nist.gov/. The Common Vulnerabilities and Exposures (CVE) is a list of entries with the information that identifies a unique vulnerability or an exposure and is used in many cybersecurity products and services including the NVD; more information about CVE can be found at https://cve.mitre.org/. The NVD has been mined to derive the statistics and visualizations with pertinent search terms such

S. Cheruvu et al., *Demystifying Internet of Things Security*,
https://doi.org/10.1007/978-1-4842-2896-8_3

as **Firmware** and **Hardware**. It is evident from Figure 3-1 (a) that the firmware-related CVEs have increased significantly and 2017–2018 saw the biggest jump when the hacker community started attacking the FW on the platforms. Similarly Figure 3-1 (b) shows that during the same time period, the HW-related CVEs also hit a peak. Please note that all these CVEs need to be investigated carefully for the impacted areas within a platform. But the trends are clearly pointing toward the HW as the last line of defense.

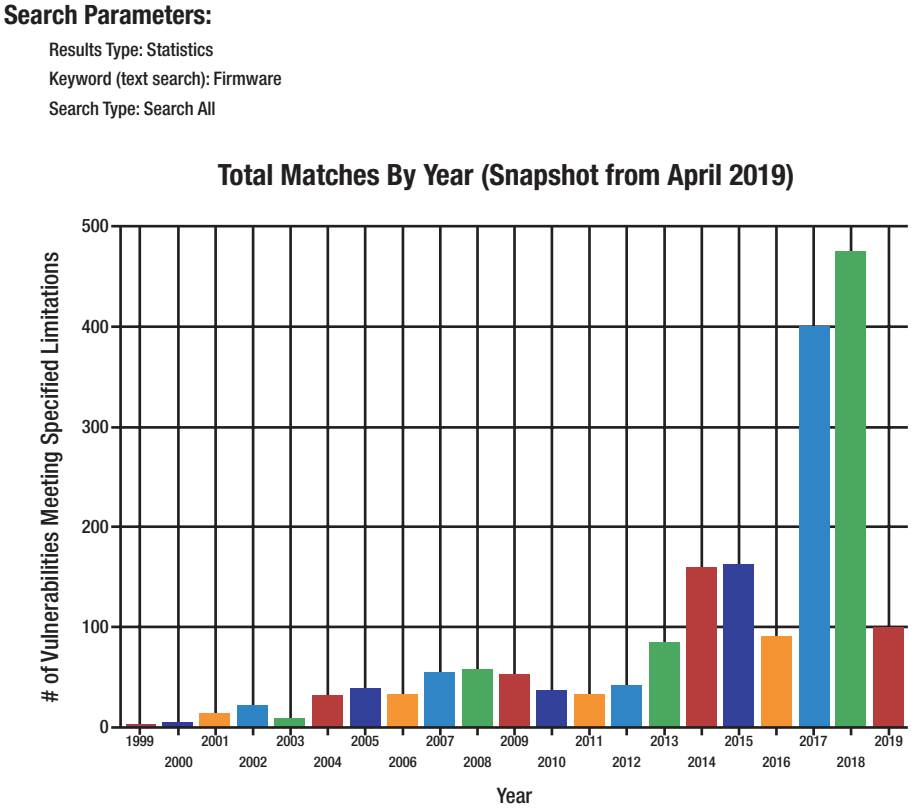

Figure 3-1. *(a) Firmware vulnerability trend chart*[1]

[1] https://nvd.nist.gov/vuln/search/statistics?form_type=Advanced&results_type=statistics&query=Firmware&search_type=all

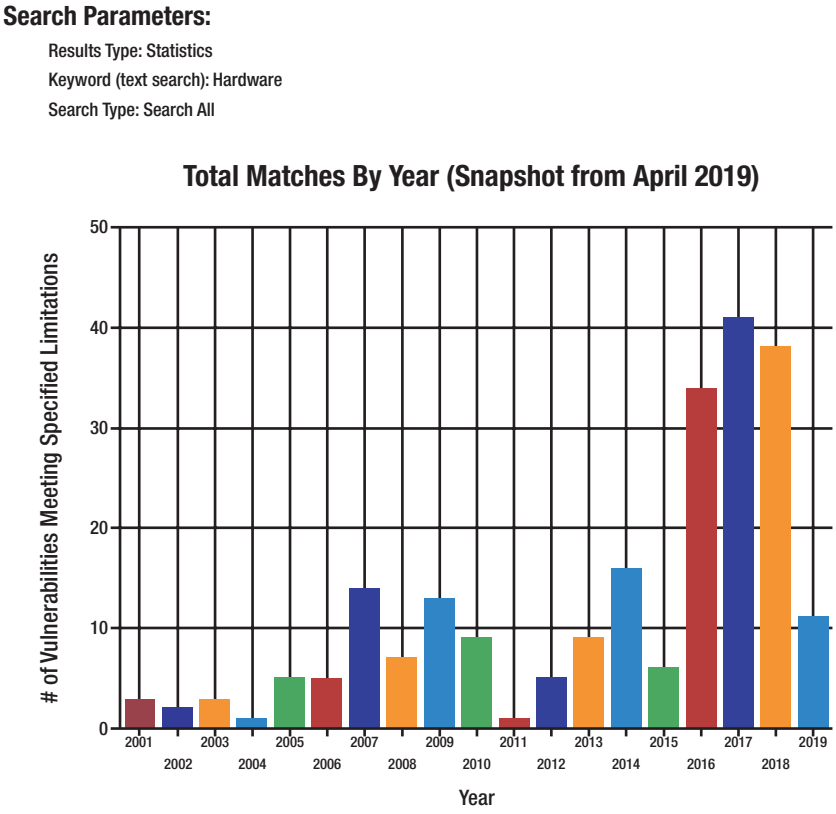

Figure 3-1. *(b) Hardware vulnerability trend chart*[2]

This chapter describes the technologies involved in securing an IoT device anchored to a Hardware Root of Trust (HWRoT) and ultimately booting into a Trusted Execution Environment (TEE). Security in an IoT environment generally involves four areas of focus:

- Protecting the device
- Protecting user identity

[2] https://nvd.nist.gov/vuln/search/statistics?form_type=Advanced&results_type=statistics&query=Hardware&search_type=all

- Protecting the data
- Managing the security at runtime

Each of these areas are worthy of detailed explanation in itself. This chapter delves into the rich set of security and privacy technologies Intel has available in their product lines and how they may be used to implement secure IoT systems. Intel's discrete CPU-PCH or System-on-Chip (SoC) products have two classes of security features; one class of features are implemented in the CPU as New Instructions (NI) with some examples being AES-NI, SHA-NI, and so on. The second class of security features are implemented in the isolated security engines with examples including Converged Security and Manageability Engine (CSME).

Note Please note that by the time this book is published, some new security features may be released by Intel, and therefore please refer to Intel web site or contact the relevant OEM/ODMs for latest information.

Background and Terminology

Before the actual security capabilities can be described, it is important to understand the terminology, the threat pyramid, the relevance of end-to-end security, and Intel Security Essentials for leveraging built-in HW security technologies.

Assets, Threats, and Threat Pyramid

Security design begins with the process of identifying a set of assets that are to be protected and classifying these assets according to the different levels of protection based on strategic or other pertinent value

vectors. A real-life scenario of protecting assets in our home would be to protect our house keys (hang on wall), wallets (place in an enclosed cabinet), passports, and jewelry (in a safe in the master bedroom). For IoT deployments, security is also determined by the return on investment (ROI). Figure 3-2 depicts the relationship between them.

- Assets (A): Anything valuable to us that is worth protecting. What assets are we protecting? It is pertinent to classify the assets and prioritize. Example asset profile = {physical devices, internal fuses, keys, content, data at rest/in transit, etc.}
- Threats (T): What are we protecting against? Become aware of threat surfaces, the areas of exposure to threats.
- Vulnerabilities (V): What are the known weaknesses in the system that can be exploited?
- Mitigation: How are we going to protect?
- Robustness rules: Specific to assets/threats. Documented conditions/criteria for protecting specific assets against specific threats.
- Threat modeling: A process to evaluate the threat scenarios considering the vulnerabilities for specific assets. This process is iterative and is expected to be done whenever the bill of materials (BOM) list in a platform changes.

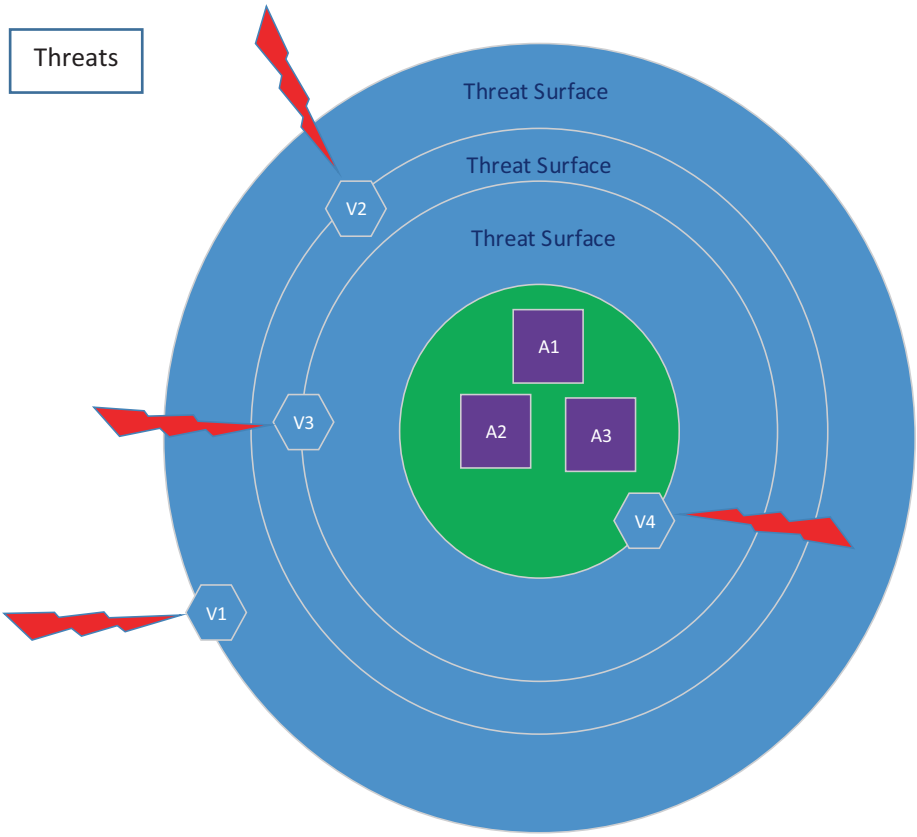

Figure 3-2. *Relationship between assets, vulnerabilities, and threats*

Inverted Threat Pyramid

The threat pyramid shown in Figure 3-3 depicts the surfaces/layers vulnerable to cyberattacks (both physical and remote) in an IoT device. The volume of attacks is high at the top and requires fewer resources, whereas the volume of attacks at the bottom is lower and requires a high amount of resources. In other words, the attack surfaces have varying degree of exposure and mandate a defense in depth approach at the platform levels.

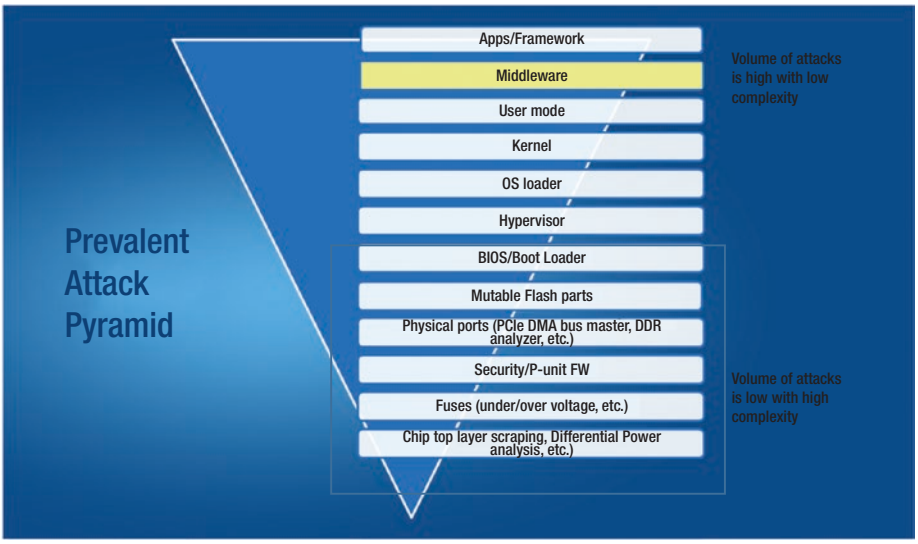

Figure 3-3. *Attack pyramid*

The rectangle outlines the IA value additions where the related security IP capabilities exemplify the assets that can be used to protect customer's assets. The effort to create exploits at the top of the inverted pyramid is low, and the ROI on the compromised assets is also low. Due to this low effort, the number of exploits is also significantly higher. As we traverse down the inverted pyramid, the effort it takes to create exploits increases significantly along with the cost, and thereby the number of exploits is typically lower and targeted in nature. The bottom six layers could be qualified as HW, and side-channel attacks plus physical attacks are relevant. The discussion of such side-channel and physical attacks is outside the scope of this book.

Sample IoT Device Lifecycle

The IoT device lifecycle pertaining to security is complicated with security involved in every phase of an IoT device lifecycle (Figure 3-4). During the build phase, the security SDK/API is critical for simplifying the device

build. The provisioning/configuring phases would require tools that scale across different CPU families and involve assigning a persona to the IoT device. The deployment phase should be flexible for seamless and potential anonymity. The connectivity should comply with the relevant security standards and specifications. The management of these devices must be secure and seamless. The retirement or decommissioning phase is equally critical for an IoT device due to the integration of different assets/secrets from multiple vendors in the system. For a detailed supply chain interactions during the lifecycle, refer to the Secure Device Onboarding technology.[3]

IoT devices have different security needs as they go through their lifecycle (on average it is many years significantly more than traditional PCs). Security is pivotal to enable IoT devices and sustain those on the market. Each stage of the device lifecycle has its specific requirements, starting from providing what is needed for onboarding a device during the start of its life to security management functions that secure runtime operations. Intel has a critical role with enabling design-in the best practice HW security model with solutions and ecosystem relationships. Intel targets to enable security capabilities and solutions for each phase working with the ecosystem. Security is not one-off, it evolves along the lifecycle with each stage having unique needs. Best practices are required to secure the entire lifecycle.

[3] www.intel.com/content/www/us/en/internet-of-things/secure-device-onboard.html

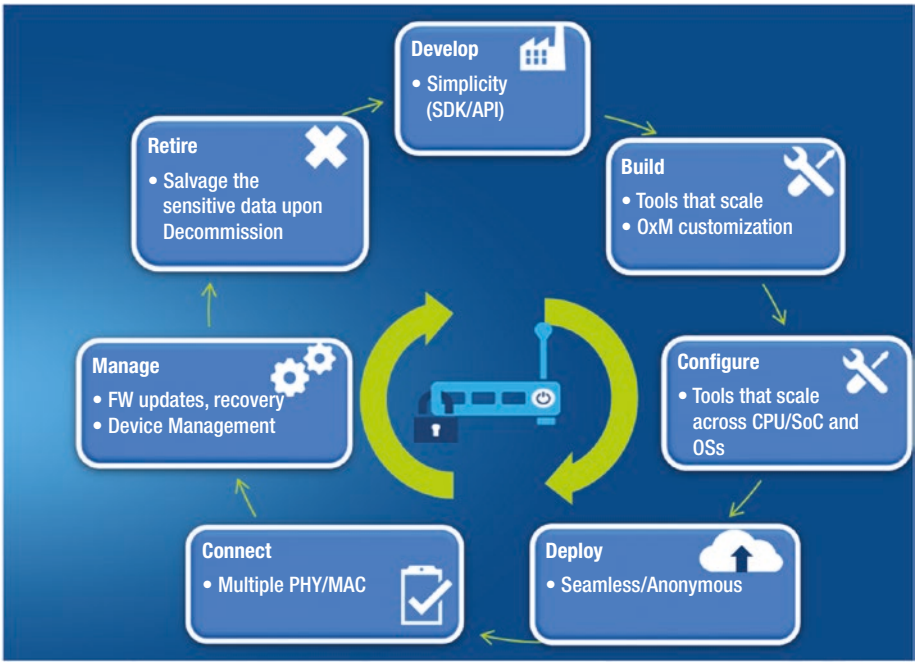

Figure 3-4. *IoT device lifecycle*

End-to-End (E2E) Security

While security pertaining to an IoT device is important, a practical IoT deployment warrants scaling security across an E2E spectrum starting with edge/Things connected to Network and then fog or Cloud. The typical E2E security involves edge/Things ➤ Gateway/Network ➤ Fog ➤ Cloud. Refer to Figure 3-5.

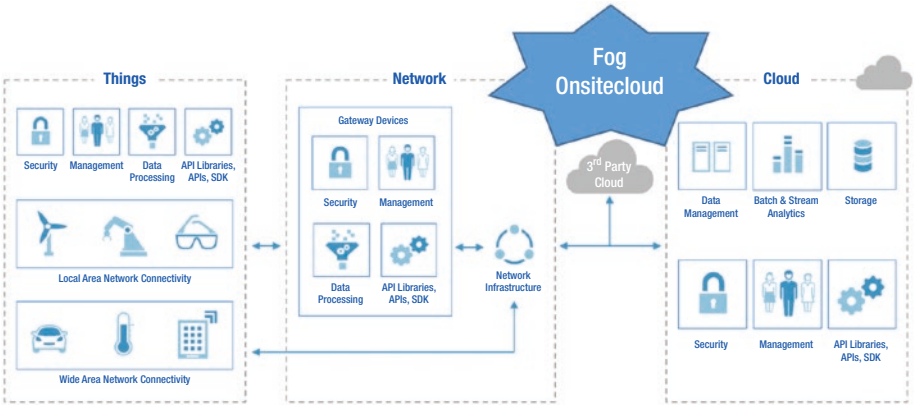

Figure 3-5. *Typical E2E security components*

Assets exist at different stages and often cross trust boundaries.

A typical flow (for a sensing application) is explained with confidentiality (encryption/decryption) and integrity (sign/verify) attributes:

1. The device securely identifies with the Gateway/Cloud (could be one time or periodic depending upon the policy enforcement).
2. The device has/interfaces to sensors (smart/dumb) and actuators, collects the data, and controls the sensors and drives the actuators.
3. Device may run some local analytics and optionally store the data encrypted.
4. Device encrypts or signs (or both) (depending on the policy) the data and sends it to Gateway.
5. Gateway decrypts/authenticates the data.
6. Gateway may run some local analytics.

7. Gateway encrypts/signs and sends the data to fog/Cloud.
8. The instances on fog/Cloud decrypt/authenticate the data.
9. Cloud applications run analytics.
10. Cloud applications encrypt/sign and store the data in databases.

Security Essentials

Security Essentials is an Intel brand initiative that defines a set of foundational security capabilities that Intel processors and Systems on Chips (SoCs) will support in order to establish a secure baseline upon which the ecosystem can build rich, secure usage models (see Figure 3-6). Security Essentials establishes a set of capabilities along with technology options for implementing each of the targeted capabilities. This allows us to project a common security posture across all supported platforms, establish a baseline for security that the industry can rely upon, and promote reuse and consistency in Intel-based security solutions. Intel provides training, collateral, technology summits, and Technology Alignment Programs with customers and ecosystem partners. In some cases, Intel partners with Independent BIOS Vendors (IBVs) and Independent boot loader vendors to enable the ecosystem with fast, secure, and functionally safe boot loader solutions.

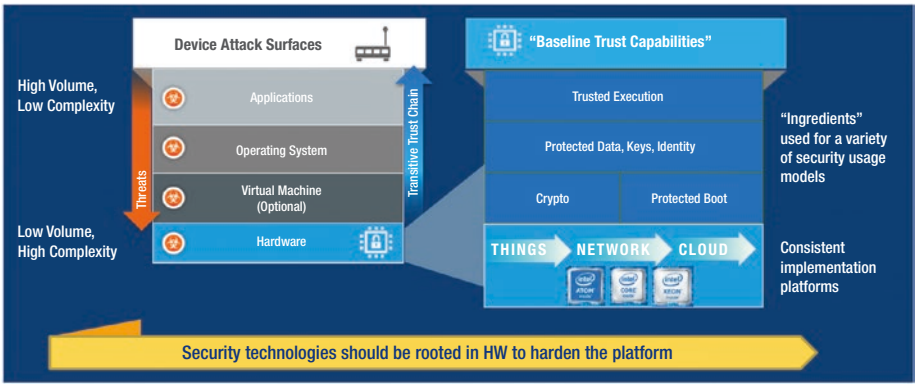

Figure 3-6. *Trusted secure foundation*

Security Essentials focuses on four buckets of capabilities: Device Identity, Protected Boot, Protected Storage, and trusted execution environment. These are later explained briefly.

Device Identity

A hardware identity refers to an immutable, unique identity for a platform. The identity has to be somehow inseparable from the platform. A hardware embedded cryptographic key, also referred to as a *Hardware Root of Trust*, can be an effective device identifier. The Trusted Computing Group (TCG) defines hardware-roots-of-trust as part of the Trusted Platform Module (TPM) specification. All TPM vendors are required to implement a hardware root of trust for storage. Intel® Platform Trust Technology (PTT) implements TPM functionality using a security engine integrated in many of its SoC products.

The IEEE community defines a device identity specification, IEEE 802.1AR, that has been adopted by the TCG. This means TPM-based device identity complies with interoperable and industry-accepted approach for secure device identity.

A software (SW) identity refers to a cryptographic fingerprint (SWFP) that describes important software that may execute on a platform. The SWFP can be reliably verified given a *whitelist* of SWFP values known to

be legitimate. SWFP is an important aspect of securely booting a platform where the goal of secure boot is to detect malicious changes to software images before they are loaded into memory.

The TCG defines methods for securely booting a platform where the SWFP of each software image loaded into memory is *measured* (aka cryptographically hashed) into a Platform Configuration Register (PCR), which is securely stored by a TPM. PCR measurements are available for comparison with whitelist values during the boot process and are available for *attestation* after the platform boots. Attestation is a protocol for proving to a peer platform that it booted a particular way. The attestation verifier might also use the whitelist to verify a peer platform node booted satisfactorily.

An IoT system that enforces a common and attested secure boot policy is a way to establish trust in a distributed set of IoT nodes. Distributed trust is an important component to establishing a secure IoT network.

Protected Boot

This capability defends against sophisticated bootkits and rootkits which have been demonstrated that reside in very early boot code and are able to launch a variety of attacks on the system. These attacks materialize without the knowledge of OS and thereby are invincible to be detected by the anti-malware entities. The TCG defines an architectural requirement for secure platform boot by defining a root-of-trust-for-measurement (RTM) where the platform must provide a secure platform reset and initial boot executive that is implemented in hardware, but TCG stopped short of defining a particular implementation.

The Unified Extensible Firmware Interface (UEFI) forum defines an interface where the UEFI BIOS boot image can be integrity verified by the RTM before it can execute, thereby ensuring the remainder of the BIOS boot process can be performed according to TCG defined secure boot principles.

Intel® TXT (Trusted Execution Technology) anticipates scenarios where a hard power reset, as a way to return to a trusted environment, is infeasible. Instead, Intel® TXT transitions the CPU to a secure operational mode using an IA instruction, then proceeds to boot a hypervisor or OS without invoking BIOS.

Intel Boot Guard is the hardware-based root of trust for system boot process. It provides an architectural enforcement of OEM boot policies and a protected initial measurement & verification of first OEM component. OEM boot policy is provided in FPF programmed by the OEM.

Protected Storage

The Storage Networking Industry Association (SNIA) defines storage security as

> *Technical controls, which may include integrity, confidentiality and availability controls that protect storage resources and data from unauthorized users and uses.*

Protected storage is a fundamental security capability required to support many other security capabilities. The Trusted Platform Module (TPM) implements secure storage primitives for several types of security objects including cryptographic keys, configuration registers, and whitelist values. Protected storage encompasses the following properties:

- Data confidentiality: Unauthorized entities cannot read the data.
- Data integrity: Unauthorized entities cannot modify the data or unauthorized data modification can be detected.
- Anti-replay protection: Unauthorized entities cannot replay/reuse stale data to storage.

Intel® Platform Trust Technology (PTT) is an implementation of the TCG Trusted Platform Module specification in a SoC that relies on hardware isolation of flash and other memory to prevent access outside of the TCG defined interfaces. Intel® QuickAssist Technology (QAT) is a hardware data encryption accelerator that also implements key storage protections. A common approach for building secure storage for data that exceeds the capacity of hardened secure storage resources calls for bulk data encryptions that allow ciphertexts to be stored on traditional storage media, but where encryption keys are stored in hardware. It is common to build a hierarchy of data encryption keys so that different access and lifecycle controls can be applied to different data. In some cases the key hierarchy itself is too large to fit into hardware-protected storage; therefore intermediate keys may be used to encrypt data encryption keys and so on until the top most keys of the hierarchy can be stored in hardware.

Trusted Execution Environment (TEE)

In general, a Trusted Execution Environment (TEE) refers to an execution environment that is isolated from the normal general-purpose execution environment. For example, the core CPU is a general-purpose execution environment, and a security coprocessor is an isolated environment. Trusted execution environments may include HW/SW/FW that establishes an isolated environment. By carefully controlling the infrastructure that produces the HW/FW/SW that implements it, the TEE can have strong guarantees regarding safe and reliable execution of TEE workloads. Typically workloads that involve the use of cryptographic keys to ensure confidentiality and integrity protection of data as it is transformed to and from ciphertext are performed using a TEE.

There are several TEE technologies available across a variety of architectures. ARM® TrustZone creates an isolated execution environment within the ARM core. Intel® Software Guard Extensions (SGX) takes a similar approach and allows multiple instances of trusted execution

environments for different applications and tenants. Intel® Converged Security and Manageability Engine (CSME) is a security coprocessor that is integrated into Intel chipsets. The CSME can be used to offload security-sensitive operations to shield them from possible attacks from the normal CPU environment. Intel® TXT allows trusted execution using CPU cache lines as RAM to minimize dependencies on external resources. It can be used for general-purpose TEE operations when cache coherency isn't needed. Intel® Virtualization Technology (VT) suite offers another form of TEE where a trusted hypervisor creates execution environments with distinct thread, memory, interrupt, and IO contexts. Virtualization allows full OS and application images to run which may be counterproductive to security due to increased attack surface of a large OS and application framework. Therefore, it may yet be appropriate to employ some other TEE capability in concert with virtualization.

Built-In Security

Built-in security features are essential to protect, detect, and correct the security issues in a platform. These features depicted in Figure 3-7 enable to protect the identity and data assets on the platforms from attacks, detect when attacks are launched, and then aid in deploying the corrective measures to make the platforms resilient.

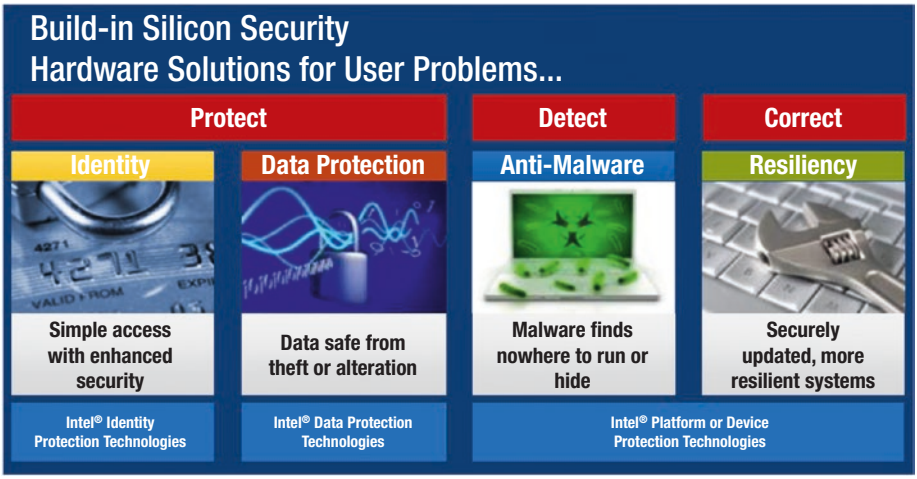

Figure 3-7. *HW solution pillars for user problems*

The identity is based on HW and possesses immutable properties and simplified access. The data asset protection includes data at rest and in transit. The detection mechanisms constitute anti-malware FW/SW components to find the malware and then pipeline into deploying the corrective measures via FW and/or SW over the air updates. Intel's value proposition includes three layers of ingredients as shown in Figure 3-8.

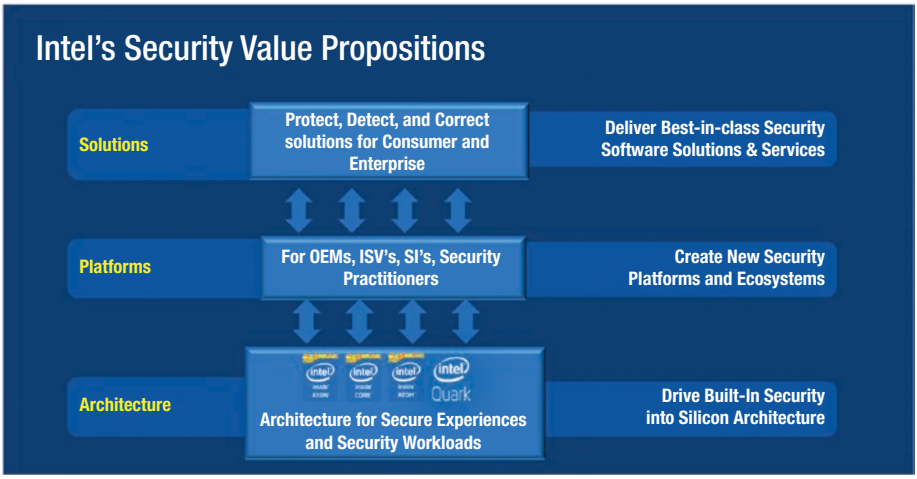

Figure 3-8. *Security value propositions*

At the bottom layer, the Intel Architecture allows leveraging built-in security features to build the platforms at the middle layer and, at the top layer, create ecosystems enriched with deployment of best-in-class security software solutions. These solutions at the top layer enable the protection, detection, and corrections in both consumer and enterprise class solutions. Intel security assets and solutions enable building and deploying an end-to-end system of systems as depicted later. The end-to-end system starts with edge devices or things on the left possessing minimal compute capacity and less robust security features; these edge devices are connected to Gateways/Network, to fog, and then connected to the cloud back ends.

The scalable strategy as shown in Figure 3-9 is to provide a minimally viable set of security capabilities that scale from low compute MCUs to atom class to Core and to Xeon server, microserver class products. Across the product lines, the four groups of security technologies are available in different capacities for implementing security features. The device identity based on HW is key for an IoT device, and protected boot ensures that only well-known FW/SW is being executed and protected storage ensures the storage of secrets and/or data securely. The trusted execution environment allows execution of code at runtime in an isolated and protected environment immune from SW and HW attacks.

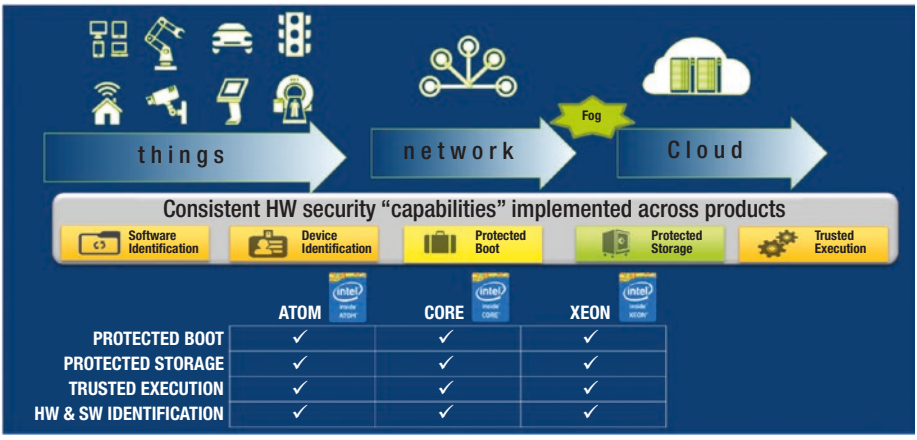

Figure 3-9. *Consistent HW security capabilities*

Base Platform Security Features Overview

Let's review the security features present in the base platform profiles of IA CPU/SOC at a very high level. As alluded to in previous sections, the security features are implemented in CPU and on dedicated security engines as shown in Figure 3-10.

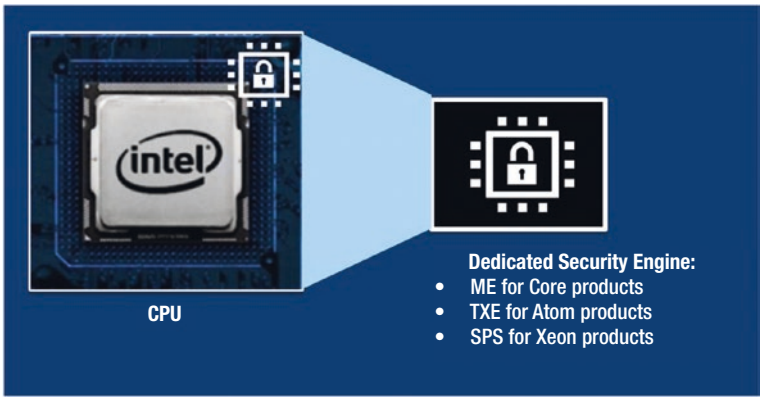

Figure 3-10. *CPU and dedicated security engines*

Intel CPUs come standard with a suite of cryptographic operations that can be performed on the main CPU. Secure, protected encryption starts with a random number seed, typically provided by a pseudorandom number generator within the client. Intel® Secure Key provides a clean source of random numbers through generation in hardware, out of sight of malware. Intel® SGX provides TEE with smallest TCB within the CPU boundaries for application to utilize.

CPU Hosted Crypto Implementations

These features include CPU new instructions for encryption/decryption, sign/verify, and random number generation: AES-NI, SHA-NI, SHA1 and SHA256, RDRAND, RDSEED, ECC. This section describes the Security features/primitives New Instructions (NI) as supported in the Intel CPUs

(as opposed to in an isolated security engine IP block). CPU crypto capabilities are supported by the CPU and the fabric. In the following sections, we will learn how the hardware-enhanced security strengthens Anti-Malware Defenses via the OS Guard (SMAP, SMEP), performing encryption/decryption, sign/verify, and random number generation. CPU security features and accelerators are available to trusted execution environments implemented by the CPU as well including Intel® SGX, Intel® VT, and Intel® TXT.

Malware Protection (OS Guard)

Intel CPU/SoCs expose HW features for OS to defend the platform against malware attacks. The particular and effective features include CPU new instructions to enable Supervisor Mode Execution Prevention (SMEP) and Supervisor Mode Access Prevention (SMAP). The SMEP feature prevents the code executing in privileged mode (ring 0) from executing code in application mode (ring 3). SMAP is a CPU-based mechanism for user-mode address-space protection and prevents supervisor accesses to data on user pages.

OS Guard (SMEP)

SMEP when enabled prevents a specific (important) privilege escalation attack vector which is supervisor mode execution from user pages. The OS can set CR4.SMEP to enable this feature, and no changes are required to applications or other OS software. However, there might be some compatibility issues with third-party ring 0 software. The changes in VMM are limited to supporting/virtualizing CR4.SMEP bit and corresponding CPUID bit. It is important to note the non-objectives so that platform-level protections can be deployed appropriately. SMEP doesn't prevent "all" privilege escalation attack vectors, nor does it prevent a specific class of vulnerability (e.g., buffer overflow).

OS Guard (SMAP)

SMAP extends the protection that previously was provided by SMEP and was developed with the Linux community, supported on kernel 3.12+ and KVM version 3.15+. The support depends on OS or VMM being used, and the CR4.SMAP has to be set to enable the feature. SMAP is analogous to SMEP (supervisor mode execution prevention) for data. There are legitimate instances where the OS needs to access user pages, and SMAP does provide support for those situations. Code executing in ring 0 (supervisor mode) is prevented from accessing the data in ring 3 (user mode). When/if CR4.SMAP = 1, CPU generates Page Fault (#PF) for the following accesses: accesses to data (not instruction fetch), data is on user-accessible page (U/S bit is 1 in all relevant paging structure entries), access is made with supervisor privilege which normally means CPU Privilege Level (CPL) < 3, applies also to supervisor accesses made with CPL = 3 (e.g., loads from GDT on segment loads). The resulting #PF establishes error code in the normal way.

Encryption/Decryption Using AES-NI

AES is a symmetric encryption standard that's widely used in the following use cases: full disk encryption, data in transit encryption, and enterprise application–specific security. All the modern compilers support the AES HW accelerators, and developers can also use via C/C++ intrinsics. Intel® Advanced Encryption Standard New Instructions (Intel® AES-NI) is a set of seven new instructions in the Intel® processor series. Four instructions accelerate encryption and decryption. Two instructions improve key generation and matrix manipulation. The seventh aids in carry-less multiplication. By implementing some complex and costly substeps of the AES algorithm in hardware, Intel AES-NI and PCLMULQDQ accelerate

execution of the AES-based encryption. The result is faster, more secure encryption, which makes the use of encryption feasible in new use-cases. Some of the properties are outlined here:

- Improve the compute efficiency of cryptographic algorithms.
- Vector AES is a promotion of AES-NI to vector form, enables two (256-bit) or four (512-bit) lanes, and increases AES throughput of cores.
- FIPS197 compliant.
- Compilers, libraries, and emulator platforms are all available now.
- AESENC, AESENCLAST, AESDEC, AESDECLAST.
- AES Encrypt Round, AES Encrypt Last Round, AES Decrypt Round, AES Decrypt Last Round.
- Instructions have both register-register and register-memory variants.
- AESIMC and AESKEYGENASSIST: Assist with AES Key Expansion, AES Inverse Mix Columns, and AES Key Generation Assist.

The platform support for AES can be determined by inspecting *cpuinfo* output and *openssl* commands as shown in the following:

```
$ grep -o aes /proc/cpuinfo
```

To verify the proper cipher order, use the following command:

```
"openssl ciphers -v"
```

See the following list that shows AES at the top of the list:

```
Openssl speed aes-256-cbc
Openssl speed –engine aesni –evp aes-256-cbc
```

http://ask.xmodulo.com/check-aes-ni-enabled-openssl.html

```
openssl  speed –elapsed aes-128-cbc
openssl  speed –elapsed –evp aes-128-cbc
```

https://software.intel.com/en-us/articles/improving-openssl-performance

Sign/Verify Using Intel® SHA Extensions

The Intel® SHA Extensions are a family of seven Streaming SIMD Extensions (SSE)–based instructions that are used together to accelerate the performance of processing SHA-1 and SHA-256 on Intel® Architecture processors (Figure 3-11). Given the growing importance of SHA in our everyday computing devices, the new instructions are designed to provide a needed boost of performance to hashing a single buffer of data. Using the SHA Extensions, the Intel® SHA Extensions can be implemented using direct assembly or through C/C++ intrinsics. The 16-byte aligned 128-bit memory location form of the second source operand for each instruction is defined to make the decoding of the instructions easier. The memory form is not really intended to be used in the implementation of SHA using the extensions since unnecessary overhead may be incurred. Availability of the Intel® SHA Extensions on a particular processor can be determined by checking the SHA CPUID bit in CPUID (EAX=07H, ECX=0):EBX.SHA [bit 29].

- New instructions in CPU to encrypt/decrypt data.
- The Intel® SHA Extensions are comprised of four SHA-1 and three SHA-256 instructions.

- There are two message schedule helper instructions each, a rounds instruction each, and an extra rounds-related helper for SHA-1.
- FIPS Pub 180-2 compliant.

Instruction	Op 1	Op 2	Op 3	Opcode
SHA1 New Instructions				
SHA1RNDS4	xmm (rw)	xmm/m128 (r)	imm8	OF 3A CC /r ib
SHA1NEXTE	xmm (rw)	xmm/m128 (r)	NA	OF 38 C8 /r
SHA1MSG1	xmm (rw)	xmm/m128 (r)	NA	OF 38 C9 /r
SHA1MSG2	xmm (rw)	xmm/m128 (r)	NA	OF 38 CA /r
SHA256 New Instructions				
SHA256RNDS2	xmm (rw)	xmm/m128 (r)	<xmm0> (implicit)	OF 38 CB /r
SHA256MSG1	xmm (rw)	xmm/m128 (r)	NA	OF 38 CC /r
SHA256MSG2	xmm (rw)	xmm/m128 (r)	NA	OF 38 CD /r

Figure 3-11. *SHA instruction family*

The availability of the SHA Extensions in a platform can be detected using the code in Listing 3-1. It is always a good idea to check the available HW crypto capabilities before leveraging them.

Listing 3-1. Detecting the SHA Extensions

```
int CheckForIntelShaExtensions() {

    int a, b, c, d;
    // Look for CPUID.7.0.EBX[29]
    // EAX = 7, ECX = 0
    a = 7;
    c = 0;
```

```
    asm volatile ("cpuid"
    Intel® SHA Extensions: New Instructions Supporting the
    Secure Hash Algorithm on Intel® Architecture Processors
    14
    :"=a"(a), "=b"(b), "=c"(c), "=d"(d)
    :"a"(a), "c"(c)
    );
    // Intel® SHA Extensions feature bit is EBX[29]
    return ((b >> 29) & 1);
}
```

Intel® Data Protection Technology with Secure Key (DRNG)

This section explains about the usage of Digital Random Number Generator (DRNG) with the new instructions supported in IA CPUs. For any IoT device, the ability to generate high-quality cryptographic keys is crucial. Two such instructions RDRAND and RDSEED are explained along with the method to determine the support and the associated programming usage. Intel® Secure Key constitutes the Intel® 64 and IA-32 Architectures instructions RDRAND and RDSEED and the underlying Digital Random Number Generator (DRNG) hardware implementation. High-quality keys for cryptographic protocols can be generated using the RDRAND instruction, and the RDSEED instruction is provided for seeding software-based pseudorandom number generators (PRNGs). RDRAND retrieves a hardware-generated random value from the NIST SP800-90A compliant Digital Random Bit Generator (DRGB) and stores it in the destination register given as an argument to the instruction. The size of the random value (16-, 32-, or 64-bits) is determined by the size of the register given. The carry flag (CF) must be checked to determine whether a random value was available at the time of instruction execution.

RDRAND is available to both OS modes: system (ring 0) or application (ring 3) software running on the platform. There are no hardware ring requirements that restrict access based on process privilege level. As such, RDRAND may be invoked as part of an operating system or hypervisor system library, a shared software library, or directly by an application. Before using the RDRAND or RDSEED instructions, an application or library should first determine whether the underlying platform supports the instruction and hence includes the underlying DRNG feature. This can be done using the CPUID instruction. In general, CPUID is used to return processor identification and feature information stored in the EAX, EBX, ECX, and EDX registers. For detailed information on CPUID, refer to References CPUID A and B. To be specific, support for RDRAND can be determined by examining bit 30 of the ECX register returned by CPUID, and support for RDSEED can be determined by examining bit 31 of the EBX register. A bit value of 1 indicates processor support for the instruction, while a value of 0 indicates no processor support. The Intel Digital Random Number Generator (DRNG) is a high-quality, high-performance, HW-based random number generator.

- It supports NIST SP 800-90 A, B, and C compliant functionality and is FIPS 140-2 Level 2 certifiable.
- It generates random numbers at a rate of 1 byte per clock.
- It is available early in the system boot/OS load process.

Both RDRAND and RDSEED return random numbers that are compliant to the US National Institute of Standards and Technology (NIST) standards on random number generators (Figure 3-12).

Instruction	Source	NIST Compliance
RDRAND	Cryptographically secure pseudorandom number generator	SP 800-90A
RDSEED	Non-deterministic random bit generator	SP 800-90B & C (drafts)

Figure 3-12. *NIST compliance for RDRAND and RDSEED*

As depicted in Figure 3-13, the RDRAND instruction is handled by microcode on each core. This includes an RNG microcode module that handles interactions with the DRNG hardware module on the processor chip. The entropy source (ES) produces random bits from a nondeterministic hardware process. HW AES in CBC-MAC mode distills the entropy into high-quality nondeterministic random numbers. The deterministic random bit generator (DRBG) is seeded from the conditioner.

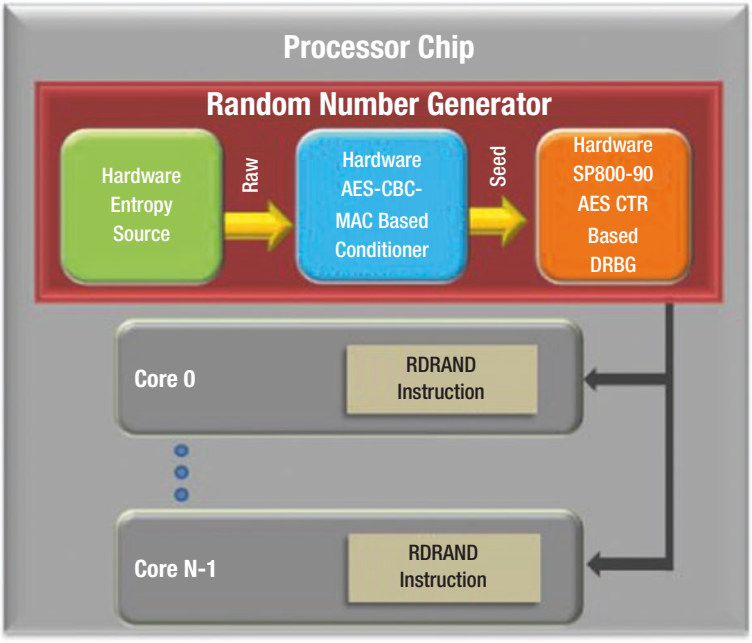

Figure 3-13. *Random number generator inside the chip*

The availability of RDRAND and RDSEED can be detected using the following register bit decoding (Table 3-1).

More information can be found at: https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide

Table 3-1. *Feature Information Returned in the ECX Register*

Leaf	Register	Bit	Mnemonic	Description
1	ECX	30	RDRAND	A value of 1 indicates that processor supports the RDRAND instruction
7	EBX	18	RDSEED	A value of 1 indicates that processor supports the RDSEED instruction

With the information from Table 3-1 and by leveraging the code in Listing 3-2, the availability of RDRAND and RDSEED can be detected in a platform.

Listing 3-2. Detecting DRNG Support

```
/* These are bits that are OR'd together */
#define DRNG_NO_SUPPORT 0x0 /* For clarity */
#define DRNG_HAS_RDRAND 0x1
#define DRNG_HAS_RDSEED 0x2
int get_drng_support ()
{
    static int drng_features= -1;
    /* So we don't call cpuid multiple times for
     * the same information */
    if ( drng_features == -1 ) {
        drng_features= DRNG_NO_SUPPORT;
```

```
        if ( _is_intel_cpu() ) {
            cpuid_t info;
            cpuid(&info, 1, 0);
            if ( (info.ecx & 0x40000000) == 0x40000000 ) {
                drng_features|= DRNG_HAS_RDRAND;
            }
            cpuid(&info, 7, 0);
            if ( (info.ebx & 0x40000) == 0x40000 ) {
                drng_features|= DRNG_HAS_RDSEED;
            }
        }
    }
    return drng_features;
}
```

One of the advantages of security hardening and acceleration capabilities applied to the core architecture is that performance enhancements derived from core silicon manufacturing process improvements also apply to security features. In many cases, this approach ensures security features' manufacturing costs scale with the other core features.

Converged Security and Manageability Engine (CSME)

This describes the Converged Security Engine capabilities including the silicon, FW, and SW ingredients. This is similar to a security coprocessor which has its own ROM, RAM, instruction set, and an isolated execution environment. Refer to a simplified architecture diagram in Figure 3-14. An excellent deep dive can be found in the book *Platform Embedded Security Technology Revealed* (www.apress.com/9781430265719).

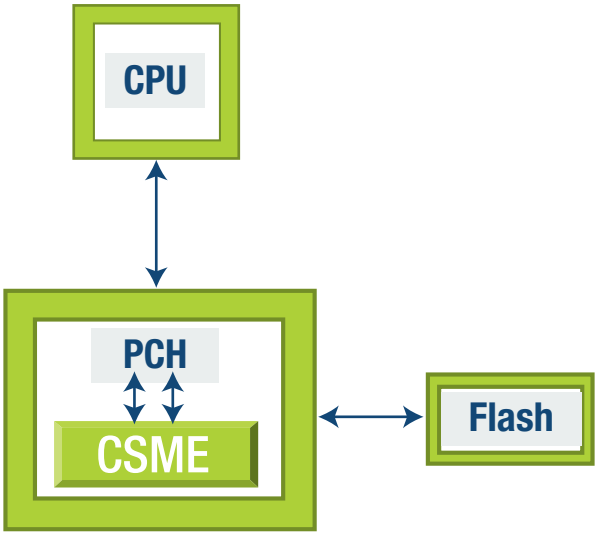

Figure 3-14. *CSME block diagram*

Features are implemented in the isolated security execution engine or equivalent to a security coprocessor. CSME is an embedded subsystem in Platform Controller Hub (PCH). It is a mini SoC within the PCH and contains a small processor, SRAM, crypto blocks, and I/O's. CSME serves three main platform roles: chipset (secure initialization/survivability), security (boot/runtime protection and enable trusted execution of platform applications), and manageability (optional extensions for out-of-band network management).

CSME supports the following:

- Crypto operations, boot, DAL, manageability (AMT, in above atom).
- The CSME supports crypto operations, HW Root of Trust–based secure boot (verified and measured), Active Manageability Technology, and other features.

- Content Protection: PAVP, Digital Rights Management (DRM)-Widevine, PlayReady, and Adobe Access. The CSME supports multiple DRMs for protecting the premium audio/video content by encrypting and/or digital watermarking.
- Secure Debug: DFX, JTAG lock. The CSME supports secure debug and manages access to DFX register space by allowing locking and unlocking of JTAG interface through which ICE emulators could be plugged in for debugging during pre/postproduction and to debug the field return parts.
- Identity Protection Technology: The CSME also supports protecting user's identity via multifactor authentication, biometrics, iris, and others.

Secure/Verified, Measured Boot and Boot Guard

Protecting the boot flow is critical to ensure that the device is not running compromised code whether it is the FW on the flash components or SW running from the mass storage device. Secure/verified boot is a process where a device authenticates the different FW/SW ingredients in the boot chain and establishes a chain of trust. Measured boot is a process where the device authenticates to a network for admission. To implement measured boot, the device stores the hash values of the boot chain ingredients, and SW entities collect these values and transmit them to a server for attestation.

Trusted Execution Technology (TXT)

The TXT is prominent in the server and microserver domain where a comprehensive security strategy is employed including a Measured Launch Environment (MLE) and instrumented OS. More about this will be discussed in the "Runtime Protection – Ever Vigilant" section.

Platform Trust Technology (PTT)

PTT is a FW implementation of the Trusted Computing Group (TCG) Trusted Platform Module (TPM) and complies with the TPM 2.0 specification. This FW is executed on the CSME or CSE on atom platforms. This feature is the most important for an IoT device which has board-level constraints imposed by BOM cost and real estate. PTT is essential for measured boot and attestation mechanisms.

Enhanced Privacy ID (EPID)

The EPID allows a device to possess an immutable "privacy preserving platform identifier" – in many use cases, it isn't required that the particular instance of the CPU be known, only that the platform is of a particular class or origin. In these situations, trust can be established without sacrificing privacy. Through this immutable identity, more secrets can be provisioned in the field during the course of the IoT device lifecycle including anonymous identification for provisioning of secrets, premium content, DRMs, and operation.

Memory Encryption Technologies

In future processors, Intel plans to introduce two new in-memory data protection capabilities including Total Memory Encryption (TME) and Multi-Key TME, or MKTME. TME technology encrypts the platform's entire memory with a single key.

TME

When enabled via BIOS configuration, this will help ensure that all memory accessed from the Intel CPU is encrypted, including customer credentials, encryption keys, and other IP or personal information on the external memory bus.

MKTME

The second new technology extends TME to support multiple encryption keys (Multi-Key TME, or MKTME) and provides the ability to specify the use of a specific key for a page of memory. This architecture allows either CPU-generated keys or tenant-provided keys, giving full flexibility to customers. This means virtual machines (VMs) and containers can be cryptographically isolated from each other in memory with separate encryption keys, a big plus in multitenant cloud environments. VMs and containers can also be pooled to share an individual key, further extending scale and flexibility. This includes support for both standard DRAM and NVRAM. Refer to the following for more information.[4, 5]

Dynamic Application Loader (DAL)

DAL technology allows building, deploying, and managing the lifecycle of a small trusted applet (Java-based applets) using the DAL SDK and Runtime environment.

[4] https://software.intel.com/en-us/blogs/2017/12/22/intel-releases-new-technology-specification-for-memory-encryption

[5] https://software.intel.com/sites/default/files/managed/a5/16/Multi-Key-Total-Memory-Encryption-Spec.pdf

Software Guard Extensions (SGX) – IA CPU Instructions

SGX constitutes a new set of CPU instructions, kernel/user mode drivers and Runtime environment, and API/SDK. This framework allows developers to build the trusted parts of the application code into enclaves. The inherent assumption is that the partition of the application into trusted and untrusted domains is already done prior to implementing SGX. SGX can be used to seal legitimate software inside an enclave to protect from attacks by the malware, irrespective of the privilege levels whether it is ring 0 or ring 3.

Identity Crisis

With the projected 50 billion IoT devices on the network, wouldn't it be ultracritical to ensure that a device is talking to the right device at the other end? A masqueraded device can do lot of damage. A method to prevent this is to implement a device identity that's immutable and use this identity to attest and provision initial secrets and additional secrets in the field during the course of the device's life. The same phenomenon applies to human identity as well. It is vital to realize that a masqueraded device is substantially hard to detect and quarantine. Intel Identity Protection Technology (IPT) uses Dynamic Application Loader (DAL) to implement mechanisms to protect the user identity via multifactor authentication and others.

The device identity (ID) decision tree can be used to select the right ID for a particular implementation. As shown in Figure 3-15, a security architect/engineer can decide the right identity based on the platform requirements and use cases. If an identity is required but mutable (changeable), a SW identity may suffice, but immutable identity requires identity to be in HW. If this identity now has to be anonymous, select EPID, else the identity as supported in PTT/TPM may be adequate. The EPID's cryptographic properties are briefly explained in the following section.

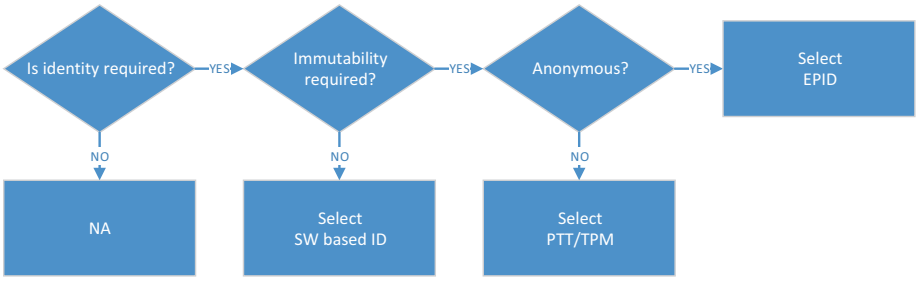

Figure 3-15. *Device identity decision tree*

Enhanced Privacy Identifier (EPID)

The EPID is a novel technology that addresses all aspects of the active anonymity problem: *authentication, anonymity, and revocation*. Intel® Enhanced Privacy ID (Intel® EPID) provides an immutable hardware root of trust, enabling IoT networks to confidently identify devices and to secure their communications.

Anonymity

Intel EPID also offers sophisticated privacy capabilities that enable anonymous communication to safeguard networks and customers' data. EPID is an anonymous digital signature scheme with the following attributes (Figure 3-16): a private key for signing and a single group public key for verifying signature of multiple keys. EPID is an open standard: ISO/IEC 20008/20009 and TCG Mature Technology, shipping since 2008, 2.4B keys since 2008.

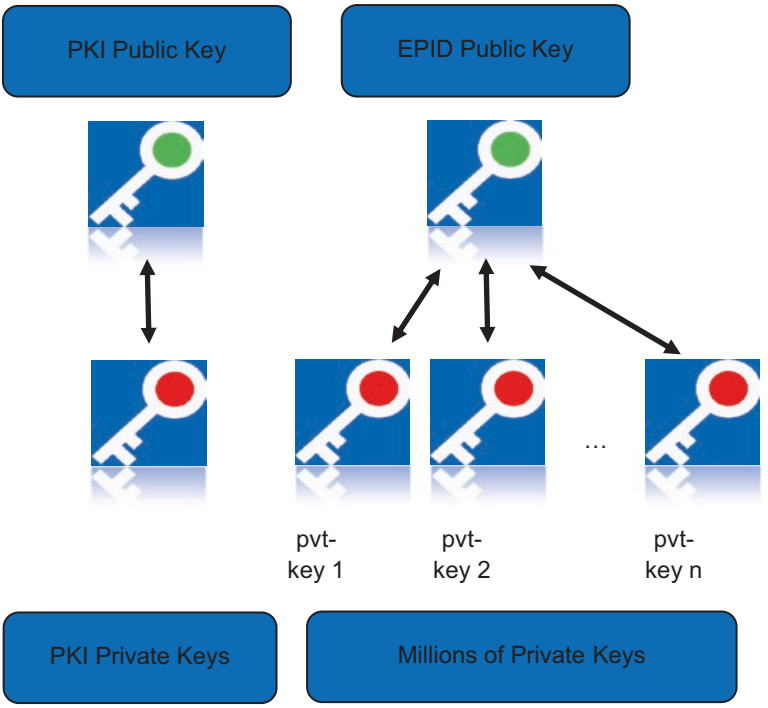

Figure 3-16. *PKI system vs. EPID*

As depicted in the figure, the PKI is a system with a public-private key pair, whereas the EPID is a system with one public key associated with many private keys formed into a group. In both cases, the private keys are provisioned into the devices, and the public keys are available to the back-end servers for authentication/admission.

PTT/TPM

The Endorsement Key (EK) supported in the Intel® PTT or discrete Trusted Platform Module (TPM) serves as a direct identity for IoT devices. An Endorsement Key is a special purpose TPM-resident RSA key that is never visible outside of the TPM. An EK certificate is used to bind an identity, in

terms of specific security attributes, to a TPM. The primary use of an EK certificate is to authenticate device identity during Attestation Identity Key (AIK) certificate issuance.

Device Boot Integrity – Trust But Verify

Imagine the IoT device booting an image that's not the original from boot storage. In this circumstance, any protections that you deploy at higher layers wouldn't be adequate to protect the device. Once the immutable identity is ensured as explained in the previous section, it becomes vital to follow through by booting securely. The boot loaders such as BIOS, UEFI, coreboot, and FSP can be classified into pre-OS boot loaders and will be referred as such. Let's unravel the ∗boot chaos with many terms employed in the industry today:

- Trusted Boot: Definition varies according to industry. Used to characterize a trusted system with a chain of trust.
- Secure Boot: HWRoT based. Authenticates starting with the first instruction executed on host (Core/Xeon/Atom).
- UEFI Secure Boot: UEFI Boot manager ensures device boots only signed FW and OS loaders. UEFI Driver signing and protocol extensions. This is also known as BIOS as Root of Trust.
- Windows Secure Boot: Leverages UEFI Secure Boot to continue into Windows OS, a Windows certification requirement.
- Direct Boot: An OS image such as Linux bzImage is loaded from stage 2 of the pre-OS boot loader.

- Verified Boot: Cryptographically verifies the Initial Boot Block of the pre-OS boot loader or UEFI or BIOS using boot policy key. A verified boot using Intel Boot Guard is shown in Figure 3-17.
- Immutable Root-of-Trust exists in the hardware.
- Root-of-Trust protects the initial boot process.
- It uses cryptographic keys to authenticate and validate the integrity of the Initial Boot Block (IBB).
- IBB maintains a secure boot chain by passing control to the next stage boot image after authentication and integrity verification.
- The final stage boot image passes control to the OS after authentication.
- Measured Boot: Measures the Initial Boot Block (IBB) and subsequent stages into platform storage such as Trusted Platform Module (TPM) or firmware-based TPM or secure storage.

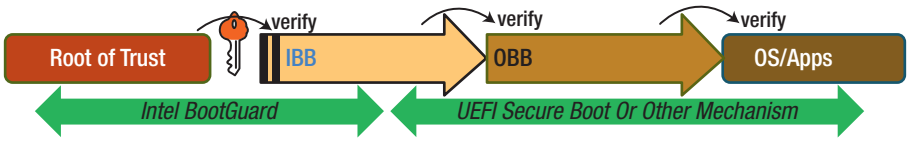

Figure 3-17. *Verified boot flow with Boot Guard*

The following terms will be useful to understand the following sequence that describes the process of Measured Boot using Boot Guard as shown in Figure 3-18:

- Hashing algorithms typically employed include Hash_alg = SHA1, SHA256, SHA384, SM3.
- Extending: It is a process of updating a PCR with a hash.

- PCR: Platform Configuration Register hosted inside PTT/TPM. The PCR 0–7 are used for pre-OS environment, and PCR 8–15 are used for OS and beyond. Refer to the TCG TPM specification for recommended PCR allocations.
- The new PCR value can be computed with PCR_new = Hash_alg(PCR_old || Hash_alg(data_new)).
- Logging: Keeps a log of all measurements in an ACPI table.
- ACM: Intel Authenticated Code Module, integrated in the BIOS/UEFI/boot loader for authenticating and measuring the IBB.

1. Upon power ON, CSME starts by computing the hash of ACM, and the hash of the ACM is stored in PCR 0.
2. The ACM computes the hash of IBB and extends it into PCR 0.
3. The IBB computes the hash of OEM Boot Block (OBB) aka the second stage pre-OS boot loader and extends the hash into PCR 0 and stores the hash of Platform Config Data into PCR 1.
4. The OBB computes the hash of OS loader and stores the corresponding hash into PCR 4. It stores the hash of Firmware Boot Policy in PCR 7.
5. The OS loader computes the hash of OS kernel and stores the hash into PCR 8.
6. The OS kernel can compute the hash of the user mode drivers/libraries and applications and extend the respective hashes into PCR 8-15 to meet the platform chain of trust requirements.

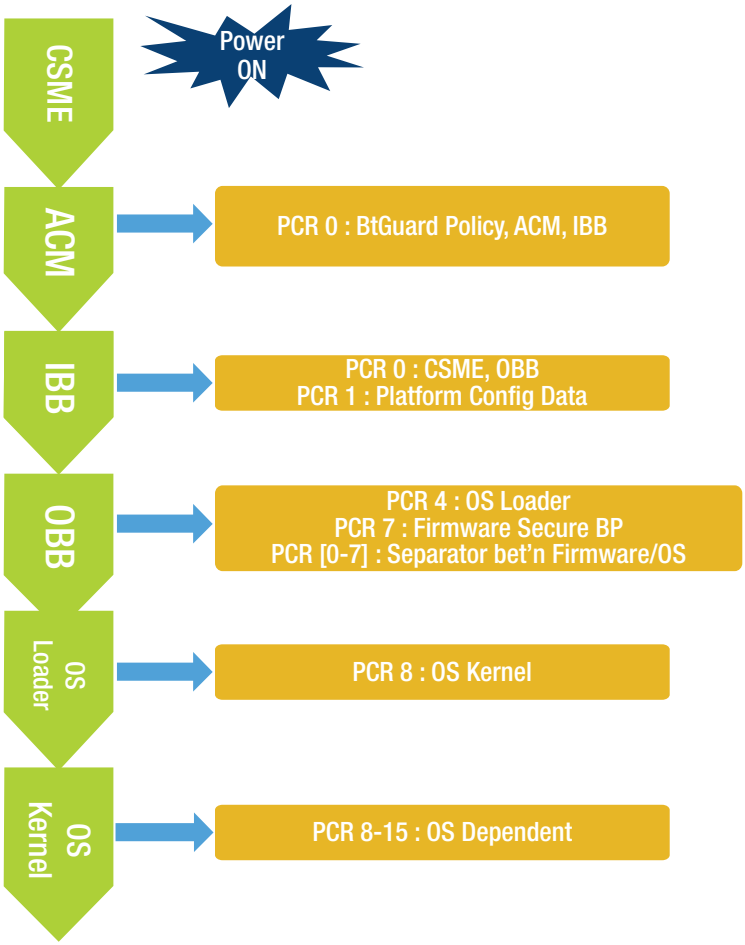

Figure 3-18. *Measured Boot sequence*

Secure Boot Mechanisms

The stack below describes the lowest layer to be the HW layer, and above that is the firmware layer which includes the modules required to handle the HW IP blocks and Digital Rights Management. Above that is the boot loader/UEFI used to initialize the CPU and chipset. The optional hypervisor supports the Virtual Machine Manager (VMM) functionality. The upper layers include the OS ingredients for kernel and User mode.

Above that layer are the middleware/frameworks and applications. This diagram (Figure 3-19) also illustrates the security goal that trust begins at the lowest layers and must be extended into the layers above – and that doing so requires conscious techniques to get it right. If/when those techniques fail, the stack recovers by falling back to lower layers.

The stack includes booting into application TEEs and the need to distinguish security-sensitive function and workloads that should be separated from "traditional" function and workloads. We can refer to the TEE and lower layers as the trusted computing base upon which the rest of the stack depends. The stack also supports networking and the idea that lower layers implementing the TCB can be linked (in an IoT use case) so that a Distributed TCB (DTCB) can be formed that supports distributed trusted workloads such as key management/migration, device management, SW/FW update of an IoT fog/network, and so on.

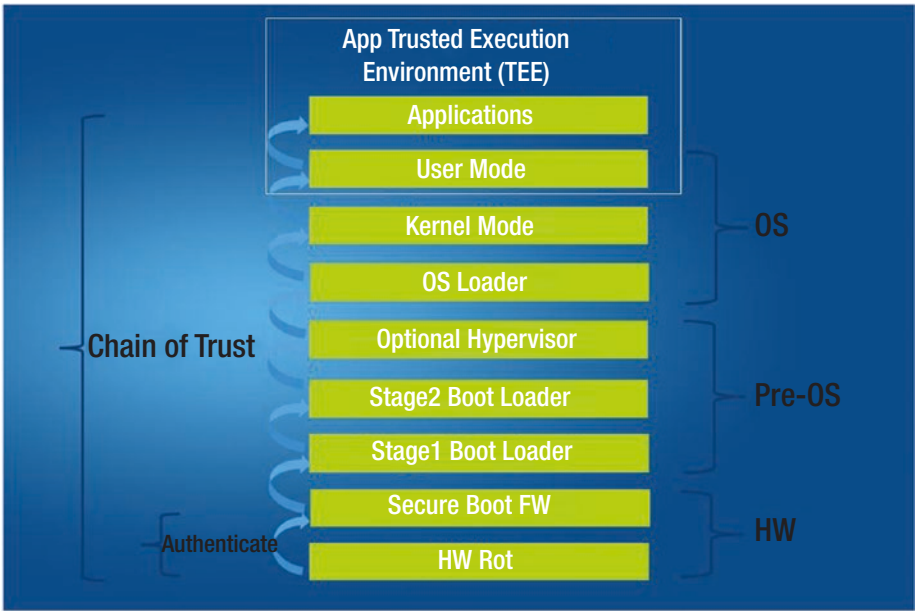

Figure 3-19. *Describes the boot flow on a core along with the chain of trust and signing implications*

Secure Boot Terminology Overview

Secure Boot Types: With the Field Programmable Fuse (One Time Programmable) profile options within the SoC, you can configure the device in an unsecured boot where the boot ingredients in stages are assumed to be trusted and no authentication is performed, referred to in Figure 3-20.

- Verified Boot: Boot policies are enforced during the boot process. Starting with the Root of Trust for verification, the currently executing module verifies the next module against a policy. The boot process is stopped if secure boot guarantee is violated. It is important to note that this only provides assurance that the boot policy was enforced.
- Measured Boot: Integrity measurement is placed into the TPM. Starting with the Root of Trust for measurement, the currently executing module places the integrity measurement of the next module into the TPM. Computer is *not* stopped if secure boot guarantee is violated and provable to remote systems via attestation.
- Secure Boot: A boot process which implements either Verified Boot, Measured Boot, or both. Verified Boot is often referred to as Secure Boot; Measured Boot is often referred to as Trusted Boot (also refers to TBoot sometimes).

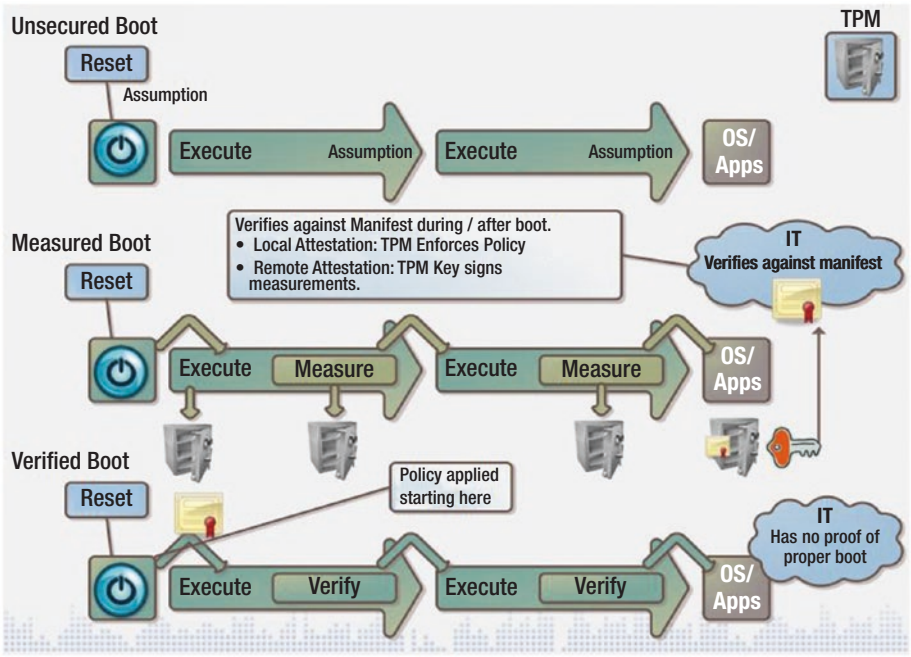

Figure 3-20. *Types of boot*

IOT devices are inherently vulnerable to physical attacks primarily due to their ability to connect to billions of devices. A first step in building a robust device is to ensure that the very first component of the boot loader is authenticated. This is implemented by a method known as secure boot which is based on a hardware root of trust in a platform. The immutable code running on on-die ROM in an isolated environment on a security engine forms an anchor. This ROM code loads the stage 1 of the boot loader into security engine's SRAM and cryptographically authenticates the image before executing it on the host CPU. The secure boot method on Intel Architecture is explained in detail including the HW and cryptographic blocks. Refer to Figure 3-21.

Overview of BIOS/UEFI Secure Boot Using Boot Guard Version 1.0 (BtG)

The verified boot flow using FSP+coreboot leveraging the Intel Boot Guard version 1.0 on Skylake platform is shown in Figure 3-21. The terms are explained followed by the sequence.

IPF: Infield Programmable Fuses also known as Field Programmable Fuses (FPF) represent storage inside the CPU/SoC for policy configuration and are One Time Programmable (OTP). The provisioning tools are provided by Intel for programming these fuses in the manufacturing flow.

Platform Power Sequence: Includes starting boot sequence for power rail stabilization.

Authenticated Code Module (ACM): Intel provided FW module loaded from flash, authenticated and executed in CPU's cache as RAM (CAR).

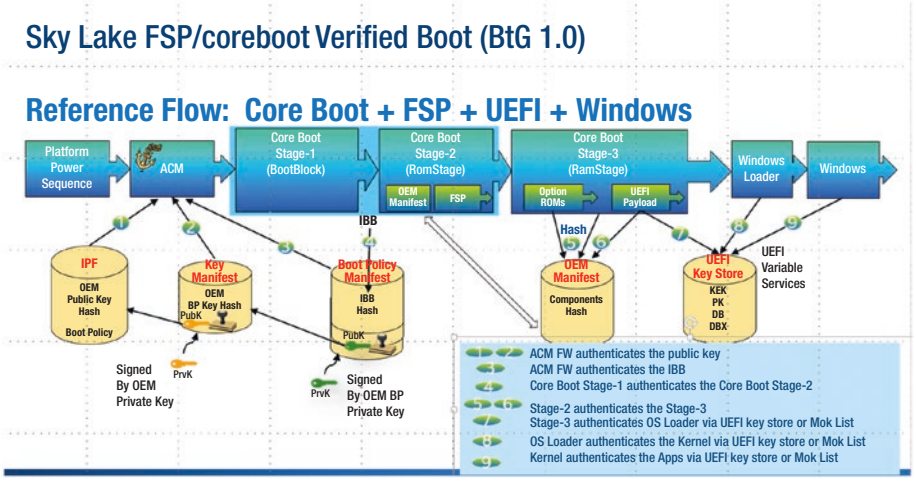

Figure 3-21. *FSP/coreboot-based verified boot on Skylake using Boot Guard 1.0*

The sequence is outlined here:

- ACM authenticates Core Boot Stage-1.
- Core Boot Stage-1: Authenticates Core Boot Stage-2 using the BPM.
- Core Boot Stage-2: Authenticates Core Boot Stage-3 using the OEM Manifest.
- Core Boot Stage-3: Authenticates OS Loader (Windows or Grub/ELILO or others).
- OS Loader (Linux or Windows or RTOS): Authenticates kernel image.
- Kernel: Authenticates the user mode and applications.

Refer to this link for starting with coreboot: www.coreboot.org/Lesson1

Firmware Support Package (FSP) is provided by Intel for initializing Intel silicon, designed for integration into a boot loader of the developer's choice. FSP source code can be leveraged for ideas and references for implementing verified and measured boot using Intel Boot Guard and PTT/TPM; more information can be found at: https://firmware.intel.com/learn/fsp/about-intel-fsp

Data Protection – Securing Keys, Data at Rest and in Transit

At rest: Storing data/secrets/content securely on the storage and whole disk encryption is the most popular example. This also is a very important problem. If a malware or even a legitimate application can access the secrets that it's not authorized, it causes an unstable device. Certain regulations such as General Data Protection Regulation (GDPR) mandate protecting the privacy of the data both at rest and in transit. For more information on encryption-related protection of data, refer to

https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-sme-obligations_en.pdf. Section (83) calls for encryption for confidentiality in: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679&from=EN

Article 6, 4 (e) also calls for encryption or pseudonymization of personal data which ensures reidentifying only with additional information. This is in contrast to anonymity where the anonymized data can no longer be reidentified.

Runtime protection problem: How do we protect the data and the code from each other in the system during Runtime? TEEs are an excellent method for this. Examples include SGX.

It is useful to think about theft threats and the idea that attackers are able to perform brute force crypto hacking as they have access to all the encrypted data and wrapped keys and so on. Encrypting using AES before storing the data on a disk makes it harder for attackers to reverse engineer and steal the secrets. An example use case for this is the Windows BitLocker technology which implements the whole disk encryption with strong passwords. There are increased threats due to persistent memory technologies supported by Optane and 3D Xpoint. These are persistent storage technologies making them subject to theft threats. Memory encryption is a mitigation where any/all data that goes out of the CPU/SOC on bus is encrypted whether it's destined for DRAM or SSD. The encryption technologies such as AES XTS 265 and secure boot existing in Optane + 3D Xpoint can be utilized to protect assets concerning flash-based memory.

Intel Platform Trust Technology (PTT)

Intel® PTT is a implementation of the Trusted Platform Module (TPM) 2.0 specification in firmware. CSME/TXE Engine is used as cryptographic processor for TPM implementation. SPI flash (TXE/CSME filesystem) is used as secure storage. PTT currently implements only mandatory and recommended TPM 2.0 commands mentioned in MSFT "signal and profile document."

As shown in Figure 3-22, the PTT includes random number generator, encryption/decryption, sign/verify, secure key generation, secure key/data storage, device identity both unique and anonymous, and device attestation.

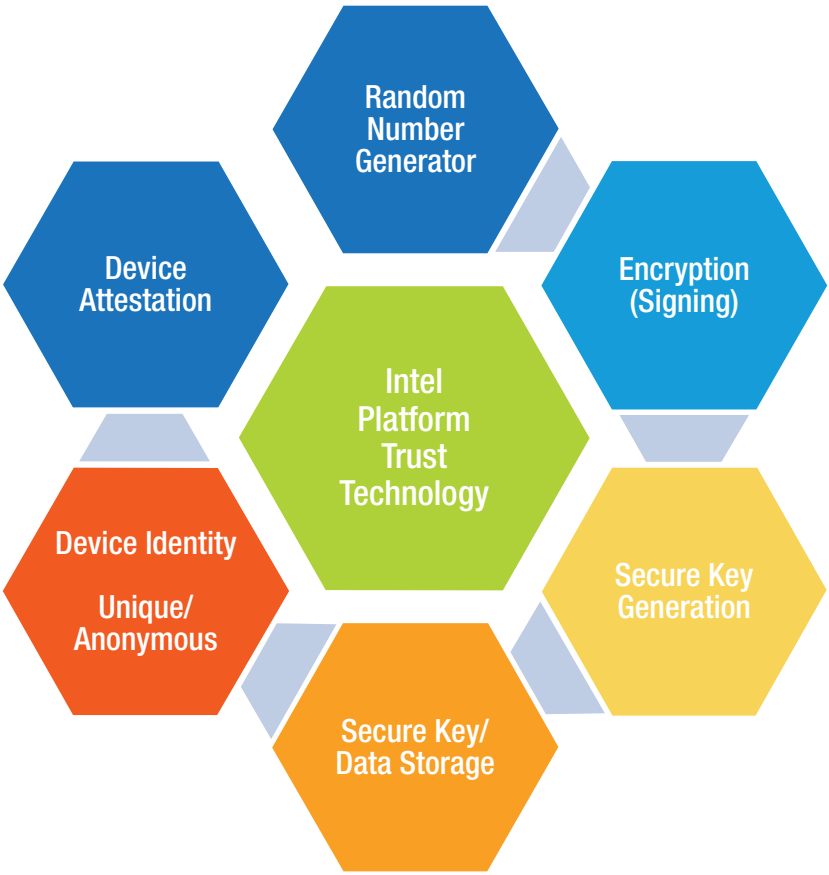

Figure 3-22. *PTT components*

Windows PTT Architecture

On Windows as shown in Figure 3-23, the host SW components include the Trusted Base Services (TBS), the TPM.sys kernel mode driver, and ACPI which interact with PTT FW through Memory Mapped IO (MMIO)–based

PTT interface. The PTT interface in turn calls into the TXE or CSME. The SPI storage is used as the secure storage where the keys and other secrets are stored encrypted and signed to ensure confidentiality and integrity. The CSME contains internal crypto engines and SRAM and uses SPI flash to store the keys in an encrypted format.

Pre-OS environment (BIOS/UEFI/coreboot) implements the following:

- Selects between available PTT/TPMs
- Enables/disables PTT/TPM
- Issues TPM clear (PPI)
- Logs measurements in TPM and ACPI for OS

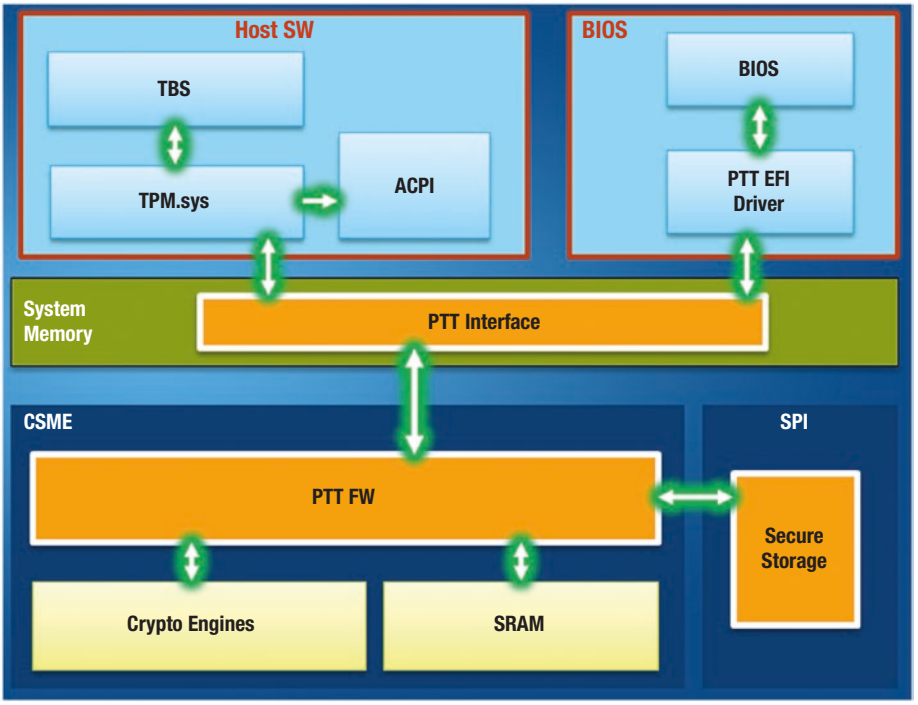

Figure 3-23. *Windows PTT stack*

Linux PTT Software Stack

As shown in Figure 3-24, in Linux OS stack, a PTT-based application has multiple mechanisms to interact with PTT including PKCS #11 and Feature API, and an expert application developer can directly interact with System API.

- TPM Device Driver (TDD) handles physical data transmission in ring 0/kernel mode.
- TPM Command Transmission Interface (TCTI) handles marshalling and unmarshalling of full TPM commands.
- System API (SAPI) enables creation and handling of TPM objects, sessions, and policies.
- Enhanced SAPI (ESAPI) enables management of the created objects, sessions, and policies.
- Feature API (FAPI) designed to capture 80% of common use cases combining operations with profile definitions.
- TAB controls access to the TPM in multiple application scenarios.
- RM manages the limited TPM resident memory.
- PKCS #11 – WIP on TPM 2.0.

TPM through SAPI specifications and implementations are mature, while ESAPI and FAPI implementations are still developing.

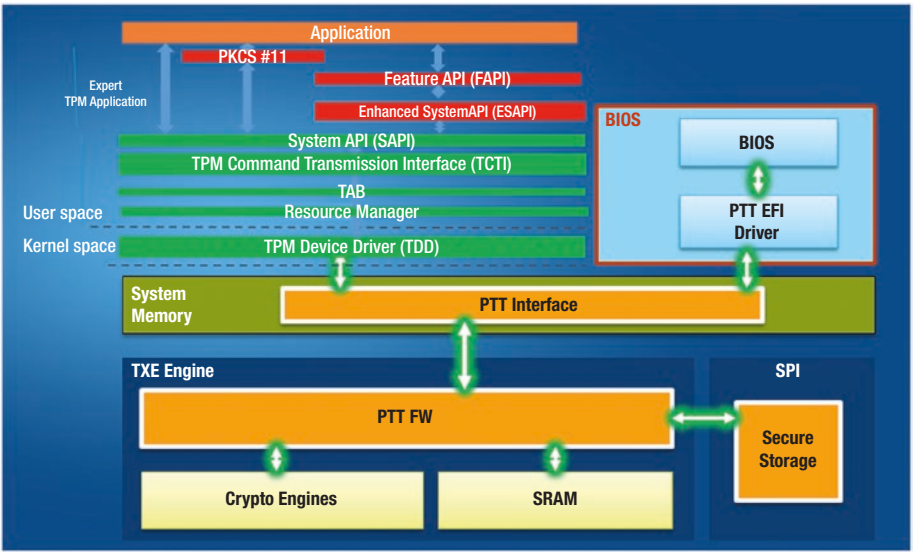

Figure 3-24. *Linux PTT stack*

Runtime Protection – Ever Vigilant

Most of the IoT devices spend their life in this phase where the device is functional and performing its intended persona. This phase is critical for devices that are "always on." The assets to be protected include data, code, and identity. Once the chain of trust is stable (secure booted), to maintain the stable chain of trust, every bit and byte must be authenticated before admitting into the system on every supported interface (USB, serial, BT/Wi-Fi). This objective can be achieved with high robustness level using a Trusted Execution Environment (TEE). The technologies available for implementing Runtime protections include Intel VT, SGX, CSME, and TXT.

Intel Virtualization Technology (Intel VT)

Virtualization abstracts hardware that allows multiple workloads to share a common set of resources. On shared virtualized hardware, a variety of workloads can colocate while maintaining full isolation from each other, freely migrate across infrastructures, and scale as needed.

CPU virtualization features enable abstraction of the full prowess of Intel® CPU to a virtual machine (VM). All software in the VM can run without any performance or compatibility hit, as if it was running natively on a dedicated CPU. Live migration from one Intel® CPU generation to another, as well as nested virtualization, is possible.

Memory virtualization features allow abstraction, isolation, and monitoring of memory on a per virtual machine (VM) basis. These features may also make live migration of VMs possible, add to fault tolerance, and enhance security. Example features include direct memory access (DMA) remapping and extended page tables (EPT), including their extensions: accessed and dirty bits and fast switching of EPT contexts.

I/O virtualization features facilitate offloading of multicore packet processing to network adapters as well as direct assignment of virtual machines to virtual functions, including disk I/O. Examples include Virtual Machine Device Queues (VMDQ), Single Root I/O Virtualization (SR-IOV, also a PCI-SIG standard), and Intel® Data Direct I/O Technology enhancements (Intel® DDIO).

Graphics Virtualization Technology (Intel® GVT) allows VMs to have full and/or shared assignment of the graphics processing units (GPU) as well as the video transcode accelerator engines integrated in Intel System-on-Chip products. It enables usages such as workstation remoting, desktop-as-a-service, media streaming, and online gaming.

Virtualization of security and network functions enables transformation of traditional network and security workloads into compute. Virtual functions can be deployed on standard high-volume servers anywhere in the data center, network nodes, or Cloud and smartly colocated with business workloads. Examples of Intel® technologies making it happen include Data Plane Development Kit (DPDK), Intel® QuickAssist Technology, and Hyperscan.

Intel® Virtualization Technology for Connectivity (Intel® VT-c) is a key feature of many Intel® Ethernet Controllers. With I/O virtualization and Quality of Service (QoS) features designed directly into the controller's

silicon, Intel VT-c enables I/O virtualization that transitions the traditional physical network models used in data centers to more efficient virtualized models by providing port partitioning, multiple Rx/Tx queues, and on-controller QoS functionality that can be used in both virtual and nonvirtual server deployments.

As shown in Figure 3-25, the isolation capability enabled by VT technology is being utilized to create an architecture with a Trusted Execution Environment (TEE). The TEE is implemented as a secure VM with privileged execution and access to resources; examples include Microsoft VSM and Trusty (`https://source.android.com/security/trusty/`).

Virtualization and VM Isolation components include Intel® VTx (CPU), Intel® VTd (I/O), VmFunc (Hypervisor).

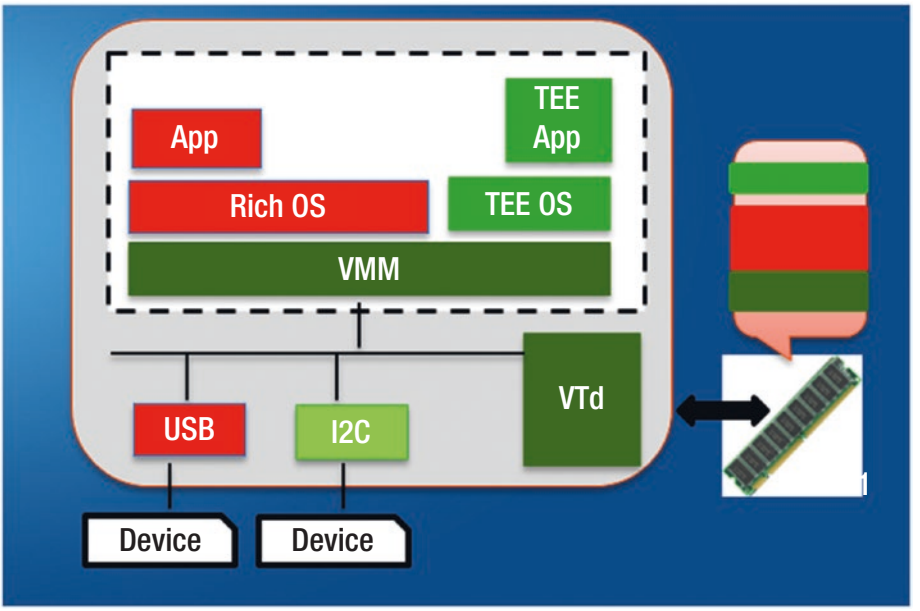

Figure 3-25. *TEE using virtualization environment*

TEE OS: Thin OS running alongside rich OS. Examples are Microsoft VSM, Android Trusty, and so on.

Rich OS: Regular OS that executes non-security-sensitive workloads. Examples are Microsoft Windows, Linux, Android, and so on.

Trusted computing base (TCB): VMM + TEE OS + TEE App.

Isolated execution: VMs are isolated from each other by the VMM.

Trusted Input/Output: Can be implemented by assigning I/O Controllers to different VMs.

Software Guard Extensions (SGX)

This section explains the usage of Software Guard Extensions (SGX) for implementing a Trusted Execution Environment (TEE) with the new instructions supported in IA CPUs. For any IoT device, the ability to execute code that handles secrets/assets in a protected environment is crucial. SGX leverages the partitioning of code into trusted and untrusted domains which interact with each other via well-defined SGX instructions.

How does SGX work as shown in Figure 3-26? The following model describes the interactions between the application and the SGX enclave.

1. Application is built with trusted and untrusted parts.
2. Application runs and creates enclave which is placed in trusted memory.
3. Trusted function is called; code running inside enclave sees data in clear; external access to data is denied.
4. Trusted function returns; enclave data remains in trusted memory.

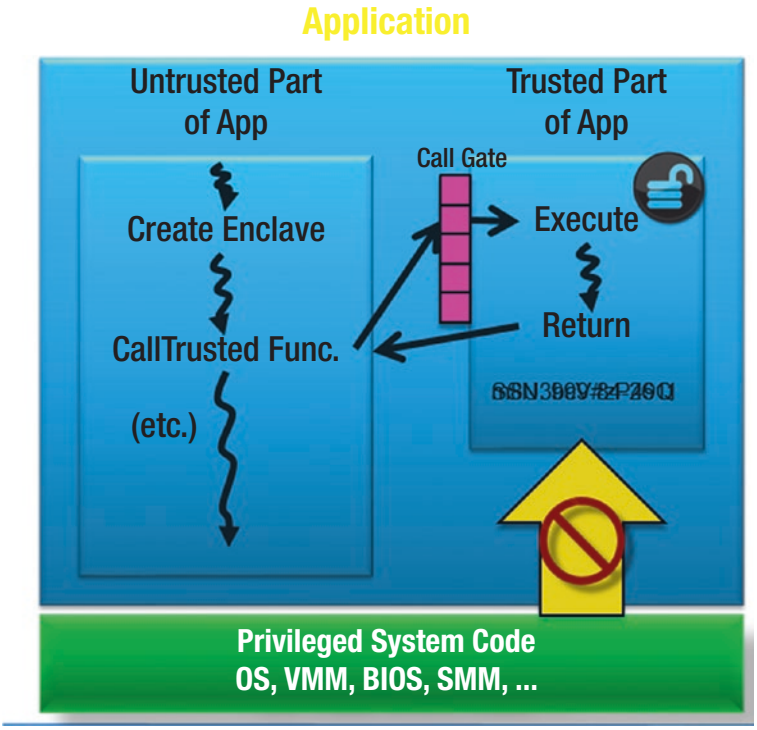

Figure 3-26. *SGX in action*

It is important to understand the software development model for the benefit of the developers (Figure 3-27):

- Sensitive code and data are partitioned into an "enclave" module which is a shared object (.so).
- Define the enclave interface and use tools to generate stubs/proxies.
- SGX Libraries provide APIs (C/C++) to encapsulate heavy-lifting implementation.
- Use a familiar toolchain to build and debug.

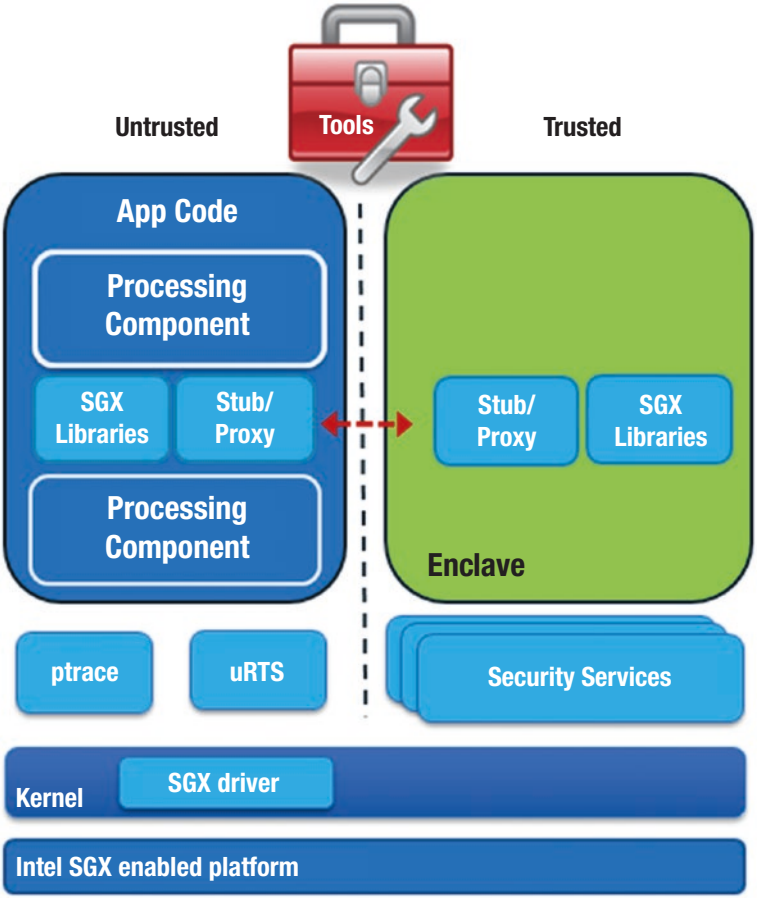

Figure 3-27. *SGX SW development model*

For further details, please refer to SGX web portal at: https://software.intel.com/en-us/sgx

Intel CSE/CSME – DAL

Intel Converged Security Engine in CSE/CSME is a dedicated engine for security and provides a HW root of trust for the platform. Dynamic Application Loader (DAL) exposes a general-purpose execution

environment and is in production use since 2011 (Sandy Bridge) and exists in almost every Intel-based platform. It extends the CSE FW by dynamically loading signed CSE applications at Runtime. It allows faster deployment of FW applications by decoupling the application development from the platform development lifecycle. The FW applications are stored on host filesystem, thus avoiding flash size considerations. DAL enables binary-level portability for applications and is based on a virtual machine; DAL applications are written in the Java programming language. Refer to Figure 3-28.

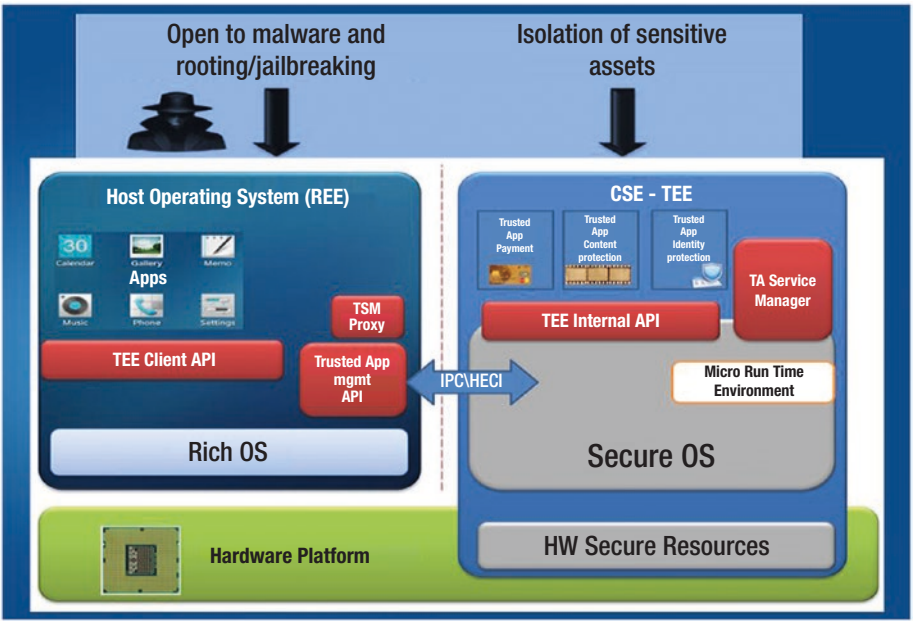

Figure 3-28. *DAL architecture*

Isolation from Rich Execution Environment

All the trusted applications (TAs) run in an isolated environment as supported by DAL and with the following attributes:

- TAs run in separate Java-like VM environment.
- TA-to-TA snooping is prevented using sandboxing.
- DAL prevents TA direct access to resources of other TAs.

Authenticity and Security

The DAL applications or TAs are subjected to the following robustness rules:

- DAL allows installation of signed and encrypted DAL TA in the CSE (security coprocessor).
- The TA can use the secure services, that is, secure storage to access SPI flash.
- Intel or OEM signed TAs can be installed.

Portability

The TAs have the binary-level portability subjected to the following scope:

- DAL is based on a virtual machine; DAL applications are written in Java.
- DAL enables binary-level portability for FW applications across the OS and HW platform.

Following are sample applications where DAL is deployed:

- Intel® Identity Protection Technology (Intel® IPT).
  - Identity protection and e-payment: OTP (one-time password), PTD (protected trusted display), PKI (public key infrastructure), NFC (near field communication).
  - Intel® PKI (PEAT) for IT market: Symantec Managed PKI, Intel IT.
  - McAfee (Intel Security): MFAb (Multifactor Authentication for Business), True Key – using IPT.
- Intel® Security Assist (ISA): A self-updater service which recommends security products to end users.
- China UnionPay (CUP): Implementing a Tap and Pay e-Commerce solution.
- Intel® Software Guard Extensions (Intel® SGX): The "Secure Enclaves" technology consumes CSME platform services using DAL.
- IOT Retail SmartPOS (Point Of Sale): Based on Atom platforms with Android.

Intel Trusted Execution Technology (TXT)

Intel® Trusted Execution Technology (Intel® TXT) provides hardware-based security technologies to help build a solid foundation for security. Built into Intel's silicon, these technologies address the increasing and evolving security threats across physical and virtual infrastructures by complementing Runtime protections such as antivirus software. Intel TXT also can play a role in meeting government and industry regulations and data protection standards by providing a hardware-based method of verification useful in compliance efforts.

As shown in Figure 3-29, Intel® TXT capable processors and chipsets allow establishing of the "root of trust" and "Measured Launch Environment" (MLE) to support trust decisions; within the computing platform, a MLE is needed. A "root-of-trust" is also needed which should be established first at the silicon level and then extended to the entire solution stack. The technology draws upon a rich set of security/virtualization features embedded into the IA processors and also integrated into the BIOS as well as other platform ingredients.

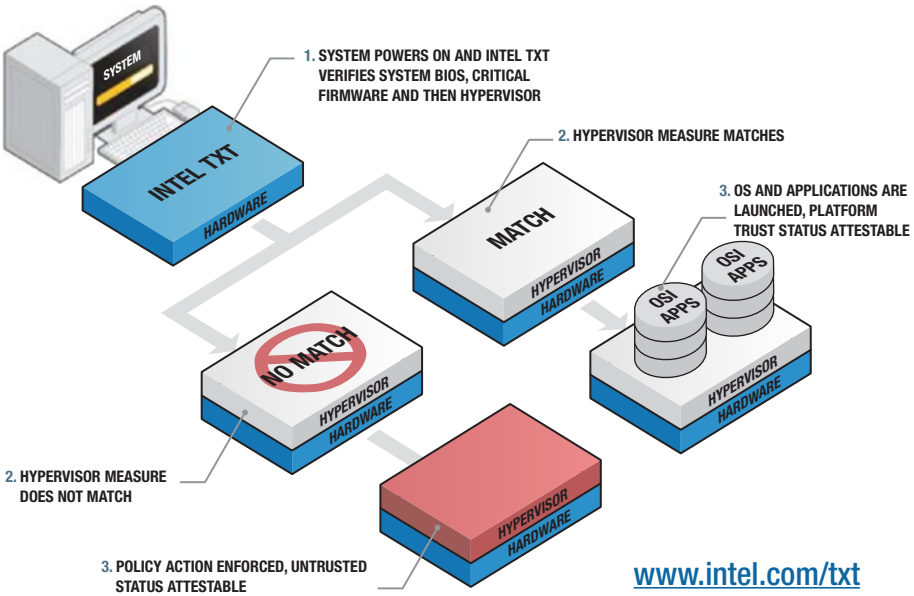

Figure 3-29. *TXT flow*

Figure 3-30 depicts the critical enabling requirements for the technology in server implementations. Intel TXT is specifically designed to harden platforms from the emerging threats of hypervisor attacks, BIOS, or other firmware attacks, malicious rootkit installations, or other software-based attacks. It increases protection by allowing greater control of the launch stack through a Measured Launch Environment (MLE) and

enabling isolation in the boot process. More specifically, it extends the Virtual Machine Extensions (VMX) environment of Intel® Virtualization Technology (Intel® VT), permitting a verifiably secure installation, launch, and use of a hypervisor or operating system (OS).

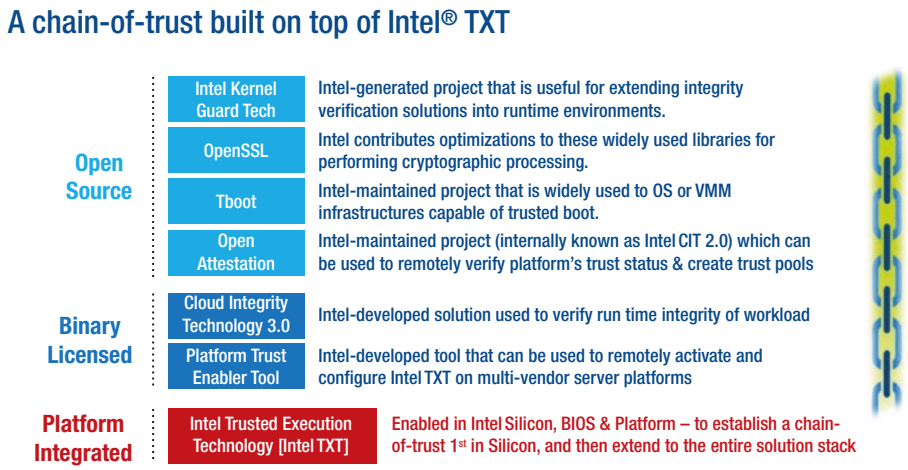

Figure 3-30. *TXT chain of trust*

Intel TXT gives IT and security organizations important enhancements to help ensure more secure platforms; greater application, data, or virtual machine (VM) isolation; and improved security or compliance audit capabilities. Not only can it help reduce support and remediation costs, but it can also provide a foundation for more advanced solutions as security needs change to support increasingly virtualized or "multitenant" shared data center resources.

Threats Mitigated

Intel assets as described earlier can be leveraged to improve the robustness and to defend against both zero-day and other attacks. Refer to Figure 3-31.

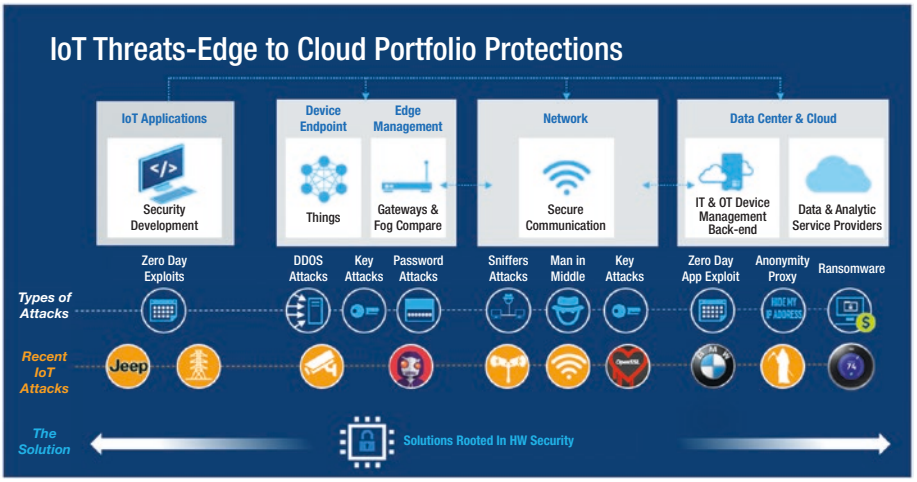

Figure 3-31. *Mitigation of IoT threats*

Zero-Day Attacks

Attacks that are designed to exploit a previously unknown vulnerability are referred to as zero-day attacks.[6] These attacks are harder to detect in time to minimize the damaging impact.

IoT applications: The impact of a compromise due to zero-day attacks can be minimized by handling all the high-value assets/secrets in a protected Runtime environment such as a TEE. DAL, SGX, and Trusty provide such defenses. Examples include remote car control in the jeep scenario and Ukraine power grid.

- Mitigation: Intel® Security Essentials, Intel Stratix® FPGA, protected boot, and attested software measurements can be implemented to mitigate the risks resulting from the preceding zero-day attacks. These solutions also enable a simplified TEE-based IP protection for ecosystem.

[6] https://csrc.nist.gov/glossary/term/zero-day-attack

Other Attacks

Other high impacting attacks include the distributed denial of service (DDOS), network attacks, and attacks on cloud infrastructures which hold rich troves of data.

Device Endpoint and Edge Management: The DDOS/key/password examples include CCTV Hijack and Mirai botnet.

- Mitigation: Intel® Secure Device Onboard can be deployed to mitigate the risks resulting from the preceding attacks. This is accomplished by not shipping devices with default credentials and instead use EPID identity designed-in for privacy preserving provisioning model to eliminate human misconfiguration with automated onboarding.

Network: Sniffers and man-in-the-middle examples include Tornado Siren Hijack, WPA CRACK, and Heart Bleed.

- Mitigation: Intel® Security Essentials API, Intel® Platform Trust Technology, Intel® Software Guard Extensions. Simplified HW secured key management and provisioning APIs. HW secured SSL transport APIs. PTT or TEE protected data and key storage.

Data Center and Cloud: Anonymity Proxy and ransomware examples include Infotainment VIN Online service app, Reaper, Thermostats, and WannaCry.

- Mitigation: Wind River Helix Device Cloud. Automated Over-the-Air (OTA) updates for firmware and software, provisioning, credential management, suspend, decommission, and firewall policy update to isolate/quarantine.

Conclusion

Security is not a blanket and requires pragmatic approach. It needs understanding of the assets to be protected against a set of threats in a system consisting of a set of vulnerabilities. Intel has a lot of HW security assets which can be leveraged to boot an IoT device securely and continue building on the chain of trust tethered to a HWRoT. Intel has security features residing in the CPU and PCH. The device identity, boot integrity, data protection, and Runtime protection are the four fundamental buckets of capabilities for securely booting into a TEE with a relevant TCB and later into a distributed TCB.

References

- https://software.intel.com/en-us/articles/intel-sha-extensions
- https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni
- www.intel.com/content/dam/doc/white-paper/enterprise-security-aes-ni-white-paper.pdf
- https://software.intel.com/sites/default/files/m/d/4/1/d/8/10TB24_Breakthrough_AES_Performance_with_Intel_AES_New_Instructions.final.secure.pdf

Security Hacks

- http://spectrum.ieee.org/cars-that-think/transportation/self-driving/hackers-take-control-of-a-moving-jeep
- http://spectrum.ieee.org/automaton/robotics/robotics-hardware/video-friday-bacteria-driving-robot-drone-with-gun-freaky-snakebot
- CPUID A: Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2: Instruction Set Reference, A-Z. [Online] http://www.intel.com/content/www/us/en/processors/architectures-software-developermanuals.html.
- CPUID B: Intel® Processor Identification and the CPUID Instruction. [Online] April 2012. http://www.intel.com/content/www/us/en/processors/processor-identification-cpuidinstruction-note.html.