CHAPTER 2

Intel's Embedded Solutions: from Management to Security

Security is, I would say, our top priority because for all the exciting things you will be able to do with computers—organizing your lives, staying in touch with people, being creative—if we don't solve these security problems, then people will hold back.

—Bill Gates

Teflon, the famous chemical, was discovered by Roy Plunkett of E. I. du Pont de Nemours and Company (commonly shortened to DuPont), in 1938 and trademarked in 1945. Teflon's major application today is in manufacturing nonstick cookware. However, it was not intended for helping grandmas make delicious pancakes when it was first discovered. For decades, it has been used in artillery shell fuses and the production of nuclear materials.

Temper foam was invented in 1966 by Chiharu Kubokawa and Charles A. Yost of NASA's Ames Research Center to protect astronauts' bodies when they are hurtling toward the earth. Today, temper foam is used to make mattresses that people sleep on every night.

The list of old inventions finding new applications in new domains goes on. The new applications benefit a much wider population and improve more people's quality of life.

When Intel's Active Management Technology (AMT) first appeared in 2005, it was marketed as an advanced system management feature for Intel 82573E series gigabit Ethernet controllers. In 2007, a new embedded coprocessor, namely the management engine, was introduced. Originally, the management engine was designed primarily for implementing the AMT rather than running security applications. At that time, the main problem that was supposed to be resolved by the embedded engine and AMT was the high expense and difficulty of system management by network administrators. The management engine was a component of Intel chipsets with vPro technology. The Intel AMT implementation was moved from gigabit Ethernet controllers to the management engine and became a feature of vPro.

Intel AMT is not the only application on the management engine. The first security application on the engine was the integrated TPM (Trusted Platform Module, see Chapter 7 for details). The number of security applications has been increasing in recent years with every release of the engine. In the latest releases, most applications running on the engine are related to security. The applications either realize "pure" security functionalities, or provide security infrastructures for other consumer features. For example, TPM and Boot Guard (refer to Chapter 6 of this book for details about Intel's Boot Guard technology) are security modules, whereas the dynamic application loader (DAL, see Chapter 9) is not implemented for security per se, but requires security as a building block.

In addition to more powerful applications and functionalities, the embedded engine has also been deployed on more platforms—not only chipsets for traditional personal computers, laptops, workstations, and servers, but also SoC (System-on-Chip) products, for example, in-vehicle infotainment, tablets, and smartphones, where security is becoming a critical infrastructure. The AMT is still widely provisioned on desktop computers and laptops, but has become an optional add-on for other mobile devices. On Intel's SoC platforms, the engine carries only security applications.

Just like Teflon and temper foam, today, the engine is realizing its greater value in the new usage model—providing robust security solutions and trusted execution environments to all forms of computer systems. The security and management engine is contributing to the promotion of people's computing experience every day and making a more substantial impact than ever before.

This book is not the first literature on the engine. Back in 2009, Intel Press published *Active Platform Management Demystified: Unleashing the Power of Intel vPro Technology*, authored by Intel's Arvind Kumar, Purushottam Goel, and Ylian Saint-Hilaire.[1] It will be referred to as the "2009 AMT book" in this chapter.

The 2009 AMT book is a systematic introduction to the management engine and AMT. It raises the platform management problems to be resolved, evaluates existing solutions, and then proposes the innovative AMT solution. It covers technical details of the management engine and the AMT, as well as instructions for setting up and configuring the AMT.

Although the engine's design has been improved in many ways since the 2009 AMT book was published, the fundamental architecture of the engine remains unchanged. A large portion of the technical descriptions in the 2009 AMT book still applies to today's security and management engine. Even after five years, it is still the best reference for infrastructures of the management engine and the AMT.

The remainder of the chapter is organized as follows. In the next section, we briefly revisit the 2009 AMT book. We will begin with a review of the hardware and firmware architectures of the management engine, and then look at the platform management problems and compare different solutions by analyzing their advantages and disadvantages. Next, a high-level introduction to the architecture of the AMT is presented. Finally, select security applications that feature on the security and management engine today are presented, with reasons for housing the applications in the embedded engine.

Management Engine vs. Intel AMT

What are the differences between the two terminologies, management engine and AMT? Do they mean the same thing?

The management engine refers to a computing environment consisting of dedicated hardware and firmware components. It has its own real-time operating system and hardware resources such as processor and memory. Just like a computer with Core CPU (central processing unit), applications can be installed and executed on the management engine. The applications are not generic software. They are implemented in firmware and designed specifically for running on the engine.

On the other hand, Intel AMT is a firmware application running on the management engine. It invokes the infrastructure and kernel application programming interfaces (APIs) provided by the management engine to build system management functionalities.

When the management engine was first introduced, Intel AMT was the primary application and it had attracted tremendous media attention. Hence some literatures use "management engine" and "active management technology" interchangeably. Today, although Intel AMT is still the most senior member of the application family, many new applications have joined the family and been deployed on the engine.

Intel AMT vs. Intel vPro Technology

Intel's vPro technology is a marketing name that covers a wide range of security and management features that are built in Intel processors and chipsets. The vPro technology resolves prevailing manageability, security, and energy efficiency problems with hardware-based protection, which is considered, when compared with software-based solutions, less vulnerable to threats such as viruses, worms, and hackers.

Many consider the AMT to be the essence of vPro. However, the vPro technology is comprised of not only AMT, but also other useful ingredients, such as:

- Intel Trusted Execution Technology[2] (TXT)
- Intel Virtualization Technology[3]
- Intel Identity Protection Technology[4] (IPT)
- Intel Anti-Theft Technology[5] (will be end of life in January 2015)

Besides AMT, some of these vPro ingredients also rely on the embedded engine to function. For example, IPT (refer to Chapter 10) and Anti-Theft.

Management Engine Overview

The management engine is made up of hardware and firmware. However, outside of its boundary, appropriate software drivers and applications must be installed on the host in order for the host to communicate with the embedded system through the dedicated host-embedded communication interface (HECI).

Hardware

The hardware is comprised of a processor, code and data caches, DMA (direct memory access) engines, cryptography engines, read-only memory (ROM), internal memory (static random-access memory, or SRAM), a timer, and other supporting devices. The devices are connected through an internal bus that is not exposed to the external world. This ensures independence, isolation, and security of the engine. The management engine's hardware devices are only accessible by the processor, the DMA engines, and the cryptography engine.

The hardware architecture is illustrated in Figure 2-1.

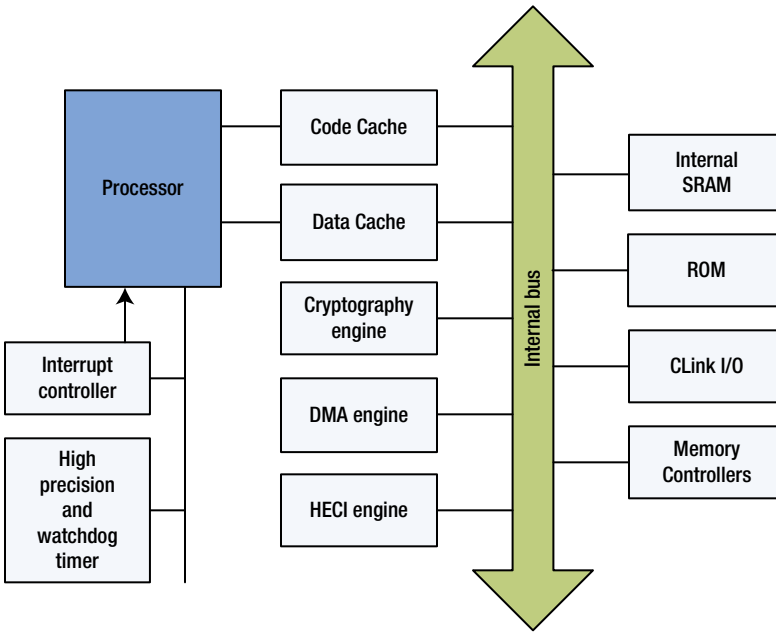

Figure 2-1. *Hardware architecture of the management engine*

Early generations of the management engine used ARC as the central processing unit. Other processors have replaced ARC in newer generations. The processor model and frequency in a specific engine depends on the form factor on which the engine is deployed. The model of the processor does not impact the engine's high-level firmware architecture.

There is a small code and data cache to help the processor reduce the number of accesses to the internal SRAM. The internal SRAM is the memory that stores firmware code and data at runtime. The capacity of SRAM varies depending on the product, but generally ranges between 256KB and 1MB.

In addition to the internal SRAM, the management engine also uses a certain amount of DRAM (dynamic random-access memory) from the main system memory. Code and data pages that are not recently accessed may be evicted from the SRAM and

swapped out to the reserved memory. When a page is needed again, it will be swapped in to the SRAM. During the boot process, the DRAM region that will be used by the management engine is reserved by the BIOS (basic input/output system) for the engine's dedicated access. The reserved region, by design, is not visible to the main host operating system. That being said, the management engine's security architecture assumes that the BIOS may be compromised and the local host may be able to read and write the reserved memory region. The size of the reserved memory varies from product to product, but usually in the range between 4MB and 32MB. This is only a small fraction of the DRAM installed on today's computing devices, and hence the impact to the main operating system performance is negligible.

For many embedded applications, it is necessary to transmit bulk data between the embedded memory and the host memory. However, the engine's processor cannot address the host memory. Therefore, dedicated DMA engines are introduced for moving data between the engine's memory and the main system's memory. Notice that the reserved memory is considered the engine's memory and not the host memory. When addressing the host memory, the DMA engines can only understand physical addresses and not virtual addresses that are specific to operating systems processes. The DMA engines can only be programmed by the embedded firmware running on the management engine. The DMA engines can also be used to move a large amount of data between two buffers of the engine's internal memory. Experiments show that, when data is greater than 1KB in size, it is more efficient to invoke a DMA engine for data copying than calling `memcpy()` of the processor. The firmware cannot program a DMA engine to move data between two host memory locations.

The cryptography engine device offloads and accelerates heavily-used cryptography algorithms so those resource-consuming operations can be performed faster and they do not occupy the processor's clock cycles. The algorithms implemented by the cryptography engine include AES (Advanced Encryption Standard), SHA (Secure Hashing Algorithm), DRNG (Deterministic Random Number Generator), big number arithmetic, and so on. See Chapter 3 of this book for a complete list of algorithms and their API descriptions. The cryptography engine is only accessible by the engine's firmware. They are not directly available to the host, although some embedded applications implement and expose external interfaces for the host applications to take advantage of the cryptography engine. Notice that the cryptography driver in the firmware kernel not only abstracts interfaces for the cryptography engine hardware, but also implements other cryptography algorithms that are not available in the hardware.

Overlapped I/O

As shown in Figure 2-1, there are three master devices—processor, DMA, and cryptography engine—on the management engine. They all can access the embedded memory and process data. These devices are independent of each other and therefore can function at the same time without mutual interference, as long as the assets (for example, memory and global variables) that are being accessed by more than one device are properly protected against racing conditions. The protection is usually realized by employing semaphores or mutexes. By commanding multiple devices to work simultaneously, firmware applications can be optimized to minimize the system

resource idle time and boost performance. The mechanism implemented by the security and management engine is de facto equivalent to overlapped I/O (input/output) or asynchronous I/O for traditional operating systems.

The idea is straightforward. After process A initializes a long cryptography operation, such as the exponentiation and modulo of RSA (a popular asymmetric-key cryptosystem invented by Ron Rivest, Adi Shamir, and Leonard Adleman) decryption, instead of sitting idle and waiting for its completion, the processor may switch to process B and perform operations that do not require the cryptography engine. In the meantime, the processor may either periodically inquire about the status register for completion of the RSA operation or watch for an interrupt signaled by the cryptography engine. Similarly, the DMA engines can also participate in the synchronization to further expedite the operations.

An interesting example of the overlapped I/O design is the flow for decrypting and parsing an H.264 video frame during movie playback. For this application, the player running on the host receives encrypted video frames from a remote content server, but the player as user-mode software is not allowed to access the content key or the clear content. The wrapped content key is sent to the security and management engine, which in turn uses its device private key to unwrap and retrieve the plaintext content key. The engine then decrypts the encrypted frames, performs slice header parsing, and sends back the resulting headers to the host. Finally, the player submits the encrypted frames and parsed headers to the GPU (graphics processing unit) through the graphics driver for playback.

Because of the limited memory capacity of the embedded memory, a large frame has to be split into chunks before it is processed. The optimal size of a chunk depends on how much embedded memory is available.

The firmware has three tasks in this usage:

1. Copy a chunk of an encrypted video frame from the host memory to the internal memory. This step is carried out by a DMA engine.
2. Decrypt the encrypted frame. For most cases, it is an AES decryption, offloaded to the cryptography engine.
3. Parse the clear frame. This step is conducted by the embedded processor.

The firmware runs the three steps repeatedly on all chunks of the frame, until the entire frame is processed.

A sequential approach would be to repeatedly exercise steps 1 to 3 for all chunks of a frame, respectively. The advantage is obviously simple firmware control logic. Figure 2-2 depicts an example of a frame that consists of four chunks. For simplicity, assume that the three tasks for a chunk— DMA copy, decryption, and parsing— take the same amount of time (denoted as one time slot in the figure). The number of time slots needed for processing a frame of n chunks is $3 \times n$. Processing all four chunks of the frame takes as many as 12 time slots.

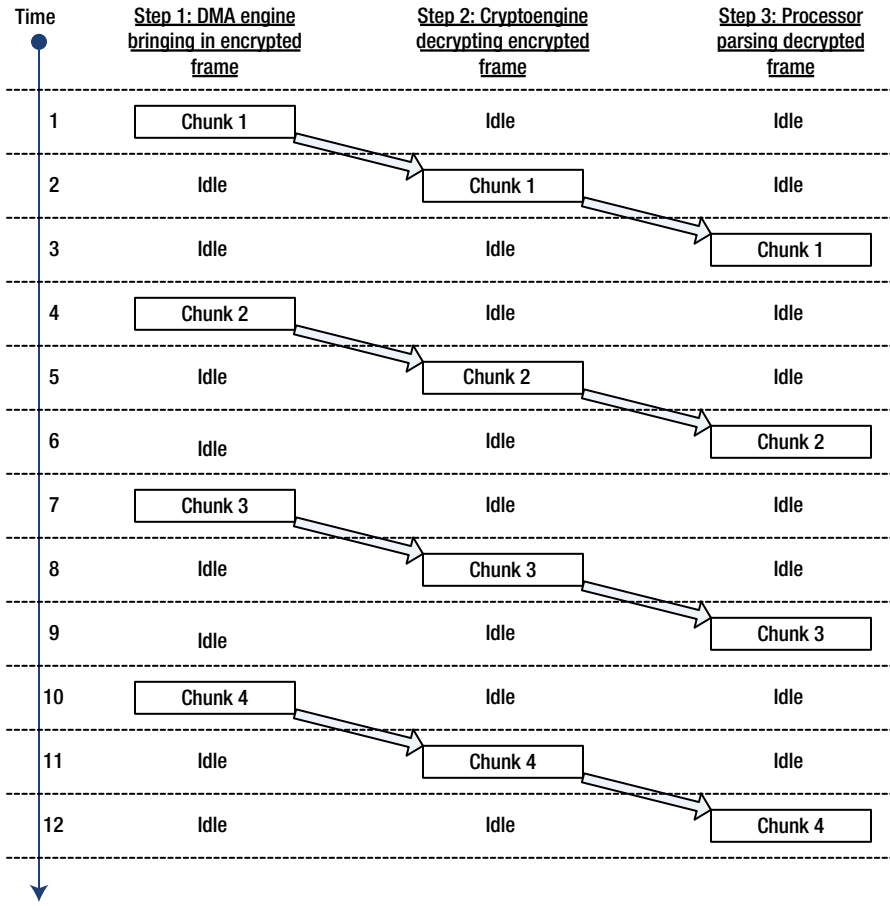

Figure 2-2. *Frame parsing flow without using overlapped I/O*

Obviously, the sequential approach lacks efficiency. In this design, when step 1 is running, the DMA engine is busy; however, the cryptography engine and the processor are both idle. Similarly, in step 2 and step 3, only one device is working at any moment and the other two are not being used.

To implement an overlapped I/O optimization, the firmware must simultaneously manage three chunks of the frame (namely: previous chunk, current chunk, and next chunk) of the same size in three distinct memory buffers.

The firmware first initializes DMA for the next chunk of frame, then triggers the AES decryption for the current chunk (the current chunk has been DMA'ed into the embedded memory in the previous iteration), and finally parses the previous (decrypted) chunk of the frame (the previous chunk has been DMA'ed into the embedded memory and decrypted in the previous two iterations). When the parsing is finished, the processor waits for the completion of the AES and the DMA. Figure 2-3 explains the flow graphically.

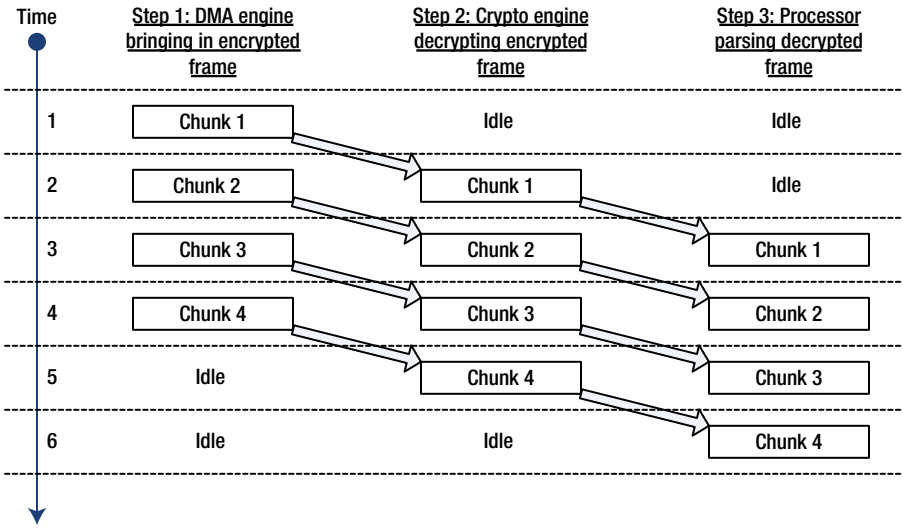

Figure 2-3. *Frame parsing flow using overlapped I/O*

It is easy to see from Figure 2-3 that processing four chunks takes only six time slots thanks to the overlapped I/O optimization. In general, the number of time slots taken for processing a frame of n chunks is $n + 2$.

Note that for the security and management engine, the processor, the DMA engines, and the cryptography engine all operate at the same speed. The exact frequency varies among different products. This is the major difference between the embedded overlapped I/O and its counterparts for the host operating systems, where the I/O devices, that is, hard drive, keyboard, and so forth, are usually operating at significantly slower speed than the main processor.

Admittedly, managing three masters may result in fairly complex firmware logic. The best practice for software engineering tells us that complicated code is more prone to bugs and errors. Therefore, such optimization strategies should be exercised with extra care. And the implementation must go through thorough testing and validation to cover all corner cases. For certain use cases, such as video frame parsing, as the throughput requirement is extremely high to guarantee smooth playback, utilizing the overlapped I/O trick is necessary.

■ **Note** If multiple master devices are available on the embedded system, consider overlapped I/O to improve performance.

Firmware

The security and management engine's embedded firmware implements the runtime operating system, kernel, and applications.

There are numerous products and form factors of the engine. A specific version of firmware is intended for running on the corresponding engine hardware only, and a specific engine is intended for running the corresponding version of the firmware; for example:

- Intel series 5 chipset (codename IbexPeak) can load only security and management engine firmware version 6.x. It cannot load version 5.x or other firmware. It cannot load firmware from a third party or a hacker.
- Security and management engine firmware version 6.x can only execute on the Intel series 5 chipset. It cannot be executed on series 6 or other chipset generations. It cannot be executed on SoC products, nor can it run on a third-party's or a hacker's hardware platforms.
- Security and management engine firmware designed for the Bay Trail tablets cannot execute on Intel chipsets or other generations of Intel tablets.

The hardware and firmware mapping is enforced by different image signing keys. The hash values of the signing public keys are hardcoded in the ROM on different products.

Figure 2-4 shows the high-level architecture of the management engine firmware.

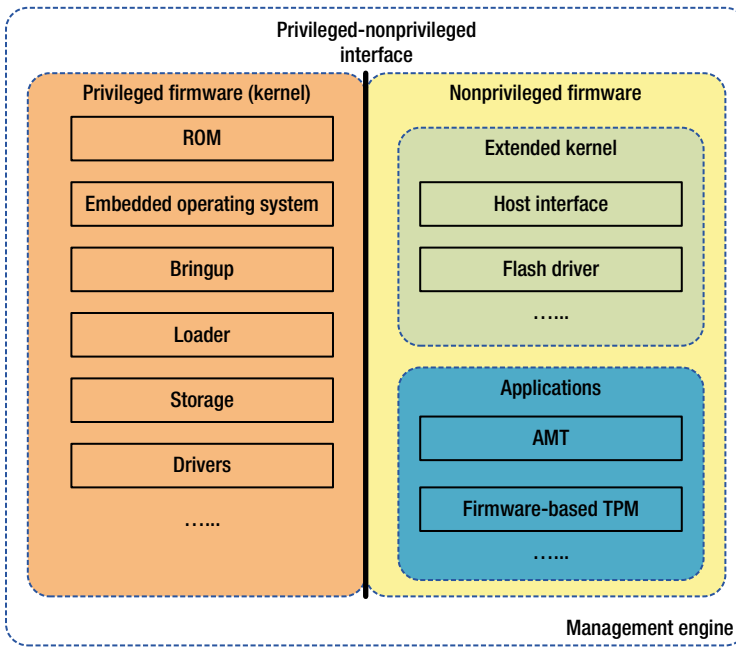

Figure 2-4. *Firmware architecture of the management engine*

There are two storage media—ROM and flash nonvolatile memory—that store the firmware's binary data and executable code. The ROM inside the management engine stores the boot loader. The code in ROM cannot be modified once manufactured. Thanks to this property, ROM is used as the root of trust of the engine. The boot loader code is usually smaller than 256KB.

The rest of the firmware is stored in flash. The flash is divided into multiple regions, for security and management engine firmware, BIOS, network controller, and so forth, respectively. Depending on which embedded applications are chosen to be included, the management engine firmware can consume from a few hundred kilobytes to 1.5 megabytes of flash space. The region for firmware is further divided into regions for executable code, configuration data, embedded applications' variable storage, and so on. The OEMs (original equipment manufacturers) are mandated to lock down the flash so it cannot be altered after the manufacturing process is completed. However, the management engine does not depend on the flash lockdown for security. The threat analysis assumes the flash can be replaced or reprogrammed by an attacker as he wishes.

As shown in Figure 2-4, firmware modules are logically divided into two categories: privileged and nonprivileged. The privileged firmware boots the engine, loads other modules, abstracts hardware devices (such as DMA engines and cryptography engines), schedules threads, manages synchronization objects (such as semaphores, timers, and mutex), and coordinates communications between embedded applications. The privileged firmware is the kernel and it implements only infrastructure for internal applications. It usually does not contain applications or expose external interfaces that are visible to the host.

The nonprivileged firmware is made up of one or more applications that realize their designed functionalities. The management engine firmware must contain at least one nonprivileged application. The Intel AMT, a nonprivileged module, is one of such applications. One notable difference that distinguishes the AMT from other applications is that the AMT also includes network stacks. Although most applications leverage the kernel for external communication, the AMT uses firmware wired and wireless network stacks for communicating with the remote managing console. As will be described later in this chapter, the firmware shares the same network devices with the host. The nonprivileged modules are further separated from each other by task isolation. The boundary between the privileged and nonprivileged domains is safeguarded by hardware and the privileged, to prevent privilege escalation attacks from the nonprivileged code.

Chapter 4 of this book provides a detailed introduction about the firmware architecture.

Software

Two classes of software programs run alongside the engine: drivers and user-mode applications.

The HECI is intended for transmitting a small amount of data between the host and the management engine firmware. The HECI is implemented as a circular buffer with limited bandwidth; therefore, the size of the data in general should be smaller than 10KB. The data transmitted through HECI can be commands for the firmware and the firmware's responses, but not massive data. The DMA engines should be used to move large amounts of data between host and firmware.

During the boot process, the BIOS can exchange messages with the firmware through HECI. On the host operating system, only ring 0 drivers may access the HECI device to send and receive messages. Together with the management engine firmware, Intel also releases HECI driver software for the HECI communication for various operating systems. The HECI driver is also called the *management engine interface (MEI) driver*. On Linux and Android, it is a device driver that supports the main kernel-based distributions.

Most firmware applications serve the role of trusted execution environments for the corresponding host applications. The firmware applications are typically used for handling sensitive secrets that must not be visible to the host and for offloading critical operations that involve the secrets. The software and firmware together realize specific functionalities. The software agents communicate with firmware applications through the HECI interface and DMA.

For example, a movie player application sends a 128-bit or 256-bit encrypted content key to firmware in a HECI message, and then the firmware uses the unique device key stored in the engine to decrypt the content key. Then the player sends another HECI command to initialize playback. Note that the device key must be securely provisioned to the engine beforehand and the device key must never be exposed to the host.

The software may also place bulk data, such as an encrypted video frame of over 1MB in size, in the host memory and notifies the firmware of the data size and the physical address thorough a HECI command. Upon receiving the HECI command, the firmware invokes its DMA engine to bring in the video frame from the host. Note that the embedded engine's DMA devices understand physical memory address only. Virtual memory must be converted to physical memory by a ring 0 driver before delivering to the firmware.

Platform and System Management

As defined in the 2009 AMT book, a *platform* is a computer system and all of its hardware components: motherboard, disk storage, network interface, and attached device, that is, everything that makes up the computer's hardware, including BIOS. On the other hand, a *system* has a broader definition. It includes both the software and the hardware of a computer.

Today, the concept of a "platform" for mobile devices should be extended to cover hardware that is not present in traditional computer systems. There is a long list of hardware that is commonly embedded in mobile platforms: GPS (global positioning system), cameras, sensors, fingerprint reader, and so forth.

The network administrator's responsibility is to make sure all computers in an enterprise are up and running normally. Even before the Intel AMT was invented, there were numerous manageability solutions available in the market to help network administrators do their jobs.

Software Solutions

There are several categories of manageability software. For example, firewalls analyze network data packets and determine whether they should be allowed or blocked, based on the rules and policies configured by network administrators. Antivirus software

detects and removes malicious software programs from the system. Remote desktop control agents such as VNC (virtual network computing) and SSH (secure shell) enable IT support technicians to remotely manage a system to perform diagnosis and resolve problems.

Although very convenient and useful in daily system management, software solutions also suffer from obvious limitations:

- *Dependability*: Manageability software runs in the operating environment that they are attempting to monitor, manage, and repair. When the operating system is not booting or not configured correctly, the software manageability solutions may fail to function.
- *Availability*: Manageability software is not able to perform management tasks when the system is in low-power states (sleeping or hibernating).
- *Reliability*: Manageability software is usually launched during boot and runs quietly in the background. However, it may be accidentally or intentionally turned off by end users or other system "clean-up" utilities.
- *Security*: Software solutions are naturally less trustworthy than hardware solutions. They are vulnerable to denial of service (DoS) attacks, may be compromised to report bogus information, or may even be hijacked and become a threat to other computers in the same network.

Hardware Solutions

In contrast to software solutions, hardware solutions for manageability do not depend on the operating system or software programs; hardware solutions can be functioning when the computer is in a low-power state; and hardware-based security measures can be applied if desired.

The KVM (keyboard, video, and mouse) is a representative hardware approach. In a typical KVM setup, the computer being managed is locally connected to a network KVM device, which connects the computer's I/O devices to a remote management console over the network. A network administrator can manage numerous computers from a single console simultaneously. Sitting in his office, the administrator can see the display of the computer being serviced and control its keyboard and mouse, as if he is sitting in front of the managed computer. Figure 2-5 is a symbolic representation of the management solution based on network KVM.

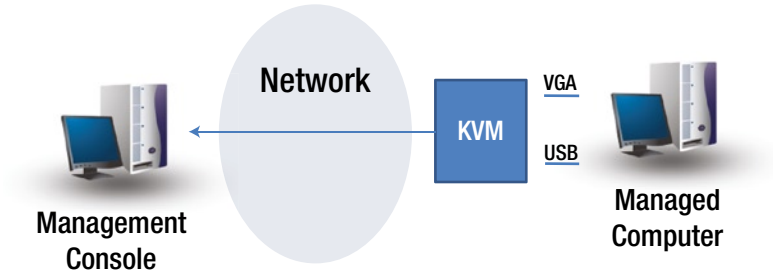

Figure 2-5. *Network KVM connected to a managed computer*

The equipment cost is the main factor that prevents the network KVM solution from being deployed on every computer. As can be seen in Figure 2-5, the KVM stands on the side of the computer; there must be a KVM device to support a computer (multiple computers physically located in the same location can share a multiport network KVM). The retail price of a 16-port network KVM ranges from a few hundred to over a thousand US dollars. This significantly raises the cost of network and system administration.

A more advanced hardware management solution is the baseboard management control (BMC). The BMC is a specialized embedded system that monitors various physical states, including, but not limited to, the temperature, humidity, or voltage of a computer or server. If a reported value strays out of the normal range, the administrator will be notified. A BMC combined with network KVM can realize very powerful management functionalities, including remotely power cycling, seeing displays, and controlling the keyboard and mouse. See Figure 2-6 for a symbolic representation of the BMC.

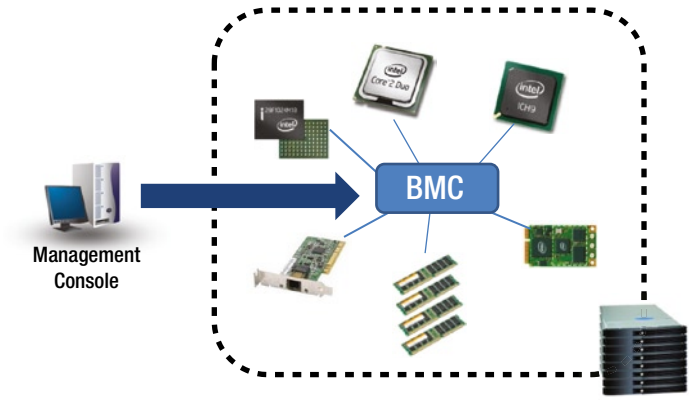

Figure 2-6. *Baseboard management controller*

The powerful capability and convenience of BMC comes with a price. Due to the cost, BMC is usually only justifiable for deploying on large servers that carry critical tasks.

In-Band Solutions

An important component of any management methodology is how the data of the managed machine is transmitted to the managing console for diagnosis and analyzed. The communication link determines the security and reliability of the communication.

An in-band solution leverages the communication and network stacks of the underlying operating system and is often utilized by software management solutions, such as VNC, SSH, and so on. The in-band communication suffers the same limitations of software management, that is, dependability, availability, reliability, and security.

Out-of-Band Solutions

In contrast to in-band, an out-of-band solution employs dedicated channels for communicating with the console. Generally speaking, out-of-band solutions are more robust and secure than in-band solutions, thanks to the isolation from the host being managed.

For example, a network KVM device implements a network interface separated from the network stack of the managed computer's operating system. The connections of KVM and the computer run side by side and are independent of each other.

The 2009 AMT book extends the definition of "out-of-band" for a special case, where the wired or wireless network adaptor is shared by both the operating system and an isolated management device. In this case, although the management device is located inside the chassis of the computer and it is not equipped with dedicated network hardware, it is still considered out-of-band because the management does not depend on the operating system. Figure 2-7 illustrates the sharing of a network card.

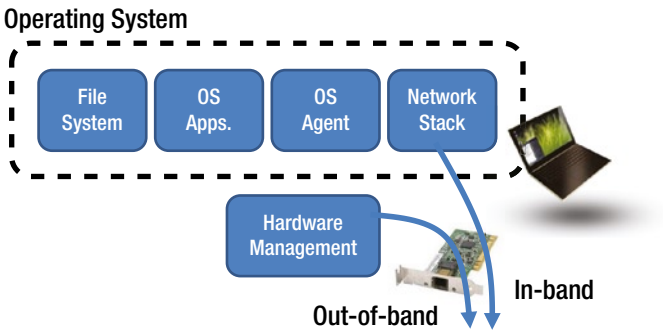

Figure 2-7. *Out-of-band management: both the operating system and the hardware management traffic can use the same network hardware*

Sharing a network device such as a NIC (network interface card) certainly reduces the bill of material (BOM) cost, but this slightly compromises functionality and security compared to using a dedicated network device. Functionality-wise, if the network card itself is malfunctioning and requires troubleshooting, then the communication channel between the computer and the managing console is essentially broken. Because no data

can be received from the problematic computer, the administrator may have to debug the issue on site. Security-wise, for the network sharing to function properly, it is required that both the network driver on the operating system and the management device agree and obey a predefined protocol. If the driver is compromised and does not follow the protocol, it may cause a racing condition on the hardware and mount, and at a minimum, denial of service attacks, so that the system data cannot be sent to the console.

To avoid the complications of network device sharing, most security applications running on the embedded engine, unlike the AMT, do not use the firmware's network stacks to communicate with remote entities. Instead, if an application is required to exchange data with a remote server (for example, an authentication server), then it will rely on software programs running on the host operating system as the proxy.

Intel AMT Overview

We have seen different management solutions and their pros and cons. Table 2-1 gives a summary.

Table 2-1. *Comparison of Management Solutions*

Solution	Functionality	Dependability	Reliability	Availability	Security	Cost
Software, in-band	Fair	Poor	Poor	Poor	Fair	Good
Hardware, out-of-band with separate network device	Good	Good	Good	Good	Good	Poor
Hardware, out-of-band with shared network device	Good– (cannot debug NIC)	Good	Good	Good	Good	Good

As shown in Table 2-1, there is no perfect solution. However, the hardware out-of-band solution with a shared network device is the best option overall. Intel AMT is such a solution with the following desirable characteristics:

- It resides in the chipset and it is always available on all Intel vPro platforms.
- It is independent of the host operating system and power state.
- It is functional even if the host is in a lower power state or has crashed.
- It shares the network device with the host so that the hardware overhead is minimal.

The AMT ships with three software components: BIOS extension, local management service and tray icon, and remote management. They serve three configuration scenarios, respectively: through HECI before the operating system is loaded, through HECI after the operating system is loaded, and through the network.

BIOS Extension

The BIOS extension for the engine is called the Intel management engine BIOS extension (MEBX). It is a BIOS component similar to other extension ROMs. It allows the administrator and the user to perform basic configurations for the management engine and the AMT, including changing the password for authentication, turning on and off the AMT, assigning Internet Protocol (IP) addresses, configuring network protocols, selecting the engine's sleep and wake policies, and so on.

The primary reason for introducing the BIOS extension is to protect end users' privacy. By the nature of BIOS, it requires a human being's physical presence and knowledge of the correct password to authenticate to the management engine and change configurations.

The BIOS extension communicates with the embedded engine through the HECI channel. A HECI driver is implemented in the BIOS extension to facilitate the communication. The BIOS extension does not implement encryption algorithms. There is no protection applied to the HECI interface, and the messages are sent in the clear. Data sent to the engine by the BIOS extension is stored by the engine securely in nonvolatile memory with appropriate protections.

The BIOS extension executes before the BIOS delivers the end-of-POST (power-on self-test) signal to the embedded engine. The engine relies on the end-of-POST signal to determine whether a received HECI command was initialized from the BIOS extension or from the local host operating system. Select settings are deemed legitimate only if made through the BIOS extension interface. The embedded engine rejects such commands by returning an error if they are received after the end of POST.

Figure 2-8 demonstrates the flow of interactions between the host and the management engine during and after the boot process. The initial boot block is a firmware module loaded before the BIOS to facilitate the secure boot path. After the BIOS has initialized the system DRAM and reserved the exclusive region for the engine to access, it sends a DRAM-init-done HECI message to notify the engine.

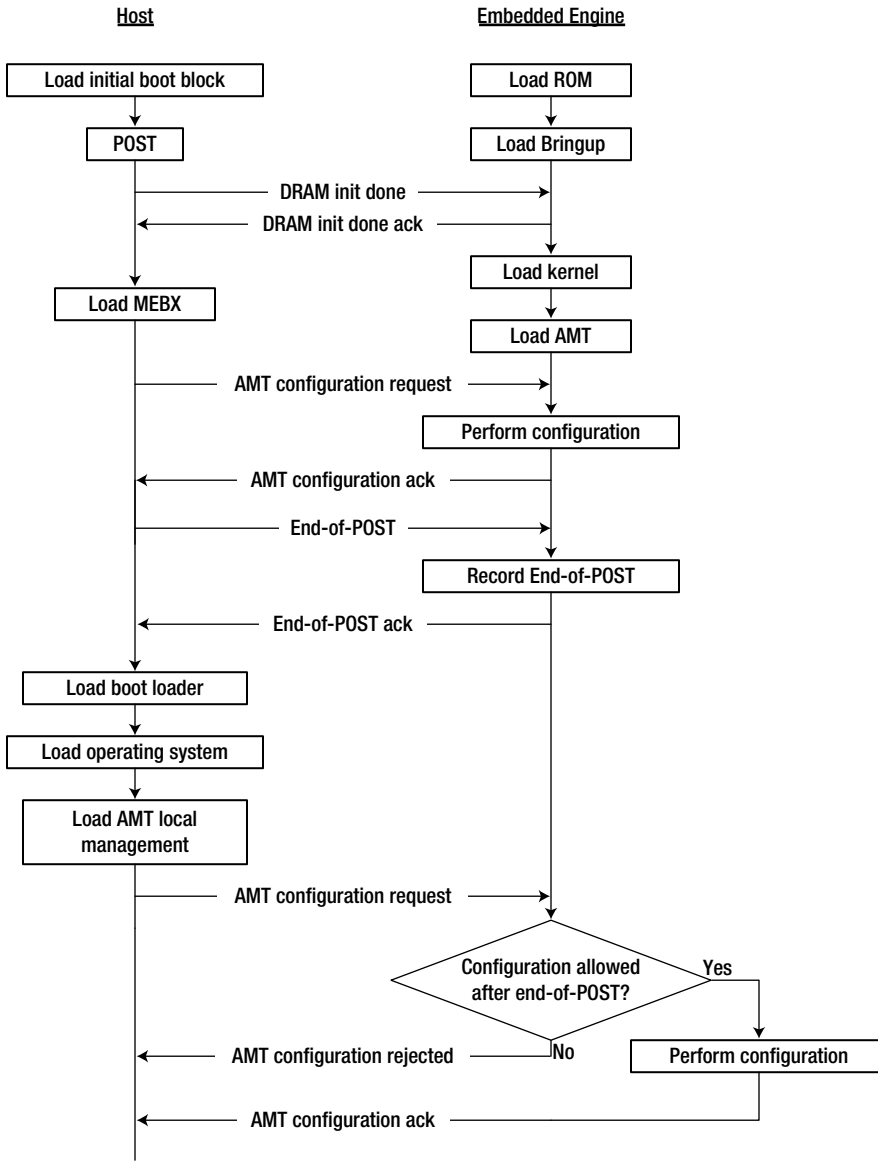

Figure 2-8. *Interaction between the host and the engine for AMT configuration, with MEBX loaded during the boot process*

The HECI commands initiated from the MEBX are delivered to and handled by the kernel or the AMT firmware module. Because end-of-POST has not happened yet, the firmware will always honor the requests, perform configurations, and return to the MEBX. Once the administrator has finished his configuration work, he exits the MEBX. Next, the BIOS sends the end-of-POST command to the management engine, signaling that the BIOS is now handing the control to the boot loader and the operating system. An AMT configuration command received by the engine after end-of-POST will be examined and processed only if it is permitted after end-of-POST, based on predefined policies.

Notice that the BIOS may not be an Intel production. Therefore, the BIOS, including all BIOS extensions, is excluded from the engine's trusted boundary. The engine does not depend on the integrity of the BIOS to achieve its security objectives. For example, during authentication, the password entered by an administrator or user is transmitted from the BIOS extension to the engine for examination, and not in the other direction. And even though an end-of-POST message never reaches the engine, the engine will not leak any secrets. By design, the most harmful attack a compromised BIOS component is able to launch against the engine should be to DoS the engine. For example, if the DRAM-init-done message never reaches the engine, then the engine will be operating in a degraded mode, because it does not have DRAM to run applications that require a large amount of memory.

Local Management Service and Tray Icon

The purpose of the AMT's local management service is to provide a similar programming interface for both local and remote applications.

As depicted in Figure 2-9, the local AMT application or the AMT user notification service opens a virtual network connection to the AMT firmware and it uses WSMAN (Web Services-Management). The application or the UNS does not have any knowledge about the firmware's HECI mechanism. The local management service consumes the HECI driver and redirects the network traffic to the HECI link to the embedded engine. The AMT application is developed by third-party software vendors, and the user notification service is provided by Intel.

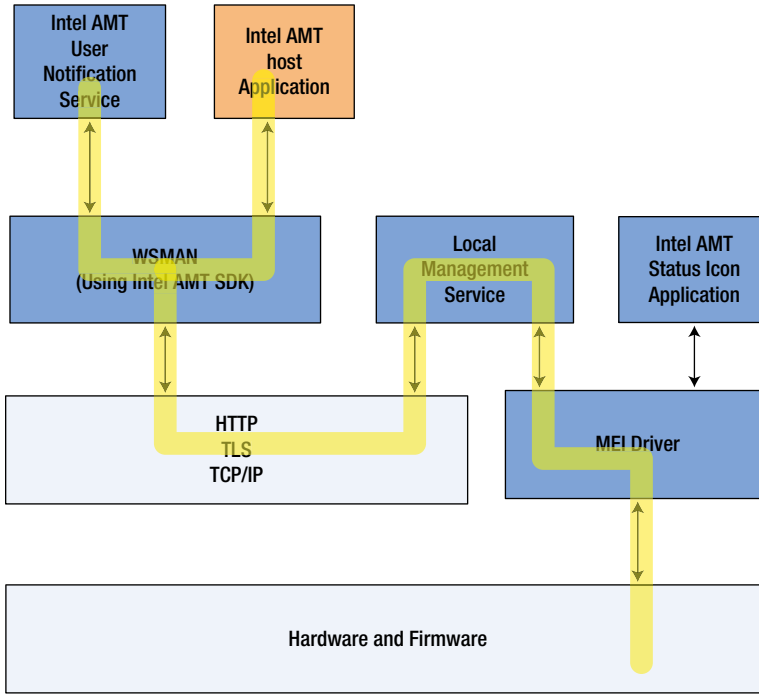

Figure 2-9. *Local software components of the AMT*

There is also a tray icon application that is developed by Intel. The tray icon application fetches status information of the management engine from the HECI interface.

Remote Management

Intel releases the AMT SDK (software development kit) to facilitate developers to interact with the AMT firmware and integrate the AMT features into their existing system management consoles and applications.

Earlier versions of AMT supported EOI (External Operations Interface) over SOAP (Simple Object Access Protocol), but the latest AMT releases only support the WS-Management interface.

Refer to the Intel AMT Implementation and Reference Guide[6] for details on the remote management development with AMT SDK.

The Engine's Evolvement: from Management to Security

Seven years since its first deployment, the management engine has become the *security* and management engine. The evolvement did not happen overnight. The shift of focus from system manageability to security reflects the increasing importance of security in today's computing industry and ecosystems.

The security and management engine has a number of desirable properties that make it not only a good manageability solution but also an excellent security solution.

Embedded System as Security Solution

What makes a solution a good one for running security applications?

Advanced techniques have been developed for creating trustworthy software solutions. These techniques include a managed runtime environment (MRTE), tamper-resistant software (TRS), a secure virtual machine (VM), Intel TXT, Intel Software Guard Extensions (Intel SGX), and so forth. Refer to the Intel Corporation white paper "Using Innovative Instructions to Create Trustworthy Software Solutions," for an introduction to the various secure software solutions.[7] However, these solutions suffer from different restrictions. And software, by its nature, is more vulnerable to attacks. It is hard for software to gain a comparable level of trust as equivalent hardware solutions. For example, several content protection schemes allow playback of certain high-definition contents only if the video path is protected by hardware.

Although it could provide very strong protection, a pure hardware solution is not preferable either. The problem of realizing security applications in hardware is the lack of flexibility and high cost. For convoluted features, it is very difficult to avoid bugs. Software programs can be patched with minimum overhead, but hardware issues may not be patchable and may require recall, which is a disaster for computer manufacturers.

A firmware/hardware hybrid is the solution that inherits the advantages of both software and hardware. On one hand, firmware runs on dedicated hardware and features hardware-level protection for security applications. On the other hand, the firmware can be stored in rewritable nonvolatile storage, and enjoys simpler deployment and the flexibility of being patched or updated at a relatively small cost.

The security and management engine is such a firmware/hardware hybrid product. Security-wise, a few highlights of the design are listed next. More details can be found in Chapter 4 of this book.

- *Independency*: The engine enjoys its own computing environment that is independent of the main operating system running on the host. The engine can run normally when the operating system crashes with a blue screen or cannot boot. Even if the host is sleeping or hibernating, the engine can also run normally. Notice that the reserved memory may not be available when the host is in a low-power state. Consequently, certain firmware features that require a large amount of memory may not function when the system is in a low-power state.

- *Isolation*: The engine does not share a processor or main memory with the host. The reserved memory is under strong confidentiality and integrity protection (see Chapter 4 of this book for details), so it is virtually isolated from the rest of the DRAM that is controlled by the host operating system. The networking devices, even if compromised, do not compromise the engine's security objectives. The DMA engine and HECI channel do not rely on the correct behavior of the host. In general, an external adversary (malware, virus, and so forth) is not able to infect the firmware.

- *Closed system*: The engine loads only firmware that is digitally signed by Intel for the engine. Attackers cannot easily change the firmware kernel or add/remove applications.

- *Small attack surface*: The only general interface that is available to all firmware modules to the host is the HECI channel. A small number of modules may invoke DMA and other low-level I/O, such as GPIO (general-purpose input/output), as needed. And only the AMT application may access the network. Data intake from these interfaces is not trusted by the security and management engine, and is fully validated before being processed. Invalid input data may cause wrong calculated responses from the engine, but will not crash the engine or compromise the security of the engine.

- *Programmability*: In addition to its native firmware applications, the engine opens its security capability to third-party host applications by exposing security APIs through HECI. See Chapter 9 of this book for more information.

- *Power efficiency*: Because the engine runs at a low frequency (from approximately 200MHz to 400MHz, depending on the product) compared to the main CPU, the power consumption is in the scale of milliwatts. In addition, the engine supports power gating. After being idle for a configurable number of seconds, it enters the sleep state to conserve power. Events that can wake up the engine include a HECI message from the host or interrupts from I/O devices.

Flexibility-wise, only a small portion (more specifically, the boot loader and standard library functions) of the engine's firmware is stored in ROM for the sake of root security and performance, and all application firmware is stored in flash. This enables a firmware update to fix or patch hardware or firmware bugs in the field.

We have seen the advantages. But is the engine perfect? What about the "cons"?

- *Cost*: The engine is a separate core and it shares few hardware devices with the main operating system. Although more isolated and secure, this adds the BOM cost of the platform, compared to security solutions that do not introduce a dedicated processor.

- *Limited computing bandwidth*: To save power and cost, the engine's processor runs at a relatively low frequency. This restricts it from serving applications that require high throughput. However, note that most security applications do not require overwhelming performance and the bandwidth is not a major concern.
- *Difficult firmware update deployment*: It is relatively easy for software to push patches and updates to end users' devices. This helps software vendors fix vulnerabilities and add new features in a timely manner. The story of a firmware update is completely different, however. Because the firmware is part of the security and management engine, and a component of the chipset or SoC, firmware hotfixes and maintenance releases must be thoroughly tested for compatibility by OEMs before being pushed to devices that are in the field. This process usually takes anywhere from a few weeks to a few months, and may not happen at all. To address the problem of firmware updates, a stringent security review process is exercised in the attempt of minimizing the need for hotfixes.

Overall, the pros of using the engine as the security solution outweigh the cons, making the engine *the* ideal place for security solutions.

Security Applications at a Glance

Realizing these attractive properties of the infrastructure, no one would be satisfied if the management engine remained just a system management tool. System manageability is an important and useful application, but it does not make use of the full potentials offered by the engine. Now that the engine is available on the system, why not make the most out of it?

First, the engine should be used as frequently as possible—not only when management service is requested on the system. After all, how often do system problems happen? They do not happen every day.

Second, a successful state-of-the-art technology should not benefit only the network administrators and the employees in enterprises. It should bring values to a larger population.

There are clearly many more possibilities and opportunities to be explored on the security and management engine. In today's mobile age, the demand for secure mobile services that involve valuable assets is gaining significant momentum. As a result, the embedded engine is reborn with new security features that are serving all end users every day.

EPID

Thanks to its direct access to hardware and isolation from the host operating system, it is convenient to leverage the security and management engine as the root of trust for the platform. The EPID (enhanced privacy identification) is a security mechanism exclusively built in the engine and serves as the hardware security root of trust for various applications running on the platform.

During Intel's manufacturing process, a unique EPID private key is retrieved from an internal key generation server and programmed to the engine's security fuses. At runtime, the engine's firmware uses the EPID private key to prove to the local host or a remote server that it is a genuine Intel platform and eligible for premium services that are available only to Intel products. Those applications rely on a hardware infrastructure that is only supported by Intel's products. For example, the CPU upgrade service, PAVP (protected audio and video path), and so on.

Leakage of an EPID private key would allow hackers to write software masquerading as Intel hardware. Such attacks may break into the applications that were built on the EPID and then steal secrets, such as user's stock brokerage passwords or copyrighted contents. To prevent the EPID key from being compromised, comprehensive protection mechanisms for the EPID private key at rest and at runtime are implemented by the engine. Of course, the EPID key generation process is also safeguarded with very strong and restrictive policies. In fact, except for the purpose of debugging, no human being is supposed to know any EPID private key value. Having said so, a key revocation scheme is supported by the engine in case of incidents.

To summarize the requirements, the EPID credential must be unique per platform; it must always be available; and the deletion, alteration, theft, or cloning of the EPID credential on one platform to another platform shall not be feasible without employing special hardware equipment and significant resources. Such a level of security strength is very difficult, if not impossible, to achieve by software solutions. The security and management engine is the ideal place to implement EPID functionalities. It offers not only ample security protection, but also flexibility in supporting EPID applications because the engine is a hardware/firmware hybrid device.

Chapter 5 of this book has more information on EPID.

PAVP

Some applications need to securely display frames and play audio to the user. The security requirement is that software running on the host operating system must not be able to peek or steal the contents being securely played back.

For example, alongside the wide deployment of the media playback feature on mobile computing devices is the problem of protecting copyrighted contents from piracy. Some content creators (such as movie studios) consider software protection insufficient and require their high-definition content, when playing back on computers, to be protected by hardware mechanisms. In other words, if a user's computer is not equipped with the required hardware capability, then that user won't be able to enjoy those contents.

Another example for the secure display usage is Intel IPT, where a sprite of keypad is displayed on the screen for the user to enter a password by mouse clicks. The sprite must be hidden from the host to prevent attacks by screen scrapers.

Intel's PAVP technology is a hardware scheme that protects video and audio assets from software attacks. Initially introduced for Blu-ray, PAVP is now used by a range of applications that rely on content protection to function.

The PAVP is realized by a few components: player software and graphics drivers on the host, the security and management engine, and the GPU. The ultimate security goal of content protection is to make sure that the content encryption key and the clear content are only visible to hardware and not exposed to any software components, including ring 0 drivers.

The responsibilities of the engine in the PAVP solution include:

- Establishing a PAVP session between the software and the GPU.
- Delivering content encryption keys to the GPU.
- Implementing the HDCP[8] (high-bandwidth digital content protection) protocol.

Chapter 8 has more information on PAVP.

IPT

Identity theft is one of the most infamous and costly cybercrimes. Anyone that uses the Internet to manage assets (such as music, photos, social life, financial accounts, and so on) can potentially be a victim. Strong authentication and transmission protection is necessary to deter identity theft. Intel IPT, backed by the security and management engine together with other components, is a cutting-edge technology for protecting end users' identities.

The IPT is an umbrella product name that comprises a numbers of features, including, as of this writing, OPT (one-time password), PTD (protected transaction display), PKI (public-key infrastructure), and NFC (near-field communication). Additional functionalities may be introduced to the IPT family in the future. These features work collaboratively to offer comprehensive identity safeguarding for the users for multiple scenarios.

- *OPT*: Implements as the second factor in a multi-factor authentication solution. The user's computer is the second factor (something you have), and the OPT is generated by the security and management engine's firmware and transmitted to the remote server for authentication. The technology eliminates the need for a physical token, meanwhile maintaining the security level.
- *PTD*: Allows a trusted entity to draw and display a secure sprite on the screen directly with the help of PAVP. The sprite is completely invisible to the host software stack. The secure display is commonly utilized for delivering sensitive information that is for the user's eyes only—for example, a keypad for authentication.

- *PKI*: Provides a robust private key management mechanism, including key generation, key storage, signature generation, and decryption. Once a private key is generated by or imported to the security and management engine, it will never be output in the clear. The engine performs private key operation under the hardware protection.
- *NFC*: Allows a user to tag his NFC-capable credit card against the NFC sensor on his computer to conveniently complete online transactions with positive identity authentication.

More technical details about the security and management engine's role and responsibility for IPT can be found in Chapter 10.

Boot Guard

Intel Boot Guard is the technology for protecting boot integrity for Intel platforms. The system's boot block is measured by hardware and the boot is allowed if and only if the measurement is successful, that is, the boot block is not altered. The hardware elements that perform the boot integrity check are the security and management engine and the CPU.

Intel Boot Guard offers two configurations: *verified boot* and *measured boot*. The engine is equipped with an array of field programmable fuses. For *verified boot*, an OEM programs the fuses with the hash value of its public key before the conclusion of the manufacturing process. The corresponding private key is used by the OEM to sign its initial boot block module, the first OEM's component that executes during boot. During the boot process, the engine and the CPU first verify the public key in the OEM's initial boot block manifest by comparing its hash with the preconfigured hash in the field programmable fuses, and then verify the OEM's signature on the initial boot block using the public key.

Alternative to using a digital signature, the *measured boot* configuration leverages the TPM on the platform. The TPM can be either a discrete TPM or a firmware-based TPM that is built in the security and management engine.

Chapter 6 of this book has more technical details on Intel Boot Guard technology.

Virtual Security Core: ARM TrustZone

ARM is an industry leader in low-cost and low-power processors, with applications in a host of mobile embedded devices, especially in the smartphones and tablet markets.

ARM deploys several security measurements among various families of products. For instance, the SecurCore family[9] provides mitigations against software, hardware, and side-channel attacks, for small form factors, such as smart cards. In particular, the SecurCore solutions enable customization of security features for a specific design and provide development process tools with added security controls.

For SoC platforms, ARM's security solution is called the TrustZone technology[10] (a.k.a. security extension). TrustZone is supported by ARM1176 and the Cortex-A series processors. In contrast to Intel's security and management engine that uses a dedicated security core, the TrustZone takes a different approach. The TrustZone splits a physical processor core and treats it as two virtual cores (or modes): one nonsecure mode and one secure mode. The nonsecure mode is also called normal mode or untrusted mode; the secure mode is also called trusted mode. The two modes share the same hardware resources but they operate independently. Some literatures refer to "mode" as "world."

Secure Mode and Nonsecure Mode

Context switch between the nonsecure mode and the secure mode is conducted through a third mode, the monitor mode, which is managed inside the secure mode. The current mode of operation is indicated by the nonsecure (NS) bit, which is bit 0 of the secure configuration register (SCR). The SCR is a read/write register that is accessible in the secure mode only, and recommended by ARM to be programmed by the monitor mode. Besides the NS bit, the SCR is also used to configure whether an interrupt—FIQ (fast interrupt request) or IRQ (interrupt request)—should be branched to the monitor mode for processing. The entry to the monitor mode can be triggered by software executing a dedicated instruction, the Secure Monitor Call (SMC) instruction, or by a subset of the hardware exception mechanisms.[11] Figure 2-10 shows the relationships among the secure mode, nonsecure mode, and the monitor mode.

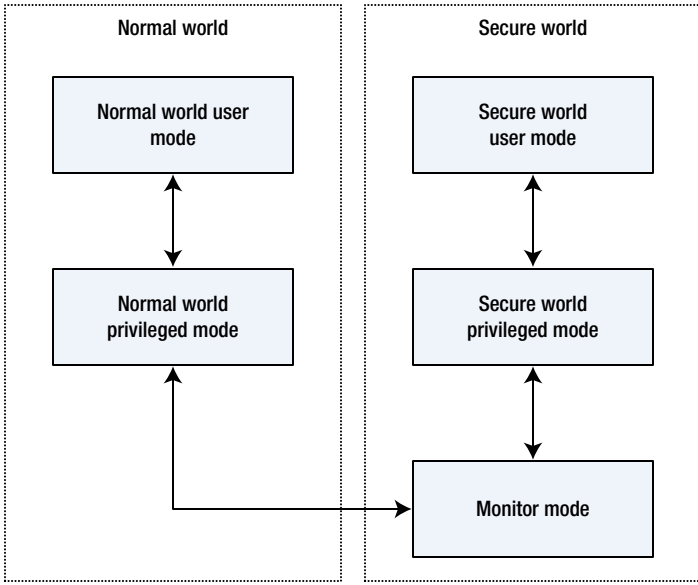

Figure 2-10. *Modes in an ARM core implementing the security extensions*

The switches between the two modes are strictly controlled by hardware. The secure mode is essentially another level of execution privilege. The secure mode must not leak secrets to the normal world or allow any form of privileged escalations. Applications run mostly in the normal mode, but small security-specialized code that handles secret data and sensitive operations is executed in the secure mode. For example, the key processing for content protection is run in the secure mode.

In addition to the processor, the separation of the two modes permeates all hardware, most interestingly, memory and device buses.

Memory Isolation

The memory infrastructure inside and outside of the processor core must also be isolated into two modes accordingly.

The level 1 (L1) cache in the processors is managed by the so-called memory management unit (MMU), which converts the virtual address space that is seen by the software running on the processor onto the physical address space. The MMU features an L1 memory translation table with an NS field, and entries in the TLB (translation look-aside buffer) are tagged with the NS bit. The secure mode relies on the value of the NS field to determine the value of the NS bit of the SCR when it is accessing the physical memory locations. The nonsecure mode ignores the NS field. In other words, the secure mode is always allowed to access memory belonging to both the secure mode and the nonsecure mode. Select processor models feature Tightly Coupled Memories (TCMs), which are high-performance SRAM that exist at the same level of L1 cache. There can be up to two blocks of TCM present on each instruction and data interface. Software can configure the TCMs to be accessible to the secure mode or nonsecure mode.

The Memory Protection Unit (MPU) was introduced to ARM cores starting from ARM7. This unit allows partitioning of memory into different sections and assigning them different security attributes, for example, marking the code section as read-only in order to prevent runtime alteration attack at runtime. The read/write permissions are based on two-level User and Privilege mode access; if a User mode application tries to access the Privilege mode memory, then the processor triggers an exception. The initial boot routine and interrupt handling vectors executes in the Privilege mode.

Bus Isolation

The isolation of bus interfaces and devices is required to prevent attacks from system devices. The AMBA3 (the third generation of the Advanced Microcontroller Bus Architecture) AXI (Advanced Extensible Interface) bus protocol defines controls to identify operating modes for all transactions. The AXI bus adds metadata to bus control signals and labels all read and write transactions as secure or nonsecure. The hardware logic in the TrustZone-enabled AMBA3 AXI bus fabric ensures that secure-mode resources cannot be accessed by nonsecure mode components.

The AMBA3 APB (Advanced Peripheral Bus) is used for secure peripherals and interrupts. The APB is attached to the system bus using an AXI-to-APB bridge. The APB per se is not equipped with an NS bit or its equivalent. Therefore, the AXI-to-APB bridge hardware ensures that the security of APB peripheral transactions is consistent with the AXI security signals.

Physical Isolation vs. Virtual Isolation

Conceptually, TrustZone has its similarities to Intel TXT in the sense that both achieve isolation between the secure and nonsecure modes through a trusted virtual machine or execution environment. In reality, on many Intel platforms, the security and management engine is the counterpart for security solutions that are realized by TrustZone on ARM platforms.

The obvious advantage of TrustZone over a dedicated security core is its lower BOM cost—only one core is needed for two modes of operation. But are there tradeoffs?

Although ARM's TrustZone and Intel's security and management engine both feature hardware-based security operating environments, their architectures are completely different. The isolation between the nonsecure mode and the secure mode is *virtual* for TrustZone, versus *physical* for the security and management engine. For the virtual separation mechanism, safeguarding the border of the virtually secure world and defending against threats could be a challenging task.

In addition to security, power efficiency is another important consideration for modern mobile platforms that aggressively power save. For TrustZone, the secure mode and the nonsecure mode run at the same frequency. In contrast, the security and management engine runs at a lower frequency than the main processor, resulting in less power consumption at the tradeoff of a slower operation of security tasks, which in most cases do not require high performance.

Furthermore, as described earlier in this chapter, Intel's embedded solution is also a management engine. Its many unique properties make it an excellent choice for platform management applications.

References

1. Arvind, Kumar, Purushottam Goel, and Ylian Saint-Hilaire, *Active Platform Management Demystified—Unleashing the Power of Intel vPro Technology*, Intel Press, 2009.

2. Intel Trusted Execution Technology, www.intel.com/txt, accessed on January 30, 2014.

3. Intel Virtualization Technology, www.intel.com/content/www/us/en/virtualization/virtualization-technology/hardware-assist-virtualization-technology.html, accessed on January 30, 2014.

4. Intel Identity Protection Technology, www.intel.com/content/www/us/en/architecture-and-technology/identity-protection/identity-protection-technology-general.html, accessed on January 30, 2014.

5. Intel Anti-Theft Technology, www.intel.com/antitheft, accessed on January 30, 2014.

6. Intel AMT Implementation and Reference Guide, http://software.intel.com/sites/manageability/AMT_Implementation_and_Reference_Guide/default.htm, accessed on January 30, 2014.

7. Hoekstra, Matthew, Reshma Lal, Pradeep Pappachan, Carlos Rozas, Vinay Phegade, and Juan del Cuvillo, "Using Innovative Instructions to Create Trustworthy Software Solutions," http://software.intel.com/sites/default/files/article/413938/hasp-2013-innovative-instructions-for-trusted-solutions.pdf, accessed on January 30, 2014.

8. Digital Content Protection LLC, "High-bandwidth Digital Content Protection," www.digital-cp.com, accessed on May 10, 2014.

9. ARM SecurCore Processors, http://www.arm.com/products/processors/securcore/, accessed on April 1, 2014.

10. ARM TrustZone Technology, www.arm.com/products/processors/technologies/trustzone/index.php, accessed on January 30, 2014.

11. ARM Limited, "ARM Security Technology—Building a Secure System using TrustZone Technology," http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf, accessed on April 1, 2014.