Cell Broadband Engine processor vault security architecture

by K. Shimizu, H. P. Hofstee,
and J. S. Liberty

Abstract

Current data protection technologies such as those based on public-key encryption and broadcast encryption focus on the secure control and protection of data. Although these protection schemes are effective and mathematically sound, they are susceptible to systematic attacks that utilize any underlying platform weakness, bypassing the cryptographic strengths of the actual schemes. Thus, ensuring that the computing platform supports the cryptographic data protection layers is a critical issue. The Cell Broadband Engine™ (Cell/B.E.) processor security architecture has three core features that are well suited for this purpose. It provides hardware-enforced process isolation in which code and data can execute in physically isolated memory space. It also provides the ability to perform hardware-supported authentication of any software stack (i.e., “secure boot”) during runtime. Finally, the architecture provides a hardware key to act as the root of an encryption chain. Data encrypted directly or indirectly by this key can be decrypted and provided only to an application that is running in the isolated memory and that has been verified. This significantly reduces an adversary's chances of manipulating software to expose the key that is fundamental to a data protection or authentication scheme. Furthermore, it provides a foundation for an application to attest itself to a remote party by demonstrating access to a secret.

Introduction

With the increasing connectivity and the virtualization of computing resources, we are seeing a new paradigm of computing in which the location of a process is difficult to determine and may even become irrelevant. For example, consumer computing resources are being used to solve massively parallel problems by using peer-to-peer computing technology, as in the case of the World Community Grid [1]. Also, large Internet commerce sites, which have excess capacity during off-holiday times, sell their computing capacity to other companies [2]. Thus, we are gradually seeing the beginning of an era in which one's computing process can be computing anywhere in the world and in which one's process can be sharing a single computing resource with “stranger” processes. Thus, the owner of the process must trust that the computing resource will not manipulate or steal one's process or data and that the computing resource is secure enough to keep stranger processes isolated from one another. Only when the required security technology is in place will this virtual computation world flourish.

With existing computing resources, securing the underlying platform, including the operating system, firmware, and device drivers, is very difficult. The microprocessors, which are the brains of these platforms, are typically not designed with security as a priority, or they simply do not have features that security architects consider fundamental to a secure platform. Thus, security architects design around this limitation with software-based approaches, such as using software to compartmentalize and separate applications or to check for application-code tampering.

However, protecting software with software has a fundamental flaw in that the protective software may be compromised as well. Therefore, a more ideal solution is to rethink and rearchitect the processor hardware, which is intrinsically less vulnerable than software, to support the security of the platform.

The Cell Broadband Engine† (Cell/B.E.) processor is designed with this hardware-centric secure platform goal in mind. Because its designers were given the rare opportunity to design a processor from the ground up, the Cell/B.E. processor is arguably one of the first high-performance, general-purpose microprocessors [3] in which security is an integral part of the processor architecture and not an afterthought. The main strength of its architecture is its ability to allow an application to protect itself from other software running on the platform by using features in the hardware design. The application need not assume that other software on the platform, especially the operating system, is well designed or has not been compromised by an adversary. The application can trust the hardware to protect it directly so that sensitive, high-value data can remain protected even if the platform software is compromised or modified.

This ability to guarantee security independently of the operating system or other software on the system may be key to the success of the virtual computing resource framework described earlier. With this technology, a particular process owner does not have to worry about whether the administrator of the computing resource has installed the newest operating system security fixes or whether the process will be running side by side with a malicious application. The processor design directly guarantees security regardless of the software environment of the platform.

Attack models

Existing processor architectures follow the ring protection structure [4], in which ring 0 at the center has the highest privilege and the outer rings have less privilege. Software that runs in ring 0 can access any memory area or input/output (I/O) device, while software in the outer rings is limited in the areas that it can access. Most user applications reside in the outer rings. The problem occurs when a malicious user application succeeds in moving from an external ring to ring 0; once this happens, any critical data such as keys and secrets that were protected by the ring structure become vulnerable. The Cell/B.E. processor addresses this issue by providing a protection structure that is orthogonal to the ring structure. Also, some attacks against the Advanced Encryption Standard (AES) have been successful by manipulating the processor cache [5]. The solution to this problem is twofold: The Cell/B.E. processor provides a memory hierarchy that does not use a cache architecture, and it provides an atomic hardware operation that erases the memory region used by an application.

Related work

The Trusted Computing Group (TCG) [6] is focused on using an external hardware chip, the Trusted Platform Module (TPM), as the root of its security. In contrast, because the Cell/B.E. processor vault security architecture is integrated into the processor itself, it can offer protection from attacks that the TCG approach is vulnerable to, such as buffer-overflow attacks [7]. As for other commercial processor security architectures, much of the isolation is based on hardware support for virtualization. However, support for virtualization has been part of the IBM PowerPC Architecture* [8] for several generations and is fully incorporated in the Cell/B.E. Architecture (CBEA), although it is not considered to be a primary security offering. Within academia, there have been other approaches to provide hardware-based security that is tolerant of operating

system compromises [9].

Cell/B.E. processor overview

The Cell/B.E. processor is a multiprocessor core architecture [10] (Figure 1). The cores are heterogeneous, and there are two classes of cores on the chip. The 64-bit PowerPC* processor element (PPE) is the principal core that assumes a supervisory role. This PPE runs the operating system. The other type of core on a Cell/B.E. processor is the synergistic processor element (SPE); in the current implementation, there are eight SPEs. The SPEs are the computational workhorses: each contains a reduced instruction set computing (RISC)–style single-instruction multiple-data (SIMD) set, a wide and large (128 128-bit) register file, and 256 KB of physically dedicated private memory, or local storage (LS) [11]. The high-bandwidth element interconnect bus (EIB) connects these processor cores to one another and to the off-chip system memory and I/O.

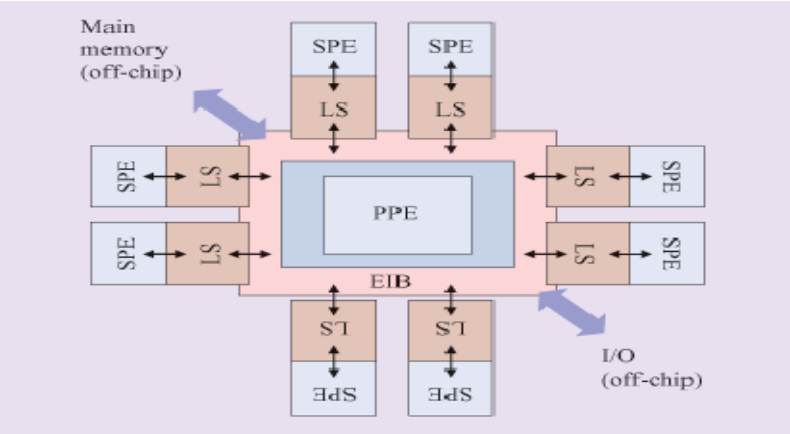

Figure 1

Cell/B.E. Architecture: Nine heterogeneous processor cores, including one PowerPC–based PPE (runs operating system) and eight SPEs (data processing workhorses). Cores are connected to one another, system memory, and I/O through the EIB.

Figure 1

The SPE plays a key role in the Cell/B.E. processor security architecture. It fetches instructions from the LS and loads data from and stores data to the LS. However, the LS is not a hardware-managed cache. Instead, the LS memory region is mapped in the system memory map, and software (either the software running on the PPE or the software thread executing on the SPE) is expected to explicitly transfer code and data into the LS. The transfers can occur with any resource on the EIB, such as main memory, LS of other SPEs, and I/O devices.

For example, consider a programming model in which the PPE software launches an application thread on an SPE. After confirming that the SPE is stopped, the PPE initiates a data transfer of the code and data for the application from system memory into the LS of the SPE. Once the transfer is complete, the PPE sets the SPE program pointer to the entry point of the code and sends a command to the SPE to start executing. During the execution of the application on the SPE, the application code must explicitly initiate a data transfer with the source address (e.g., in system memory) and the destination address (an LS address) if it must transfer data between the LS of the SPE and system memory (in order to obtain more data blocks or code blocks).

To summarize, on one side, the SPE processor reads from and writes to the LS, and on the other side, the LS receives reads and writes from agents on the EIB. These memory transfers via the EIB are explicitly controlled by software and are not initiated by hardware.

Cell/B.E. processor security

Three architectural features define and differentiate the Cell/B.E. processor security architecture: the secure processing vault, the runtime secure boot, and the hardware root of secrecy features.

Secure processing vault

To achieve a secure platform, a processing environment must exist in which a single application can execute in isolation from all other executing software threads in the system. The Cell/B.E. processor secure processing vault provides such an environment. Within the vault, the execution of the application and its data cannot be manipulated or observed; the hardware design prevents other applications from doing so.

The goal of isolating a process thread is not new; however, in contrast to our hardware-based method, existing approaches have used software to enforce the separation. The operating system or the hypervisor (which is also known as the virtual machine monitor and is the software layer with the most authority in a virtualized system) is responsible for separating processes. For example, the operating system ensures that the memory location of the high-value data is protected from reads and writes from nonauthorized processes. The problem with this approach is that if an adversary takes control of the operating system or the hypervisor, security is instantly compromised. Furthermore, because of the sheer size and complexity of the operating system and, to a lesser degree, the hypervisor, they can be brittle and difficult to make secure.

The adversary can use the operating system to change the permissions for the memory area it is trying to access or simply use the operating system to read the memory location, since the operating system can read any memory location in most systems. An adversary looks for a weakness in the operating system design, such as a buffer overflow vulnerability, exploits this hole to gain control of it, and then executes actions that are restricted to the operating system it is attacking. Within this kind of environment, sensitive data can easily be copied by the adversary-controlled operating system. The fundamental problem with existing approaches is that the software they use to provide the isolation can be manipulated by an adversary. A better approach is for the hardware design to isolate the process in such a way that the software cannot override the isolation; this is the function of the Cell/B.E. processor secure processing vault.

In the Cell/B.E. processor, the vault is realized as an SPE running in a mode in which it has effectively disengaged itself from the bus and, by extension, the rest of the system. In this mode, the LS of the SPE, which contains the application code and data, is locked for use by the SPE alone; it cannot be read from or written to by any other software. Control mechanisms that are usually available for supervisory processes for the SPE are disabled. In fact, once the SPE is isolated, the only external action possible is to cancel its task, whereupon all information in the LS and SPE is erased by the hardware before external access is re-enabled. From the hardware perspective, when an SPE is in this isolation mode, the SPE processor access to the LS remains the same, while on the other side of the LS (the bus side), external accesses are blocked, as illustrated in Figure 2. Thus, all LS read and write requests originating from units on the bus such as the PPE, other SPEs, and the I/O have no effect on the locked region of the LS. However, for communication purposes, an area of the LS of the isolated SPE is left open to data transfers to and from other units on the

bus. The application running on the isolated SPE is responsible for ensuring that the data coming through the open communication area of its LS is safe. Also consistent with the idea that the cores execute independently, any number of SPEs can be in isolation mode simultaneously.

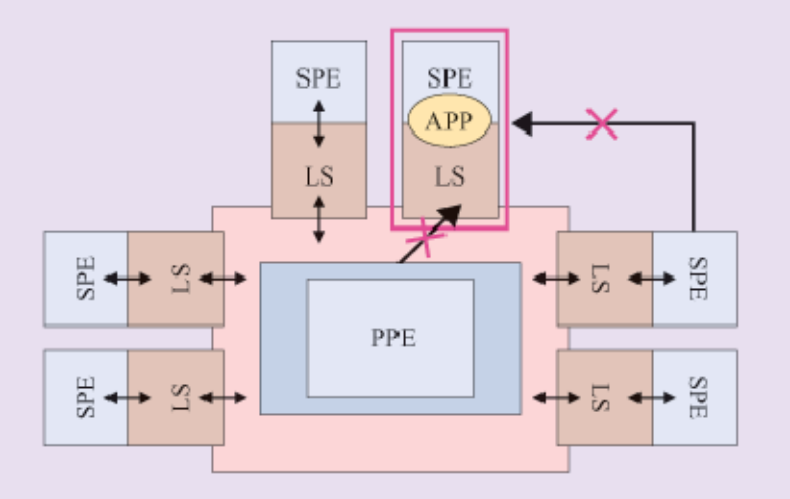

Figure 2

Example of a Cell/B.E. processor application. The PPE allocates an SPE for an application (APP), which can lock the SPE from the inside, thereby preventing the PPE and other applications from accessing the SPE. The PPE, however, retains the ability to cancel the SPE task to gain access.

Figure 2

All of this is accomplished exclusively by hardware means; no software (e.g., in the form of setting protection bits in an address translation table) is involved in the process. Because of this absolute hardware isolation, even the operating system and the hypervisor cannot access the locked LS or take control of the SPE core. Therefore, a hacker who has gained root or hypervisor privileges is not a threat to an application executing on an isolated SPE. The supervisory privileges do not enable the hacker to control the application, and they do not allow the hacker to read from or write to the memory used by it. The execution flow and the data of the isolated application are safe.

Runtime secure boot

The vault protects an application from other software that may have been modified or compromised. However, this does not address the question of what happens if the application itself has been modified. For example, an adversary can modify the application so that when it accesses valuable data within the secure vault, it copies the data into an openly accessible area outside of the vault. Such a modification must be detected so that such an application is not executed. One counter measure may be to design a software-implemented loader that checks the authentication of the application and executes it only when the authentication succeeds. However, the loader may be modified so that it does not check for authentication correctly and allows compromised code to execute within the vault. In another case, the loader may be circumvented entirely by an adversary, and the authentication step skipped. A hardware solution is needed to ensure that the authentication step is consistently and correctly executed.

Therefore, it is generally believed that the root of an authentication scheme must be implemented in hardware. If the root can be trusted, the entity authenticated by the root can be trusted, and so on as the chain of trust expands. The runtime secure boot feature is a technique based on this philosophy, whereby during power-on time, from the first basic I/O system (BIOS) code that is executed to the operating system code, the code modules go through a cryptography-based authentication check. There are a variety of ways to accomplish this; one is to have the hardware authenticate a small boot module by using a hardware key. Successful authentication of this module allows it to authenticate the operating system. If the authentication of the boot module or the operating system fails, the booting process is halted, but otherwise, the booting process is allowed to proceed normally. The concept is that since the first software to execute on the chip was authenticated by the hardware and all succeeding software code has been verified by the code that launched it, the chain of authentication guarantees that all software on the system has been indirectly or directly verified by the hardware key at power-on time.

The drawback of this approach is that it assumes that checking for compromises in the software at power-on time is sufficient. It does not protect against software compromises that occur after power-on time. However, most software-based attacks occur during runtime; in this event the chain of authentication breaks and any software launched subsequently cannot necessarily be trusted.

The Cell/B.E. processor addresses this problem with its runtime secure boot feature, which permits an application to perform a secure boot from the hardware an arbitrary number of times during runtime. Thus, even if other software on the system has been compromised, a single application thread can still be robustly checked independently. In essence, the application can renew its trustworthiness as many times as needed, even as the system stays running longer and becomes “stale.” Specifically, a hardware-implemented authentication mechanism uses a hardware key and a cryptographic algorithm to verify that the application has not been modified.

This runtime secure boot is tightly coupled with an SPE entering isolation mode. An application must go through the hardware authentication step before it can execute on an isolated SPE. When isolation mode is requested, the previous thread is first stopped and canceled, and all processor states are cleared. The hardware then automatically fetches the selected application into the LS, and the hardware verifies the integrity of the application. If the integrity check fails, the application is not executed. The check can fail for one of two reasons. First, if the application has been modified within memory or storage, the assumption is that its functionality may have changed and it cannot be trusted anymore. Second, the writer of the application may not know the cryptographic secret that is needed for a successful authentication. Otherwise, if the authentication check is successful, the hardware automatically begins execution of the application in isolation mode. Because all of these steps are controlled by hardware, verification of the integrity of the application cannot be skipped or manipulated and occurs consistently and correctly.

Figure 3 shows the state machine implementation of the secure processing vault and the runtime secure boot features. Normally, the synergistic processor unit (SPU) executes in the states in the nonisolated execution environment. A Load command initiates the transition into the vault environment. If the runtime secure boot succeeds, the state machine transitions to the SPU isolated running state, and the application is now executing in the vault. If the runtime secure

boot fails, the state transitions to the load failed state, in which case the operating system is expected to invoke the Exit command on an SPU in this state so that the SPU is recovered from the error state and is taken out of isolation mode. When the application has successfully completed execution in the isolated environment, it can also execute an Exit command to leave the vault environment. The Exit command initiates the atomic hardware operation of erasing all of the LS and the SPE register file before unlocking the isolated SPE. This is a critical step that prevents side-channel attacks such as the attack described in [5]. More details of the design can be found in [12].

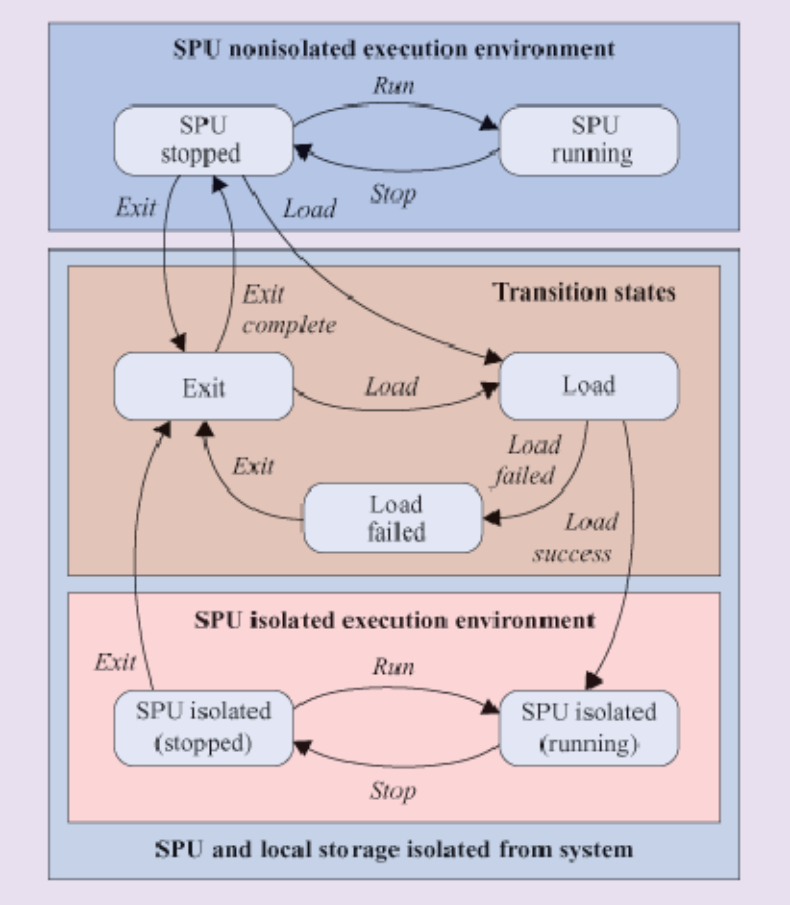

Figure 3

SPU isolated state transitions.

Figure 3

Hardware root of secrecy

One of the most important aspects of system security is the way in which keys are managed. Keys are the linchpin for system security and data protection. They are used by applications to encrypt data in storage, to decrypt encrypted files, or to establish a secure communication channel. If the keys can easily be exposed, the entire security scheme is compromised. Despite their critical role, keys are usually stored in plaintext form in storage. Ideally, instead of being left in this unprotected state, the keys are “sealed in an envelope” (or encrypted) when in storage and unsealed only when given to an authenticated application. However, this implies that another key

is used for the sealing and unsealing (in other words, for encrypting and decrypting the first key); how is this key stored? Eventually, there must be a key that is not encrypted; because this key is at the root of all unsealing, we refer to it as the root key.

Because the root key is important to keeping all other keys hidden, it must be robustly protected. The Cell/B.E. processor accomplishes this with its hardware root of secrecy feature. The root key is embedded in the hardware, and no software can access the root key; only a hardware decryption facility has access to it. This makes it much more difficult for software to be manipulated so that the root key is exposed, and the hardware functionality cannot be changed so that the key is exposed.

Furthermore, activation of the hardware decryption using this root key is tightly integrated with the SPE isolation mode, as shown in Figure 4. When an SPE enters isolation mode, the hardware decryption facility transfers the encrypted data into the isolated SPE and decrypts the data using the hardware root key. The decrypted data is placed within the protected LS and is available for an isolated SPE application to use. In fact, the decryption based on the root key can happen only within an isolated SPE and not outside it; there is no access to the root key or secrets decrypted by the root key, by hardware or software, from a nonisolated SPE or the PPE. First, this implies that a system designer can force all data decryptions by the root key to happen within the protected environment of the secure processing vault; the keys unsealed by the root key are always placed (at least initially) in the vault only. Second, only applications that have successfully passed the runtime secure boot authentication are given access to the keys unsealed by the root key. Any software that may have been adversely modified is not given access to the unsealed keys. Because the foundation of this control is grounded in both the runtime secure boot and the hardware root of secrecy features, the process is more resistant to manipulation than with a completely software-controlled access mechanism.

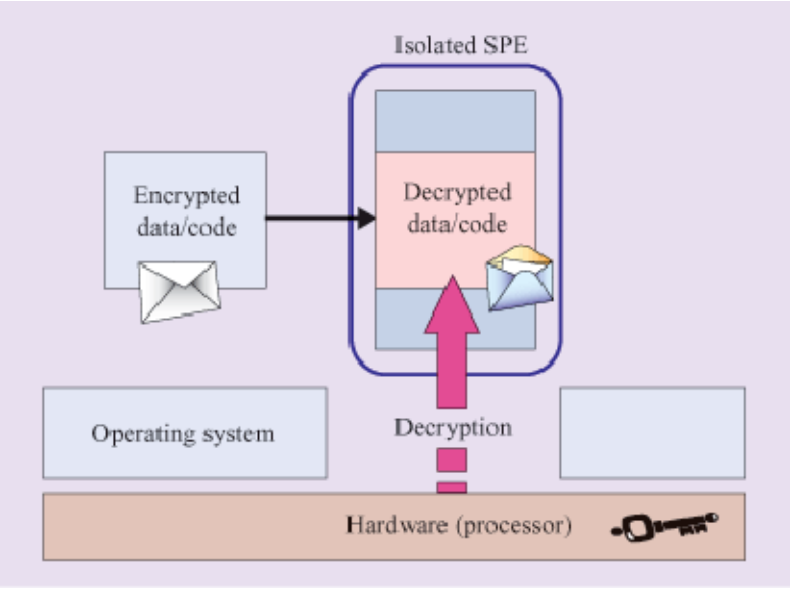

Figure 4

Hardware root of secrecy steps.

Figure 4

The term hardware root of trust is commonly used within the security community to refer to the root of an authentication chain, where application integrity is verified, while hardware root of secrecy is associated with the decryption chain, where application secrets are decrypted and controlled. The Cell/B.E. processor security architecture has the hardware root of trust and the authentication chain (described in the previous section), as well as the hardware root of secrecy and the decryption chain.

Another advantage of this feature is the answer to the question, What prevents an adversary from taking an application intended to run within the vault and executing it outside the vault? The answer is to encrypt a portion of the application code using the hardware root key. Because the code is encrypted, it cannot be captured and directly executed on a regular, nonisolated SPE. The code must be decrypted and, therefore, is forced to execute within the vault, where it can be decrypted by the root key. This reassures the application writer that a particular application will execute only within the secure processing vault.

Usage models

Encrypt-in, encrypt-out

The secure processing vault is best exploited by the "encrypt-in, encrypt-out" usage model (Figure 5) in which the incoming data is encrypted, an operation is performed on the data, and the data is re-encrypted before it is placed outside the vault. In this model, the data is in its vulnerable plaintext form only within the secure processing vault; the only code that has access to this plaintext data is that which is authenticated via the runtime secure boot; and the keys used for decryption and encryption are hidden from the system using the hardware root of secrecy. With this usage model of the vault, existing system functions such as file operations and network operations can be used "as is" without sacrificing security. Because the data is already encrypted by the time it is accessed as a payload to these operations, the secrecy and authenticity of the data can be guaranteed even if these system operations are somehow compromised. Any data with associated privacy or piracy concerns (e.g., rights-managed content or software, social security numbers, credit card numbers) can be robustly protected with this programming model. In addition, this model can be used for both streaming applications such as video decoding and nonstreaming applications such as database searches.

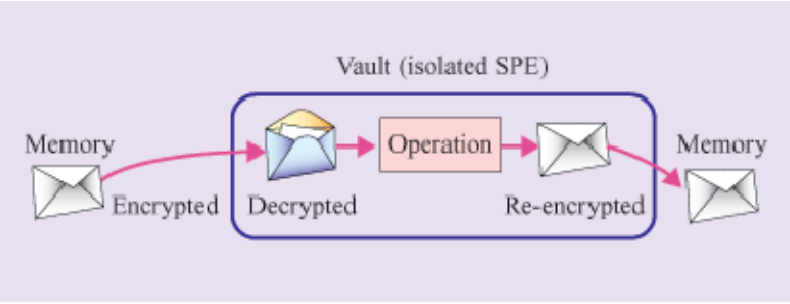

Figure 5

Encrypt-in, encrypt-out usage model.

Figure 5

Secure authenticated channel from the processing vault

System functions outside the vault are treated as part of the untrusted environment, and traditional cryptography-based methods are used for the application inside the vault to securely communicate with a server on the other side of the network (for network functions), itself at a later time (for storage functions), or another device in the system (for I/O functions). Thus, there is a clear delineation between the trusted application and the other, untrusted software on the system; this allows for a point-to-point trusted channel with just the application and not the entire platform with its many difficult-to-verify components (Figure 6).

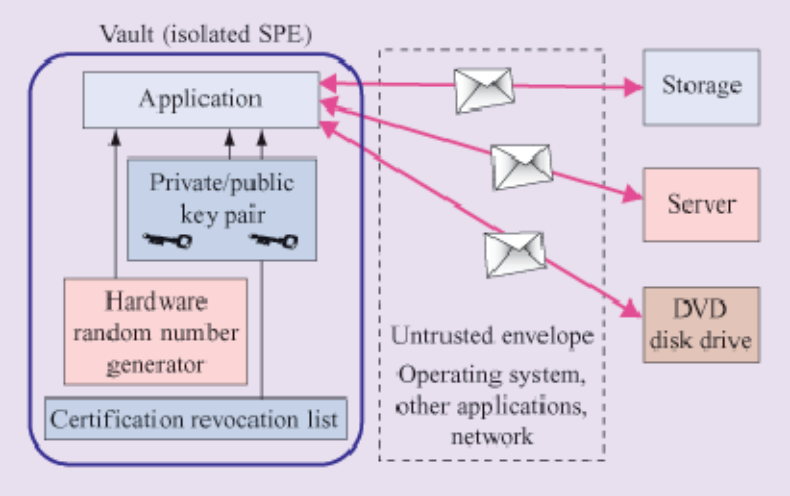

Figure 6

Anchor for secure authenticated channel—Secure Sockets Layer stack.

Figure 6

In addition to the usual use of a private and public key pair and a certificate revocation list, secure, authenticated communication is achieved by the three core security features and an on-chip hardware random number generator. For example, the vault feature can ensure that the process of authenticating its communication partner is not manipulated by an adversary. The runtime secure boot feature can protect the certificate revocation list from modification. The hardware root of secrecy can ensure that the private key is not exposed by an attacker. Also, the hardware random number can be used to protect against replay attacks by, in essence, marking the current communication with a timestamp. A replay attack is a situation in which an adversary takes an old communication message and resends it through the unsecured communication channel. Because the authentication protocol will verify that the message is authentic, a robust timestamp feature is the only way for the communication partners to realize that a "man-in-the-middle" attack is happening.

Attestation

Although earlier sections have implied this, it is worth explicitly stating that the combination of the hardware root of secrecy and the hardware-based authentication feature makes attestation possible. Attestation is a core requirement in trusted computing whereby a process can attest that it is indeed what it claims to be because it was able to access secrets that it otherwise could not have. With the vault architecture, only code images that have been

successfully authenticated are able to execute and access data that has been decrypted by the hardware root of secrecy. Because these secrets cannot be decrypted any other way, they can be used to attest to the remote party that the application executing in the secure vault is uniquely the owner of the secret.

Future work

Work is ongoing to develop a software infrastructure based on this processor security design. The processor provides only a secure foundation; the software design must provide the necessary flexibility, security, and usability. The software focus is on three main components: the SPE secure kernel, the operating system, and a build-time tool. The secure kernel is the software layer trusted to administer the secure application (the application that executes in the vault environment). For example, the secure kernel is entrusted to load and authenticate various secure applications (possibly from different sources), govern policy for permitting a process to run in the vault environment, and ensure that different applications running in the vault environment do not leak or steal secrets from one another. These software components will be released as part of the Cell/B.E. processor Software Development Kit.

In addition, a video streaming application is being developed to exploit both the security software infrastructure and the performance capabilities. In many cases, security and performance constitute a design trade-off decision. For example, with the encrypt-in, encrypt-out programming model, one can see how encryption and decryption steps can hinder performance. However, the application will demonstrate how the SPE performance for cryptography algorithms makes the programming model feasible and that security need not necessarily be compromised in order to meet performance requirements.

Acknowledgments

We thank all of the development engineers, programmers, architects, and management who made this work possible. In particular, we wish to thank (in alphabetical order) Dan Brokenshire, Alex Chow, David Craft, Michael Day, Jonathan Dement, Gilles Gervais, Sanjay Gupta, Aki Hatakeyama, Charlie Johns, Jim Kahle, Kenji Kikuchi, Mark Nutter, Mohammed Peyravian, Mack Riley, Bill Tiernan, Atushi Tsuji, Yukio Watanabe, and Emmanuel Zarpas.