Towards a General Architecture for Secure Computing [5]

Ahmed Mujuthaba
amuj003@ec.auckland.ac.nz
for CompSci725SC, University of Auckland.
21 October 2005

Abstract

We look at two recently proposed architectures for secure computing based on changes to the general purpose processor and system software. One of them protects the secrecy of cryptographic keys and the integrity of cryptographic functions that uses these keys. The other protects the secrecy of program code and related data in addition to protecting the integrity of the computational results of these programs. These two architectures are compared to learn the similarities and differences involved in providing secrecy and integrity protection to two different elements. We also look at the advantages and disadvantages of the two architectures. We then address the question of how general these architectures are.

1. Introduction

Secrecy, Integrity, Availability and Accountability have been identified as the four major needs of users with regards to information security [03]. A number of different approaches have been used in trying to satisfy one or more of these needs. These include hardware-only, software-only and combined approaches. A particularly successful approach that addresses the secrecy and integrity needs has been secure co-processors. However secure co-processors are currently used mainly by the military and large financial institutes.

Even though the stakes may be lower outside of military and finance industry when it comes to information security the need for securing information is there. Two recent proposals for addressing the secrecy and integrity needs build on the secure co-processor idea to provide similar functionality for use in general purpose processor based computer systems. Both of these do not include a co-processor but extend the general purpose processor along with some changes to system software to achieve the goal of providing secrecy and integrity.

Each of these chooses a different element for which secrecy is provided and integrity of computations using this secret element is guaranteed. In [01], the authors describe an architecture for protecting the secrecy of cryptographic keys and providing integrity of cryptographic functions using these keys. In contrast to this, [02] proposes an architecture for keeping secret program code (and hence algorithm) and related data and guaranteeing the integrity of the results from the program. Even though at first glance it seems the two architectures are solving two very different problems a closer look reveals the similarity of the two tasks and the solutions provided in the two papers. In this paper we study these two architectures as a way of finding out the issues involved in providing secrecy for a chosen element and guaranteeing integrity of computations involving the secret element. For the rest of the paper "SP-architecture" will refer to the one proposed in [01] and "Platte-architecture" will refer to the one described in [02].

This paper is organized as follows. In Section 2 we compare and contrast the main components of the two architectures. Section 3 looks at the advantages and disadvantages of each architecture compared to the other and also common to both. In Section 4 we look at whether each of the architectures can protect the secrecy of the element chosen by the other architecture, with a minimum of changes. In doing this we learn how general each of the architecture is and also get a glimpse of the issues involved in designing an architecture that is general enough to protect the secrecy of any element chosen by the user and ensure the integrity of computations involving the secret. This section is followed by the conclusion.

2. Comparison of the two architectures

For the purpose of comparing and contrasting the two architectures we'll look at the following aspects: key management, program protection, data protection, register protection, interrupt handling, operating system changes, and changes to programs and program development.

2.1 Key Management

The need for key management in SP-architecture is obvious since its aim is to protect the secrecy of cryptographic keys. Even though it's less obvious the Platte-architecture also need to deal with key management since it keeps programs and related data secret by encrypting them.

The SP-architecture uses the concept of a key chain, which is a tree structure where the keys that the user uses are at the leaf nodes of the tree and is encrypted with the key of its parent node. This allows the key chain to be stored in insecure locations because only the key at the root of the tree needs to be kept secret. This root key is called “User Master Key” and it unique to each user and is generated with secrets only the user posses. SP-architecture as described in [01] assumes this secret to be a password, although the authors acknowledge that other secrets unique to users can be used. In addition there is a “Device Master Key” which is created when the Trusted Software Module (TSM) is installed on the machine and is stored in a special secure register in the processor for the life time of TSM. The Device Master Key is used for program, memory and register protection as discussed in the following sections. Only the processor has access to the Device Master Key and it cannot be accessed by any instruction. Since the Device Master Key is installed during TSM installation there are no factory installed secrets in the SP-architecture.

In the Platte-architecture there is a factory selected public/private key pair. The private key of which is stored in the RSA unit of the processor. The public key signed by the manufacturer is made available to the public. Software vendors who want to protect the secrecy of their program code encrypt it with a secret key which is in turn encrypted with the public key of the target machine. When the encrypted program is loaded its encrypted secret key is loaded into the processor which decrypts it using its private key and stores the program secret key in a special register called “keystore” for the duration of the program execution. This key is then used to decrypt the program code for execution and also for program and memory protection as described in the following sections. Both the private key in the RSA unit and the key in the keystore are not accessible using any instructions and are used only by the processor internally. All together these allow for a program to be targeted to specific processors with the guarantee that no other processor can execute the code.

The approaches to key management taken by the two architectures are quite different, and this leads each of them having properties and/or capabilities that the other doesn't. The advantages and disadvantages of these are discussed in Section 3.1.

2.2 Program Protection

The goal of the Platte-architecture is protecting the secrecy and integrity of certain programs and hence it is obvious that it needs mechanisms to do this. SP-architecture with its goal of protecting cryptographic keys and the integrity of computations using them, it is not immediately obvious that it needs program protection. However to ensure key materials are not leaked out and to maintain the integrity of cryptographic functions that uses the keys, SP-architecture uses a Trusted Software Module (TSM). The integrity of the TSM is protected using the program protection mechanisms.

SP-architecture ensures the integrity of the TSM by hashing each cache line size segments of code using the Device Master Key and storing the hash values inline with the code. Whenever TSM instructions are fetched into the cache the processor recomputes the hash and compares it with the inline hash. However this only guarantees that the code itself has not been modified. It does not ensure the integrity of the result of running the code. To do this any memory data that the program uses during its run time should be protected against modification. Also to reduce the risk of the key materials leaking out the memory used to store data by the program must be prevented from being observed. How this is done is described in the following section on data protection. SP-architecture does not provide secrecy protection for the TSM. However it is easy to extend the architecture to provide this as discussed in Section 4. It must be noted that SP-architecture provides program protection only to the TSM and not to any other program.

The Platte-architecture provides both confidentiality and integrity protection to programs that are specially designed to take advantage of these features. Other programs run as normal without any of these protections. The special programs are encrypted and hashed at the time of distribution, using a symmetric secret key which is in turn encrypted using the processor's public key and included with the program. Unlike the SP-architecture the hash values are not included inline with the code. During the calculation of hash values the memory address of the lines are used in a way that makes each location unique. The hashes are loaded into a separate memory area when the program is loaded. When a cache line is brought into on-chip cache from main memory the processor checks the integrity of the cache line by calculating the hash value and comparing it to the stored value. If the values don't match the program is terminated immediately. If there is a match then the cache line is decrypted and executed. Hence except for the encryption/decryption the concept is similar to that of the SP-architecture. As mentioned before we need data protection in addition to program protection to ensure the integrity of the computation.

2.3 Data Protection

Both architectures use the same method to ensure data protection, even though the exact details defer. We won't go into the details except to point out some interesting differences.

Both architectures encrypt and hash any cache lines used by the protected program when the cache line is written back to main memory. The virtual memory address of the cache lines are included in the calculation hashes in a way that the same content in different memory locations gets different hash values. This allows the processor to detect any replay attacks. The SP-architecture uses the Device Master Key for encrypting and hashing while the Platte-architecture uses the program's secret key from the keystore for these. When a cache line is brought into cache, its hash is calculated by the processor and compared with the stored hash.

One difference between the two architectures is that since the SP-architecture uses hashed program cache lines that are not encrypted but data cache lines that are both hashed and encrypted, it needs some mechanism to identify instruction and data cache lines in the unified level-2 cache. SP-architecture uses a tag to identify the data cache lines in the level-2 cache. Platte-architecture does not need this because both program and data cache lines get encrypted and hashed.

Another important difference is that Platte-architecture uses a hash tree where the leaves of the tree are data and instruction cache line hashes and the parent nodes store the hash values of the children. Finally the root hash value is stored in a special encrypted cache line in the processor. This scheme prevents cache lines and their respective hash values from a previous execution of the same program being used in a replay attack. SP-architecture does not discuss in detail the method it uses for data hashes, instead suggesting that there are several possible implementations. However the architecture does not include a special register for the storage of the root hash, suggesting the authors have not considered the hash tree. Hence the SP-architecture does not provide protection against replay attacks using data and hashes from previous runs of the same program, where the replayed cache lines have the same virtual address as the cache line being replaced.

2.4 Register Protection

Since in the SP-architecture the processor registers are used only by the current running process, register protection is only needed when an interrupt occurs. This is discussed in detail in the next section. In contrast to this the Platte-architecture is based on the SPARC processor which uses register windows where each process uses a window of registers from the register set. This means mechanisms have to be implemented to prevent unauthorized register access by other processes. We omit the details of how this is done. However it is worth noting that this added complexity can be avoided by using a processor that do not use register windows.

2.5 Interrupt Handling

In SP-architecture when an interrupt occurs the processor first encrypts all the registers using the Device Master Key, treating them all as one block. The encrypted contents are then stored back in the registers. A hash value is calculated over all of the encrypted register contents. This hash value is then stored in a special register called the Interrupt Hash register. Control is then passed to the Operating System to handle the interrupt. When the protected program (TSM) again resumes execution, the processor first computes the hash over the restored register contents and compares it with the value in the Interrupt Hash register. If they do not match then the program is terminated. If they match the register values are decrypted and execution continues. Since the all the registers are encrypted and hashed as one block the only replay that will be successful is one which is exactly the same register contents, in which case it won't cause any problem. To enable the processor to identify a return back to the TSM the return address is stored in a special register before passing control to OS. On a return this address is checked.

The Platte-architecture has a completely different approach to interrupt handling. This is partly due to it being based on the SPARC architecture (see Section 2.4). Interrupts that occur while executing a protected program are handled by a separate set of interrupt handlers. When the current executing program is not a protected program the interrupts are handled by the normal interrupt handlers. Interrupt handlers for the protected programs has the beginning and end of the routine as protected regions which are entered and leaved using special instructions. Before leaving the protected region at the beginning the interrupt number and frame pointer is stored in a special protected memory area. After the interrupt routine does it work it enters the protected region at the end of the routine at which point the interrupt number and frame pointer is compared with the stored values. The protected program resumes only if they match. [02] does not have any discussion of encrypting and hashing register values when they need to be stored on the stack.

Refer to Sections 3.2 and 3.3 for discussions of some issues arising from the two approaches to interrupt handling.

2.6 Operating System Changes

Both architectures require changes to the Operating System in addition to the changes to interrupt handling discussed above. In both cases the way the protected programs are loaded is different to loading of normal programs due to the encrypting and hashing of data. Also in the case of the Platte-architecture the loader needs to be able to identify protected programs and pass the encrypted secret key associated with the program to the processor’s RSA unit.

The way the OS handles paging also need to be modified to ensure that data and their respective hash values are paged in and out together despite them sitting in different memory areas. This point is made explicit in [02]. Although this is not mentioned explicitly it is clear this would help in the SP-architecture as well.

2.7 Changes to programs and program development

Since the SP-architecture regards the OS as completely untrusted and the Platte-architecture do not trust the OS except for certain parts of interrupt handlers, both [01] and [02] recommends that the protected programs be statically linked. In addition [01] recommends the memory to be used for data by the TSM be statically allocated during compile time. The Platte-architecture allows dynamic memory allocation for protected programs during run time. However [02] recommends that the program itself include checks to verify the virtual addresses returned by the OS after a *malloc* call. The Platte-architectures also specifies that libraries be modified to allow programs to request encrypted, protected or unprotected memory. And finally the protected program must be encrypted and hashed in the case of Platte-architecture after compilation and linking. In the SP-architecture the hashing of TSM occurs during installation.

2.8 Secure I/O

The SP-architecture requires secure I/O for securely transferring the password (see Section 2.1) that is used to generate the User Master Key, to the processor. The authors propose encrypting the data between the keyboard and the secure I/O unit of the processor when the user presses a button before entering the password. However which key is used for this encryption process and how the key is shared between the keyboard and processor is not mentioned at all.

The Platte-architecture does not require the user entering any passwords and require all protected programs be distributed with the key used to encrypt them (see Section 2.1). Hence it has no mechanisms for secure I/O and it is not discussed in [02].

Issues relating to secure I/O implementation in the SP-architecture and the disadvantages of Platte-architecture not including secure I/O are further discussed in Section 3.4.

3. Advantages and Disadvantages

In this section we look at the advantages and disadvantages of the two architectures relative to each other and also common to both.

3.1 Factory installed device secret vs. no factory installed secret

As detailed in Section 2.1, the SP-architecture does not include a factory installed device secret. In contrast the Platte-architecture has the private key of a public private key pair installed in the processor by the manufacturer.

The authors of [01] claim that having a factory installed key gives rise to privacy concerns and also limits the portability of trust. A factory installed key (which has to be unique for it to work) does indeed pose privacy problems in that other parties may use it to identify the owner of a machine. The outcry over the Pentium III Processor Serial Number (PSN) [04] illustrates the general public’s stance on this type of issue. The second claim by the authors that a factory installed device secret limits portability of trust has no merit at all. The reason is that in the SP-architecture there is a device secret (the Device Master Key) even if it is not factory installed. The Device Master Key is used by the processor to encrypt and/or hash the TSM and data used by it. The portable trust in SP-architecture derives from the User Master Key. Hence even if the Device Master Key was replaced by a factory installed device key in the SP-architecture the portability of user key chain will not be affected at all.

The Platte-architecture with its factory installed private key in the device and the manufacturer signed public key available to public offers more benefits compared to the SP-architecture. The most obvious benefits are mentioned in [02]. One is software distributors are able to send the secret key with which they encrypted the program along with the program by encrypting using the public key of the processor. The other is that using this software distributors are able to control the specific machines their program will run. This maybe desirable in some distributed computing applications. Since the SP-architecture does not run protected programs other than the TSM the above mentioned advantages do not apply to it. However there is another benefit of using the private/public device key scheme that is relevant to both architectures. That is, it will enable the processor to authenticate itself to a user who uses a smartcard. This is highly desirable in the case of SP-architecture, since one of its aims is to make user trust portable and in such a setting the user would want to know if it is a SP-processor that she is giving her password to.

3.2 Untrusted OS vs. Partly trusted OS

The SP-architecture considers the OS to be completely untrusted and hence do not rely on it for any of the security mechanisms. This has the clear advantage of keeping the protected elements safe in the face of an OS compromise.

Due to how it handles interrupts, the Platte-architecture has to partly trust the OS. The only parts of the OS that are trusted are certain parts of interrupt handling routines. However this exposes the Platte-architecture to more risks in the face of an attack that compromises the OS.

3.3 Running multiple protected programs

The SP-architecture was designed to run only the TSM as the protected program. The way it handles interrupts using the return address register (see Section 2.5) means only one protected program can be supported even in a multiprogrammed environment. In addition the processor includes special flags to ensure that only one copy of the TSM is running at a time. These are not disadvantages when considering the purpose for which the SP-architecture was intended to use. However these are disadvantages if we consider extending the architecture.

On the other hand the Platte-architecture can run any program designed to take advantage of it, as a protected program. However, though never explicitly or implicitly mentioned in [02] it can not run multiple protected programs at the same time. This is due to the architecture not having any mechanism to handle key changes in the keystore during context switches. This is a disadvantage since the architecture otherwise allows multiprogramming, and it maybe desirable to run multiple protected programs at the same time.

3.4 Secure I/O

As mentioned in Section 2.8 the method for secure I/O given in [01] omits very important details about sharing keys between the keyboard and the processor. This highlights the difficulty of implementing secure I/O. However secure I/O is essential for the working of the SP-architecture.

The lack of secure I/O in the Platte-architecture disallows it from being used in applications in which secure interaction with users are needed such as the user key management scheme proposed to be used with the SP-architecture in [01]. This is a clear disadvantage.

4. A General Secure Computing Architecture

We know look at the problem of making each of the architectures protect the secret the other was designed to protect. The main purpose of looking at this problem is to get some idea of how general these architectures are. This in turn helps us understand the issues involved in designing architectures that aim to solve multiple security needs of users.

From the comparisons in Section 3 and the discussions in Section 4 it is clear that the SP-architecture cannot provide protection to programs other than the TSM. It provides integrity protection to the TSM but does not provide secrecy protection. However it can be easily modified to provide secrecy protection as well by encrypting the TSM using the Device Master Key and decrypting the cache lines after a successful hash match. The changes required to allow this architecture to provide protection to programs other than the TSM, are quite extensive. One major change needed is changing of the device key to a factory installed private/public key pair as discussed in Section 3.1. Without this change distributors of the protected programs cannot distribute the key with which the program is encrypted. Hence some major changes to the design of the architecture are necessary in order to make it suitable to handle the task the Platte-architecture solves. That is providing secrecy and integrity protection of programs.

On the other hand the changes to the Platte-architecture that is necessary in order for it to support the key chain concept and the protection of the key chain, seems not so extensive. The TSM can be run as another protected program in the Platte-architecture. The only changes required are adding a register to store the User Master Key and the addition of secure I/O. However as we have seen from Section 3.4 implementing secure I/O is not so easy. Nevertheless implementing secure I/O in the Platte-architecture would be no different to implementing it in the SP-architecture.

It is obvious from the above discussion that the Platte-architecture can be extended easily to add support to provide secrecy and integrity protection to elements other than programs. Hence it seems possible in theory at least to design an architecture that is general enough to provide solutions to different security needs of users. However as we noted at the start, these architectures only provide secrecy and integrity protection. The other two categories of security needs identified in [03], availability and accountability are not addressed. Hence to be able to provide a truly general architecture for secure computing, further work needs to be done to incorporate availability and accountability into these architectures.

5. Conclusion

We have seen by looking at the two architectures that is possible to provide solutions to information security needs based on small changes to the general purpose processor and system software. We have also seen that it is possible to design a secure computing architecture that provides secrecy and integrity protection to multiple and different elements that require them. However availability and accountability needs to be incorporated into future designs in order to fully address the security needs of users. Further work is also needed in the area of secure I/O as it is an important element in maintaining security in interactive computing tasks.

6. References

[01] Lee, R.B., Kwan, P.C.S., McGregor, J.P., Dwoskin, J., Zhenghong Wang, "Architecture for Protecting Critical Secrets in Microprocessors", ISCA '05, Proceedings of the 32nd International Symposium on Computer Architecture, 2005, Pages:2 - 13, June 2005.

[02] Platte, J., Naroska, E., "A combined hardware and software architecture for secure computing", Conference On Computing Frontiers, Proceedings of the 2nd conference on Computing frontiers, Ischia, Italy, SESSION: Track 13: special purpose architectures, Pages: 280 - 288, May 2005.

[03] Lampson, B.W., "Computer security in the real world", Computer, Volume 37, Issue 6,Pages:37 - 46, June 2004.

[04] http://www.sims.berkeley.edu/courses/is224/s99/GroupG/psn_wp.html. Accessed on 20th October 2005.

[05] Title based on suggestion by Clark Thomborson.