High Speed Memory Centric Protection on Software Execution Using One-Time-Pad Prediction

July, 19, 2004

Weidong Shi Hsien-Hsin Sean Lee Chenghuai Lu Mrinmoy Ghosh
801 Atlantic Drive
Atlanta, GA 30332-0280
Georgia Institute of Technology
shiw,leehs,lulu,mrinmoy@cc.gatech.edu

ABSTRACT

This paper presents a new security model for protecting software confidentiality. Different from the previous process-centric systems designed for the same purpose, the new model ties cryptographic properties and security attributes to memory instead of a user process. The advantages of such memory centric design over the previous process-centric design are two folds. First, it provides a better security model and access control on software confidentiality that supports both selective and mixed software encryption. Second, the new model supports and facilitates information sharing in an open software system where both confidential data and code could be shared by different user processes without unnecessary duplication as required by the process-centric approach. Furthermore, the paper addresses the latency issue of executing one-time-pad (OTP) encrypted software through a novel OTP prediction technique. One-time-pad based protection schemes on data confidentiality can improve performance over block-cipher based protection approaches by parallelizing data fetch and OTP generation when a sequence number associated with a missing cache block is cached on-chip. On a sequence number cache miss, OTP generation can not be started until the missing sequence number is fetched from the memory. Since the latency of OTP generation is in the magnitude of the order of hundreds of core CPU cycles, it becomes performance critical to have OTP ready as soon as possible. OTP prediction meets this challenge by using idle decryption engine cycles to speculatively compute OTPs for memory blocks whose sequence number are missing in the cache. Profiling and simulation results show that significant performance improvement using speculative OTP over regular OTP under both small 4KB and large sequence number cache settings 32KB due to the capability of speculative OTP technique to reduce misses on sequence number. The performance improvement is in the range from 15% to 25% for seven SPEC2000 benchmarks. The new access control protection and OTP prediction scheme requires only small amount of additional hardware resources over the existing proposed tamper resistant system but with greatly improved performance, protection, flexibility, and inter-operability.

1. INTRODUCTION

Recently, there is a growing interest in creating tamper-resistant/copy protection systems that combine the strengths of security hardware and secure operating systems to fight against attacks [15, 9, 22, 23, 24, 16]. Those systems usually possess strong security features and are able to prevent running applications from malicious software as well as physical hardware attacks. Such tamper resistant systems have a great future for solving various problems in the security domain such as digital rights protection, virus/worm detection, system intrusion prevention, digital privacy and etc. For maximum protection, tamper-resistant/copy protection system should be able to provide protection against both software and hardware based tamper including duplication (copy protection), alteration (integrity and authentication), and reverse engineering (confidentiality). It is important to note that this definition of a secure system is much stronger than many previously proposed secure systems and protection schemes such as capability based systems like Hydra [7] .

Many the above mentioned copy protection systems achieve protection by encrypting the instructions and data of a user process (every bit of information in the user's virtual space) with a single master key. Although such closed systems do provide security for software execution, they are less flexible for real world applications because of the gap between a closed tamper-resistance/copy protection system and a real world software system. Most real world software environment can be best described as a multi-domain system. In such a system, a user process often consists of program components coming from heterogeneous sources with diversified security requirements. For instance, almost every commercial application in Windows system uses some number of dynamically linked libraries (DLL) or statically linked libraries. Our study shows on average of 20-30 DLLs used by commercial applications. When the dynamic libraries are provided by middle-ware vendors, it is quite natural to expect that vendors of these libraries prefer a separate copy protection of their intellectual properties (IP) from the user applications. Some of the DLLs are Windows libraries that are mapped to many applications' virtual space thus not possible to be encrypted by a single application's cryptographic key. Although security need may be met by duplicating DLLs in each application's virtual space using the per-process cryptographic key, it is not only in-efficient but also may cause synchronization problems because many shared DLLs are responsible for maintaining system resources. The nature of de-centralized development of software components by different vendors makes it difficult to enforce a process centric protection scheme.

It is important to point out that protection provided by virtual memory system does not solve the problem because we assume that all the data and software components belonging to different security domains co-exist in the same virtual space. Though traditional capability based protection systems such as Hydra [7] and CAP [17] provide access control on information, they are not tamper resistant systems designed to protect against software duplication, alternation, and reverse engineering. Specifically, systems such as Hydra and CAP do not address the following problems:

- Integrity issue: How access control interacts with other protections provided by a copy protection system. Hardware supported integrity checking and encryption are two absolutely necessary components for a tamper resistant system that protects against physical based tampering of software integrity and confidentiality.
- Implementation issue: How control on information sharing and access can be implemented efficiently at architecture level and tied to a modern day out-of-order processor pipeline.
- Performance issue: Capability based control on information access is costly to implement and could not satisfy the performance goal demanded by a modern day computing systems.
- Programming issue: A full blown capability based system requires a different programming model and complicated security policy management, which prohibits its adoption by the industries.

In order to provide high speed protection for software in open systems, a number of challenges on both the security side and the performance side have to be addressed,

- Challenge of fine-grain security protection. A fine-grain protection on process memory must be implemented so as to protect different software components from hackers as well as other un-trusted software components. This fine-grain security protection should allow to different security requirements for different software components, by providing facilities such as selective encryption and mixed encryption. Selective encryption means that only selected regions of a user memory space are protected with encryption and mixed encryption refers to that user space can be partitioned into sub-spaces, each encrypted using different keys.
- Challenge of access control. Since all the software components and their data reside in the same virtual space, access control has to be proposed to safeguard each protected domain so that information confidentiality can not be compromised.
- Challenge of low overhead switching between security domains. Because all the software components are in the same virtual space, switching between protected domains would be far more frequent than switching between application mode and the kernel mode, which often involves heavy handed system calls. Thus a light weighted switching mechanism is required.
- Challenge of low data decryption latency. Decryption latency plays a major role in performance of a secure system that supports confidentiality protection. OTP based protection schemes have been proposed recently to meet this challenge by parallelizing data fetch and decryption (OTP generation). But, a sequence number associated with each data block has to be cached inside the chip. When the sequence number is missing in the on-chip cache, it has to be brought into the secure processor, possibly decrypted first, then used to generate OTP using a standard cryptographic function often taking hundred of cycles to complete. After all these, the result OTP can be applied to the encrypted data. Large size sequence number cache can improve performance but add significant cost on chip size.

In this paper, we present a new framework called *MEmory-Centric Security Architecture* or *memory centric tamper resistant /copy protection system* which provides protection on software integrity and confidentiality using a new set of operating system and architecture features that support secure software execution in a heterogeneous multi-domain, multi-level security system. In this system, a concept, called memory module, is used as the basic atomic unit for managing security. Memory module refine security control over the existing systems. Based on the per module security attributes, access control can be defined to prevent unauthorized access on confidential information stored in a memory module. On the performance side, a novel OTP prediction technique is presented for the first time to further reduce the latency overhead of confidentiality protection. It is important to point out that the OTP prediction technique does not trade security for performance. It is an architecture optimization and equally secure as a regular OTP based protection.

The major contributions of our work are:

- A unique memory centric protection system on software confidentiality and integrity that is different from the existing systems (e.g., XOM, AEGIS) by providing better support for fine-grain information sharing in a multi-domain, multi-task tamper resistant environment.
- A unique secure processor model that protects software components from one another to prevent illegitimate information access across different tamper resistant domains. Different from the previous systems on fine-grain access control such as Hydra and CAP, it is simple, efficient in performance with support of *access control speculative execution* (ACSE), light-weight in implementation, integratable, and transparent (require almost no change on the current programming model).
- A novel *OTP prediction* technique that significantly reduces the latency overhead on fetching OTP encrypted data from memory.

The rest of the paper is organized as follows. In section 2, we give a high level introduction to our security model. This section paves the road for the detailed presentation of each component of the memory centric security model and its architecture support in section 3, followed by evaluation and

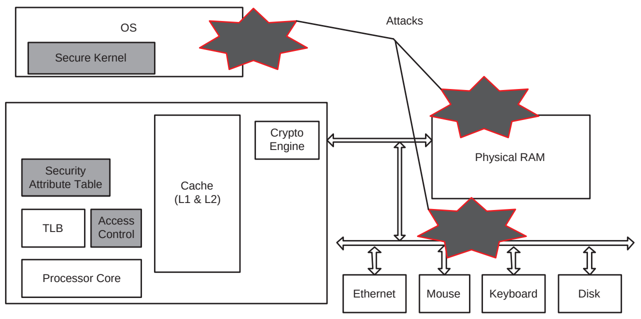


Figure 1: Memory-Centric Security Architecture

results. Discussion of related work is presented in section 5 and finally section 6 concludes the paper.

2. MEMORY-CENTRIC SECURITY ARCHITECTURE

In this section, we give an overview of our security model, called *MEmory-centric Security Architecture* (MESA) in contrast to the process-centric security model in prior work. Figure 1 generalizes the security model and its operating environment. It highlights certain assumptions made by MESA,

- It is assumed that everything outside the CPU is unprotected and subject to malicious tampering. The physical RAM itself is neither protected and hackers could read/overwrite the memory content directly without involving the CPU. Furthermore, all the system/peripheral bus traffic is exposed and could be traced by the hackers.

- Like other tamper resistant systems, there is a pair of public-private keys associated with each secure processor. The secure processor's private key is permanently burnt into the processor core and could not be accessed by software [15].

- MESA assumes hardware supported encryption/decryption and integrity check. When a cache line size memory block of either data or instructions is brought into the secure processor, it is decrypted and integrity of the entire virtual memory space is verified using hash tree or MAC tree [21, 1]. When a cache line is evicted from the secure processor, it is encrypted and hash/MAC tree is updated. The keys used for encryption/decryption and integrity verification are set by the software vendors and encrypted by the secure processor's public key.
- MESA assumes the existence of a secure BIOS stored in a securely sealed persistent storage device. Execution of the secure BIOS is protected by tamper resistant measures.

- Most of the OS codes are treated as regular program except a small set of core services, called secure kernel. The secure kernel is signed and authenticated by the secure BIOS during system boot [3]. Furthermore, integrity code is computed for each cache-line size block of instructions of the secure kernel and verified each time it is brought into the processor, therefore un-tamperable. To further enhance security, secure kernel can be implemented as firmware stored in either an on-chip ROM or an off-chip securely sealed persistent memory.

- All the security process context such as keys, root signatures, and etc defined in this paper are managed by the secure kernel and securely preserved during process context switch.

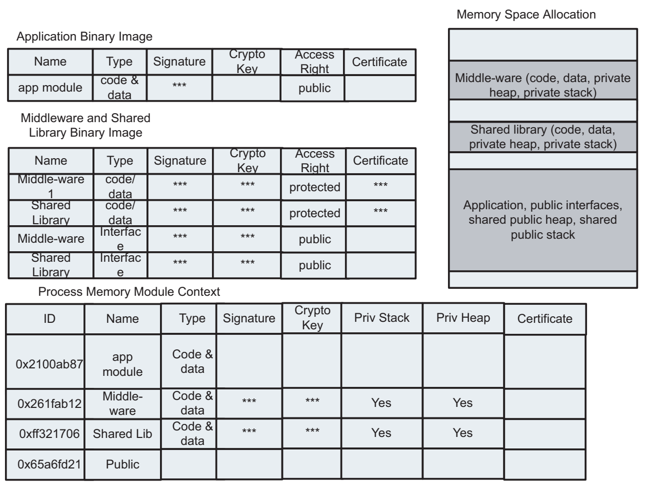
- A secure processor can either run at debug mode or retail mode. In debug mode, decrypted software can be traced. But in the retail mode, processor exceptions and traps used for debugging are all disabled, thus preventing users from tracing the software. A bit for setting the mode is defined as part of the signed binary image of an application.

First in this section, we describe MESA from system perspective. The focus would be to present a global picture how MESA operates as a system. In the next section, we will present efficient implementation of MESA at architecture level.

One critical concept MESA uses is memory module. A memory module is a virtual memory segment initialized with a set of security attributes provided by the binary images that are loaded into it. It is an information container that may hold either data or code or both. A set of security attributes are defined for each memory module beside its location and size and initially set by system or software vendors. These security attributes include security protection level, one or more symmetric encryption keys encrypted using secure processor's public key, authentication signature (MAC or hash tree), accesses control information, etc. For each application process, the secure OS kernel maintains a list of memory modules and their attributes as process context. The security attributes set by software vendors can be encrypted and authenticated using a processor's public key. They are decrypted and verified by the secure processor using the corresponding private key. Security attributes are protected when they are stored in the external RAM.

Based on the security attributes, access control of memory module can be carried out. Active module is defined as the currently executing module. When the active module is to access some memory location of some other module, the access will be checked. If the active module is allowed to access the memory module, the access will be granted. Otherwise, security exception of access violation will be raised. There are two basic types of modules, private and public. Private modules can be called by other modules but direct access to its data or instructions is disallowed. Public module can be accessed by any other module. Note that public modules may be different on other security attributes.

Secure OS kernel is responsible for managing memory modules at OS level. Among the major services provided by the secure kernel are, process and *module* creation, *module* authentication, access control. We will discuss them one by one.


Memory Space Allocation
Middle-ware (code, data, private heap, private stack)
Shared library (code, data, private heap, private stack)
Application, public interfaces, shared public heap, shared public stack

Application Binary Image

Name	Type	Signature	Crypto Key	Access Right	Certificate
app module	code & data	***		public	

Middleware and Shared Library Binary Image

Name	Type	Signature	Crypto Key	Access Right	Certificate
Middle-ware 1	code/ data	***	***	protected	***
Shared Library	code/ data	***	***	protected	***
Middle-ware	Interfac e	***	***	public	
Shared Library	Interfac e	***	***	public	

Process Memory Module Context

ID	Name	Type	Signature	Crypto Key	Priv Stack	Priv Heap	Certificate
0x2100ab87	app module	Code & data					
0x261fab12	Middle-ware	Code & data	***	***	Yes	Yes	
0xff321706	Shared Lib	Code & data	***	***	Yes	Yes	
0x65a6fd21	Public						

Figure 2: Secure Memory Module Management

2.1 Memory Module Creation

Firstly, during process creation, the secure OS kernel will create a list of memory modules associated with the process. Figure 2 shows an example application. Assume that an application developer will use compiled binaries from two different sources. Both of them want the linked libraries to be protected from reverse engineering or tampering. Each vendor will provide two binary images. One image is encrypted using a key chosen by the corresponding vendor plus a root binary image authentication signature computed using hash tree or MAC tree (message authentication tree) algorithm. The other one stores a public interface to the encrypted codes for the application developer to use. Details of how to perform encryption and authentication have been studied recently in [15, 9, 22]. Each vendor independently sets the security attributes of released binary images. During process creation, the secure kernel creates a secure memory module context based on the binary images. Each memory module is uniquely identified with a randomly generated ID. For the example, there could be three modules if the application itself does not demand special protection.

For dynamically linked libraries, a different module is created with a different ID when it is linked to a different process. However note that the code itself is not copied. It is simply mapped to the new process's memory space with a different module entry in the module context table.

Heap and stack are two types of dynamic memory that can be owned by a module. Privacy of information stored in the heap and stack is protected under the same security requirement of the module they belong to. When execution switches to a different module, the processor stack register is re-loaded so that it will point to the next module's private stack. Details of private stack are presented in the subsection of cross module procedure call.

With a private heap, there comes the issue of memory management of private heaps associated with each module. Does each module need its own heap allocator? The answer is no. Heap management can be implemented in a protected shared system library. The key idea is that with hardware

Intrinsic	Parameters	Explanation
sec_malloc(s,id)	s: size; id: module id	allocate memory from module's private heap
sec_free(p)	p: memory pointer	free memory of private heap
sec_swap_stack (addr)	addr: address	switch the active stack pointer to another module's private stack. Addr points to a location of the target module. Save <active stack pointer, active module id> to the stack context table
sec_get_id (name)	name: module name	get id of a module (secure kernel service)
sec_push_stack_ ptr()		read the current executing module's stack pointer from the stack context table and push it into its private stack
sec_save_ret_addr (addr)	addr: address	assign addr to a return address register (RAR)
sec_return()		assign RAR to PC and execute

Figure 3: MESA Security Intrinsics

supported integrity and confidentiality protection, the heap manager can manage usage of each module's private heap but could not tamper its content.

2.2 Memory Module Authentication

MESA supports three possible ways of *module* authentication. The first approach is to authenticate a binary module through a chain of certification. A module could be signed and certificated by a trusted source [12]. If the certification could be verified by the secure kernel, the created *module* would become a trusted *one* because it is certified by a trusted source. Another approach is to authenticate a *module* using a public key supplied by software vendors. This provides a way of private authentication. This approach is designed for multi-process, multi-user application to authenticate each one for sharing information. The third approach is to certify *module* using processor's public key. To give one example using figure 2, application vendors can specify that the linked shared libraries must be certified by a known source such as Microsoft. Failure of authenticating code images will abort the creation of the user process. Public key based authentication has been intensively studied in the area of distributed computing and explained in detail in [14, 6].

2.3 Cross Module Procedure Call

When a module is to call a function implemented by another module, it passes all the parameters by copying them to the callee's private stack. To show how private stacks are protected during cross-module function call, we have to show function call in assembly. It is given in figure 4 using x86 instruction set and MESA programming primitives and intrinsics listed in figure 3. Note that these security intrinsics are programming primitives not instruction set. Although a hardware implementation of MESA can implement some of them as CISC instructions, it does not have to be that way.

In the example, the caller pushes values to the callee's private stack. When callee's stack requires information be encrypted, pushed stack values will be encrypted with callee's crypto key. Switching the active stack pointer from the caller's private stack to the callee's private stack is achieved through intrinsic *sec_swap_stack(addr)*. Input addr is either a function entry address or return address. MESA maintains a table of stack pointer context for all the running modules. When *sec_swap_stack(addr)* is executed, it will save the current active stack pointer as a $<$ module, stack pointer $>$ pair and set the active stack pointer to the target module's. The

```
// CALLER SIDE
push ebp /* save stack frame pointer */
sec_swap_stack addr_of_fun_foo
// stack pointer switched to the callee's stack
// caller's esp saved to the context table
push 0x10; /* parameter */
push 0x20; /* parameter */
call fun_foo /* push return addr to callee's stack */
pop ebp /* get stack frame pointer back */

// CALLEE SIDE
sec_push_stack_ptr
ebp = esp //ebp stack frame pointer
...
r1 = [ebp + 4] //return address
sec_save_ret_addr r1 // save caller's return
esp = [ebp]
sec_swap_stack r1
// stack pointer restored pointing to the caller's stack
// callee's eps saved to the context table
sec_return
//return to caller by loading return addr in RAR to PC
```

Figure 4: Cross Module Function Call

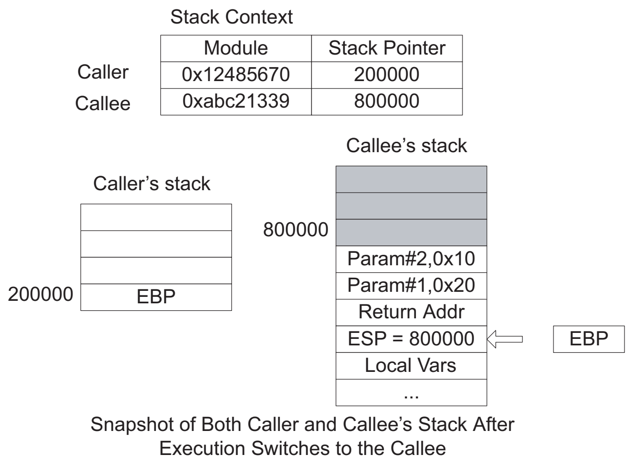


Figure 5: Snapshot of stack after execution of Code

snapshot of the caller and callee stack after the execution of the code is illustrated in figure 3. It is important to note that MESA protects against tamper on the target module's stack by only allowing values to be pushed to other module's stack. A module can not modify another module's stack pointer context because only the owner module can save the active stack pointer as its stack pointer context according to the definition of *sec_swap_stack*. Explicitly assigning values to the active stack pointer owned by a different module is prohibited by MESA.

3. ARCHITECTURE SUPPORT FOR HIGH PERFORMANCE MESA

This section discusses architecture features for supporting MESA. Inside a typical secure processor, we added a few security features at the micro-architectural level which incorporates encryption schemes as well as integrity protection schemes. Hardware enabled protection on integrity and confidentiality has been sufficiently studied in [15, 9, 22]. In addition to the architecture components necessary for integrity and confidentiality protection, we introduced new micro-architecture components for the MESA including a *Security Attribute Table (SAT)*, an *Access Control Mechanism (ACM)*, and *OTP sequence Number Prediction* support. New system features are also proposed to cope with the new security architecture to manage the secure architecture asset.

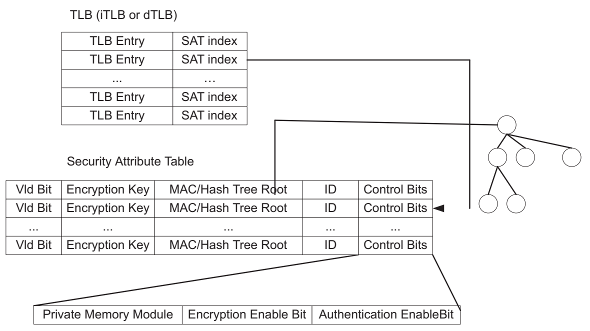


Figure 6: SAT (security attribute table) and TLB

3.1 Security Attribute Table

Secure memory module management lies at the heart of MESA. The secure OS kernel keeps track of a list of memory modules used by a application. Security attributes of frequently accessed secure memory modules are cached on-chip in a structure called SAT (security attribute table). Figure 6 shows the structure of SAT attached to TLB for instruction or data. Each entry caches a set of security attributes associated with a secure memory module. The Encryption key of each SAT entry is used to decrypt or encrypt information stored in the memory module. The secure kernel uses intrinsic *sec_SATld(addr)* to load security attributes from the memory module context stored in memory to the SAT. The Encryption keys in the memory module context are encrypted using processor's public key. The secure processor will extract the keys from the process's memory module context when they are loaded into SAT.

A secure memory module could be bound to one or many virtual memory pages of a user process. When the entire user virtual memory space is bound to only one secure memory module, the model is equivalent to a process-centric security model. As figure 6 shows, each TLB entry contains an index to SAT for retrieving the security attributes of the corresponding memory page. During context switches, the secure kernel authenticates the process's memory module context first, then load security attributes into the SAT. The SAT is accessed for each external memory access. For a load operation, if a cache miss occurs, data in the external memory will be brought into the cache, decrypted using the encryption key in the SAT and its integrity verified against the root authentication signature also stored in the SAT using hash tree [9] or MAC tree [1]. On-chip caches store and maintain only plaintext data. The SAT is also accessed when data is to be evicted from the on-chip caches. The evicted data will be encrypted using keys store in the SAT and a new root signature is computed.

If the required security attributes could not be found in the SAT, a SAT miss fault is triggered and the secure kernel would take over. First, the secure kernel would flush the cache, then load the required security attributes into the SAT. The SAT indexes stored in the TLB are also updated

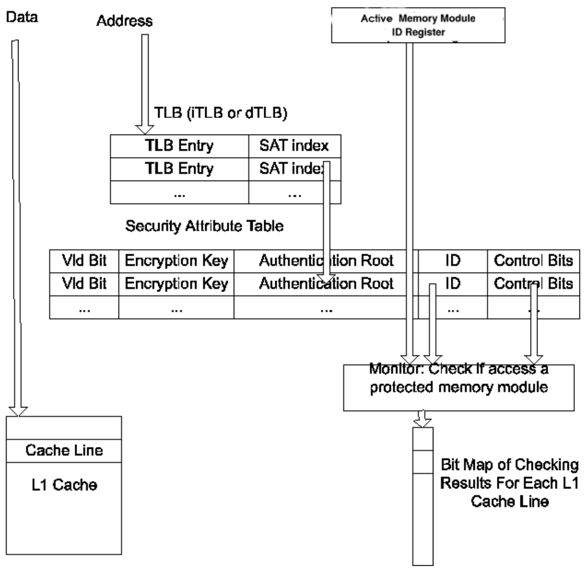


Figure 7: Information Security Monitor

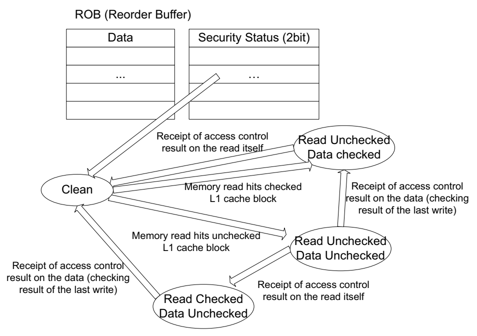

ROB (Reorder Buffer)
Data
Security Status (2bit)
...
...
Clean
Read Unchecked Data checked
Read Unchecked Data Unchecked
Read Checked Data Unchecked
Receipt of access control result on the read itself
Memory read hits checked L1 cache block
Receipt of access control result on the data (checking result of the last write)
Memory read hits unchecked L1 cache block
Receipt of access control result on the data (checking result of the last write)
Receipt of access control result on the read itself

Figure 8: Security Speculative Execution

```
r1 => memory address l1
r2 <= memory address l1
r3 = r2 + 100
```

Figure 9: Example of Access Control Speculative Execution

accordingly. A SAT miss is expected to happen very rarely. It is at user's control to choose the right number of protected memory modules so that SAT miss can be completely eliminated. For example, if there are 8 SAT entries supported in hardware, it means that at most 8 protected domains can be executed without causing SAT misses. This should be more than enough because the chance of finding programs from 8 different sources in the same application's virtual space is extremely low. Although it is possible, but not recommended to protect programs from the same vendor in different memory modules. Performance wise, vendors should always minimize SAT entry usage by packaging all the data and codes that require protection together.

Figure 6 assumes a virtually indexed cache. For physical address tagged cache, an inverse translation table is needed.

3.2 Access Control Mechanism

Efficient hardware supported access control plays a key role for protecting memory modules from being accessed by un-trusted software components. It is important to point out that encryption of a software component or memory module does not mean that it can be trusted. A hacker can encrypt a malicious library and have the OS to load it into an application's virtual space. The encrypted malicious library despite encrypted can illegally access confidential data deemed to be accessed by only the application program.

Access control is achieved through an access monitoring mechanism shown in figure 7. The proposed architecture protects confidential information of a protected memory module from being accessed by un-trusted codes with minimal impact on performance.

To minimize performance impact of access control on memory accesses, the proposed architecture conducts checking on access violation in parallel with storing information into the cache, thus incurs almost no performance loss. As shown in figure 7, there is an access control bit map (ACBM) where each bit corresponds to a L1 cache line. Each bit of ACBM denotes whether the stored data in the associated L1 cache line has passed access control security checking. When data is written to a cache line, the bit will be clear. When access control reports no violation, the bit will be set. To guarantee that a protected memory module is not updated by instructions who do not have write access right, cache lines with clear bits in ACBM are inhibited from being written back to L2 or the external memory. If an access violation is detected, the proposed access control mechanism will raise an access control exception. To support precise interrupts on access control exception, write instructions are not to allowed to retire before access control checking is completed. After the access monitor finishes checking on a memory access, it will send the result to the memory unit so that the waiting memory write can be retired.

On the performance side, access control requires only one TLB access, one SAC access, and simple comparisons. It can be completed in 2 CPU cycles with SAC access and comparison combined into one cycle considering SAC is accessed through index. Furthermore, to minimize any potential impact on performance, a speculative execution mechanism is proposed to allow processor pipeline to read unchecked data stored in the L1 cache.

We explain the Speculative Execution Mechanism using the short code sequence in figure 9 as example. Assume that data in r1 is stored to L1 cache and when the second instruction tries to read the data, access control has not been finished. In this case, the data is allowed to be loaded and used. But the instruction in the RUU will be tagged with status specifying that it is using data that has not completed its access control checking, furthermore, the read access itself has not passed access control checking. Later, when the access monitor completes checking of the fetched data (whether the fetched data is allowed to be written to the cache line by the previous write), it will forward the result to the RUU and the status will be changed to the situation that only the read access itself has not passed access con-

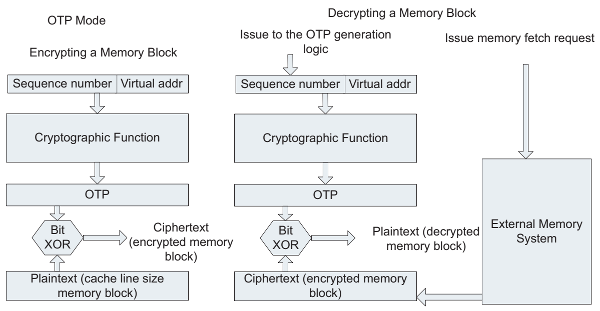

OTP Mode
Encrypting a Memory Block
Sequence number
Virtual addr
Cryptographic Function
OTP
Bit XOR
Ciphertext (encrypted memory block)
Plaintext (cache line size memory block)
Decrypting a Memory Block
Issue to the OTP generation logic
Issue memory fetch request
Sequence number
Virtual addr
Cryptographic Function
OTP
Bit XOR
Plaintext (decrypted memory block)
Ciphertext (encrypted memory block)
External Memory System

Figure 10: OTP

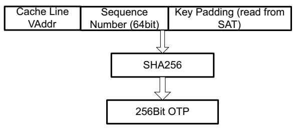


Figure 11: Generate OTP for a Cache Line Using SHA256

trol checking yet. When result of checking on the read itself (whether the program is allowed to read the cache line) is returned, the status will be marked as clean, meaning it can be retired after completion. Since access control checking takes only 2 cycles, assuming 1 cycle L1 access latency, in ideal case, checking result of the fetched data will be available right after the data is loaded into r2. One cycle later, result on checking the read itself will also be available.

3.3 One-Time-Pad (OTP) Prediction

One time pad based protection schemes have been proposed recently to address the latency overhead associated with confidentiality protection of program and data [22, 23][1]. Figure 10 shows the difference between regular block cipher based protection and OTP based protection. OTP based approaches reduces the overall data fetch latency by parallelizing OTP generation and data block fetch from the main memory. However, OTP based schemes are vulnerable to potential known plain-text attacks if the same seed is used repeatedly for the same cache line size memory block. To solve this problem, a sequence number is associated with each cache line size data memory block. The sequence number is initialized to a random secret value and encrypted when it is stored in the main memory. Each time, a dirty cache line is to be evicted from the processor cache, its associated sequence number is incremented, and the new sequence number is used to generate a OTP for encrypting the evicted data. Next time, when the data is fetched again into the processor, its associated sequence number has to be either cached inside the processor, or fetched from the memory, decrypted, and then used to generate the OTP for decryption.

OTP based schemes reduce latency only when the associated sequence number is cached on-chip. There are two conventional techniques to improve hit rate of OTP caching. First,

[1]As pointed in [1], all the proposed OTP based schemes so far are not true OTP in the sense OTP used by security community. They can be best described as OTP like.

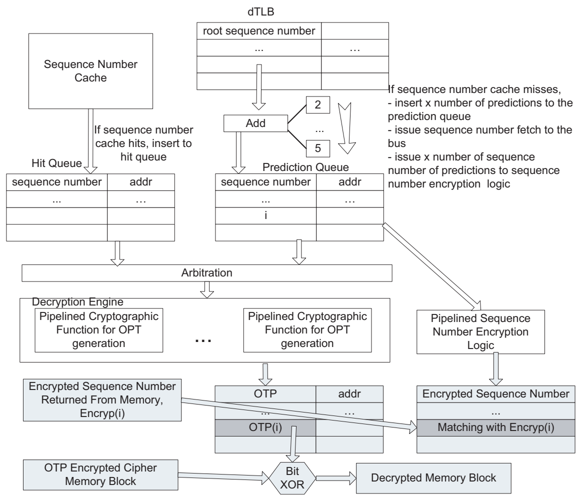

dTLB
root sequence number
...
...
Sequence Number Cache
If sequence number cache hits, insert to hit queue
Add
2
...
5
If sequence number cache misses,
- insert x number of predictions to the prediction queue
- issue sequence number fetch to the bus
- issue x number of sequence number of predictions to sequence number encryption logic
Hit Queue
sequence number
addr
...
...
Prediction Queue
sequence number
addr
...
...
i
Arbitration
Decryption Engine
Pipelined Cryptographic Function for OPT generation
...
Pipelined Cryptographic Function for OPT generation
Pipelined Sequence Number Encryption Logic
Encrypted Sequence Number Returned From Memory, Encryp(i)
OTP
addr
...
...
OTP(i)
Encrypted Sequence Number
...
Matching with Encryp(i)
OTP Encrypted Cipher Memory Block
Bit XOR
Decrypted Memory Block

Figure 12: OTP Prediction

a bigger or high associativity sequence number cache can be used with the cost of more chip area. Second, using sequence number pre-fetch to exploit the spacial locality of memory access. In this paper, we propose a third technique, sequence number prediction for reducing the overall latency of fetching encrypted data from memory. Sequence number prediction is based on the observation, 1) a pipelined decryption engine often stays idle when waiting for missing sequence number been fetched from the memory; 2) during the whole life a physical memory page is bound to a virtual memory page, many of its cache line size memory blocks are only updated very few times. Our profiling study on SPEC benchmarks indicate that many memory blocks are updated very few times during the whole process lifetime. The means that for a cache line that misses sequence number cache, its sequence number is very likely within a small range of the first time initialized random sequence number. The kinds of data that most likely to be updated rarely are constant values, constant strings, data structures that are initialized only when program starts. Sequence number of non-constant data may also exhibit predictability if the data is not dirty evicted from the L2 too frequently. To exploit this fact, we designed a OTP prediction technique shown in figure 12.

First, there is a root OTP sequence number assigned to each virtual memory page. This root OTP sequence number is initialized by a hardware random number generator each time the virtual page is mapped to a physical memory page. The root OTP sequence number is a secret not accessible by software. All memory blocks of the same page use the same root OTP sequence number as their initial value. Each time, a dirty block is evicted from the processor, its sequence number will be incremented. When a sequence number is evicted from the processor, it will be encrypted using AES encryption scheme [8]. Secondly, the decryption engine is pipelined and it stores decryption (OTP) requests in two queues, hit queue and prediction queue. When a missing cache block has its sequence found in the sequence number cache, its address and the sequence number will be inserted to the hit queue. When a missing cache block also misses the sequence number cache, the prediction logic will take the root OTP sequence number associated with the virtual page, and inserts a few sequence number guesses into the prediction queue. The hit queue has higher priority than the prediction queue. In each fetch cycle, the decryption engine will try to fetch the next request from the hit queue first. Only when the hit queue is empty, the engine will fetch from the prediction queue. As shown in the figure, the design supports potential multiple OTP generation pipelines. However, in this paper, we assume that there is only one available for decrypting fetched memory blocks.

There is a concern about OTP prediction performance over a large time window of execution. It is reasonable to suspect that prediction rate may drop as more data is updated. To address this issue, a dynamic prediction rate tracking and sequence number reset mechanism is proposed. The purpose of this mechanism is to find out virtual pages with low prediction rate caused by frequent memory updates and reset the page root sequence number to a new value so that high prediction rate can be restored.

Prediction tracking is performed in hardware using a scheme described as follows. There is a 16 bit prediction history value (PHV) associated with each virtual page. The PHV records hit or miss of the last 16 sequence number prediction on blocks of the associated page. Each time a data block is loaded from memory, the PHV of that page is shifted to left by 1 and ORed with the result of prediction represented by a single bit (e.g., 1 denotes miss and 0 denotes hit). When the total number of miss predictions of the last 16 predictions is greater than a threshold, the root sequence number associated with that page will be reset to a new randomly generated number. After reset, blocks of the involved page will use this new number for OTP generation next time when it is evicted from the L2. Figure 13 lists the operations of OTP prediction and sequence number reset in pseudo code.

```
Assume
  Baccess : address of memory block that misses on L2
  Brepl: address of memory block that will be replaced in L2
  Seq(addr) : current sequence number of a memory block
  PageSeq(addr) : root sequence number of a page
  OTP(addr) : one-time-pad associated with a block
  OTP_gen(addr, seq number, key) : crypto logic taking
   block addr and seq number for generating OTP
  seq_encrypt(addr, seq number) : crypto logic
   encrypting a sequence number for external store

When a memory block Baccess misses L2
  get key from SAT
 if seq number cache hit
   insert OTP generation request to the hit queue
 else
   seq_pred = PageSeq(Baccess);
   for (int i=0; i<prediction_depth; i++)
    insert OTP generation request to the prediction queue
    with seq number seq_pred+i, Baccess address, and key from SAT
    insert request of seq_encrypt(Baccess,seq_pred+i)
   endfor
   fetch sequence number Seq(Baccess) from memory
 endif
 fetch memory block

When encrypted sequence number Seq(Baccess) is returned
  match Seq(Baccess) with encrypted seq number predictions
  if a match is found
   update prediction history PHV with a hit
  else if Seq(Baccess) counts from PageSeq(Baccess)
   update prediction history PHV with a miss
  else // Seq(Baccess) starts from old PageSeq(Baccess)
   update the sequence number in memory so that it
    will count from PageSeq(Baccess) next time
   force Baccess in L2 as dirty so next time
    when it is replaced, it will be written back
    encrypted with OTP generated from PageSeq(Baccess)
  endif
  send OTP generation request if a match can not be found

When the encrypted memory block for Baccess is returned
  get decrypted data by OTP(Baccess) XOR encrypted data block

When Brepl is replaced in L2
  if dirty or Seq(Brepl) starts from old PageSeq(Brepl)
   if Seq(Brepl) does not count from the current PageSeq(Brepl)
    reset Seq(Brepl) to PageSeq(Brepl)
   endif
   Seq(Brepl)++;
   generate OTP(Brepl) using Seq(Brepl), Brepl address, and key from SAT
   encrypt evicted data by OTP(Brepl) XOR evicted data
  endif
```

Figure 13: OTP Prediction and Sequence Number Prediction

One basic function required by the pseudo-code is the ability to test whether a sequence number used by a memory block starts counting from the current root sequence number. Note that this function does not have to be 100% accurate because a wrong test result will only cause reset of the memory block's sequence number. A simple implementation is, after a sequence number is incremented, it is Bit XORed with the current root sequence number and a simple CRC is calculated using the result of XOR. This simple CRC is stored in pair together with the sequence number. To decide whether a sequence number starts counting from the current root sequence number, the sequence number is XORed with the current root, and a CRC is generated and matched with the one stored in memory. If they do not match, the sequence number is considered starting count from an old root sequence number. This checking is done every time before a sequence is to be incremented. If a mismatch is detected, the block will reset its sequence number to the current root sequence number.

It is important to point out that predictability of the sequence number does not mean that it is not secure. The sequence number is predictable by the secure processor does not suggest that it can also be predicted by hackers. The root sequence number for each page is randomly initialized. Each time when the sequence number associated with a memory block (including the root sequence number itself) is stored to memory, it is encrypted using AES encryption standard [8]. Furthermore, even a hacker knows the OTP of a particular memory block, it does not tell any information about the sequence number or OTPs of other blocks on the same page. The security requirement that each time an evicted L2 block is encrypted using a different non-predictable OTP is maintained.

3.4 Integrity Protection Under MESA

The existing integrity protection schemes for secure processor architectures are based on the construction of an m-ary hash/MACtree. Under the memory centric mode in which information within a process space is usually encrypted by multiple encryption keys,the original schemes of hash tree or MAC (message authentication code) tree based memory authentication cannot be directly applied [9]. Consequently, we generalized the integrity protection tree structure. We first protect individual memory module with their own hash/MAC tree, and then, a new hash/MAC tree is constructed on top of it, as shown in figure 14. In figure 14, a leaf node represents an individual integrity code and each internal node denotes a MAC of all the children nodes. All the nodes of integrity trees are securely stored in the ex-

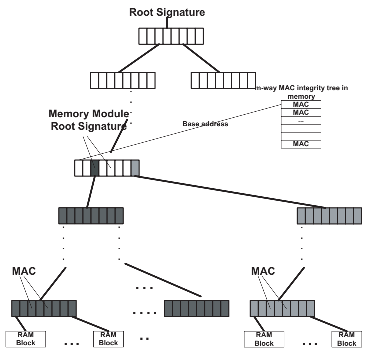


Figure 14: Layered MAC Tree for Memory Integrity Verification

ternal RAM. To speed up integrity verification, frequently accessed nodes of the MAC tree are cached on-chip. When a new cache line size block of data/instructions is fetched, the secure processor verifies its integrity by inserting it into the MAC tree. Starting from the bottom of the tree, recursively, a new MAC is computed and compared with the internal MAC tree node. The MAC tree is updated when a dirty cache line is evicted from the secure processor. The secure processor can automatically determine the memory locations of MAC tree nodes and fetch them automatically during integrity check if they are needed. Root of the MAC tree is preserved securely when a process is swapped out of the processor pipeline. MESA uses *authentication speculative execution* [15, 1, 20]. In *authentication speculative execution*, decrypted instructions are allowed to be issued and executed before the result of integrity verification is determined. Similar to *access control speculative execution*, instructions using data that has not passed integrity check can not be retired. In [20], a detailed scheme of *authentication speculative execution* is described.

4. PERFORMANCE EVALUATION

The purpose of performance evaluation is to show that access control does not incur significantly more performance overhead over protection without it. Furthermore, we want to study the potential performance advantages of OTP prediction. A cycle-accurate processor architecture is used for detailed performance evaluation.

4.1 Simulation Framework

Our simulation framework is based on SimpleScalar [5] running SPEC2000 integer and FP benchmarks compiled with -O3 option. We implemented architecture support for access control and OTP prediction over SimpleScalar's out of order Alpha processor simulator. We also integrated a more accurate DRAM model [11] to improve the system memory modelling, in which back conflicts, page miss, row miss, page miss are all modelled following the PC SDRAM specification. The architectural parameters used for performance evaluation are listed in table 1. We used pipelined OTP decryption and encryption engines. SHA-256 [18] has 64 rounds and each round is performed in one pipeline stage. With 1.0 GHz processor core clock rate, each round can be completed in two core clock cycles, giving a total SHA-256 based OTP generation latency of 128 cycles or 128ns under 1.0 GHz core clock speed. Under this pipelined design, a new decryption request can be issued every two clock cycles. To model OTP prediction faithfully, we added a memory profiling support to SimpleScalar that keeps track important memory information for evaluating OTP prediction, such as number of times a memory block is evicted from L2 cache, sequence number allocated to each virtual page, and etc. Each benchmark is fast-forwarded according to SimPoint's suggestion [19] then simulated for 400M instructions in performance mode. During fast-forwarding, L1 cache, L2cache , sequence number cache prediction mechanism are also simulated. The profiled memory status is also update during fast-forwarding. To study sensitivity of OTP prediction to execution time, we also run each benchmark in a simplified mode that simulates the memory hierarchy and OTP prediction using 8 billion instructions. Other OTP prediction related parameters are prediction history window, which is 16 bits. By default, the sequence number of each virtual page is reset if the number of prediction misses over the last 16 is greater than or equal to 12. Prediction range is the number of guesses generated for each missing sequence number.

Parameters	Values
Fetch/Decode width	8
Issue/Commit width	8
L1 I-Cache	DM, 8KB, 32B line
L1 D-Cache	DM, 8KB, 32B line
L2 Cache	4way, Unified, 32B line, write back cache 256KB and 2M
L1/L2 Latency	1 cycle / 6 cycles (256KB), 12 cycles (2M)
I-TLB	4-way, 256 entries
D-TLB	4-way, 256 entries
Memory Bus	200MHz, 8B wide
SHA-256 latency	64 stages, 2 cycle each, 128ns total
AES latency	120ns
Access Control Monitoring latency	2 cycle
Sequence number cache size	4KB, 8KB, 32KB
Prediction history window	16bit
Prediction range	4

Table 1: Processor model parameters

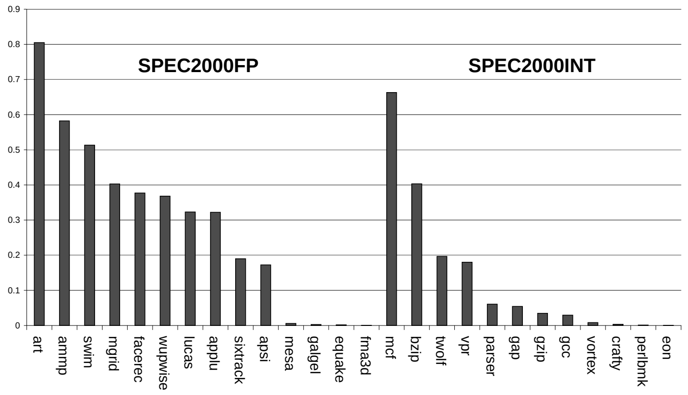


Figure 15: L2 miss rates for SPEC 2000 (256KB)

Both SPEC2000 INT and FP benchmarks were used for our evaluation. We subset the simulations for those with high L2 misses[2] as indicated in figure 15. All the benchmarks are simulated under the security setting that data confidentiality has to be protected.

4.2 Performance Analysis

[2]The reason we did not choose facerec and lucas was because their absolute numbers of misses are low despite of their high miss rates.

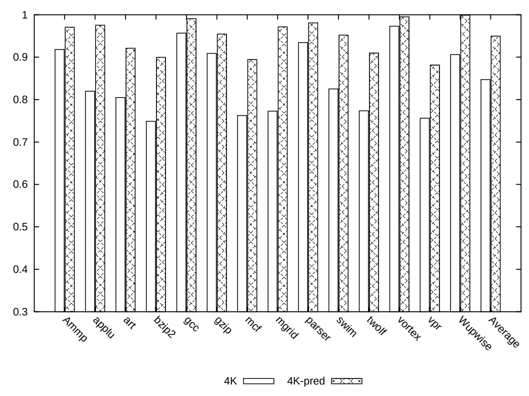


Figure 16: Normalized Performance of Sequence Number Prediction plus Sequence Number Cache vs. Sequence Number Cache Only

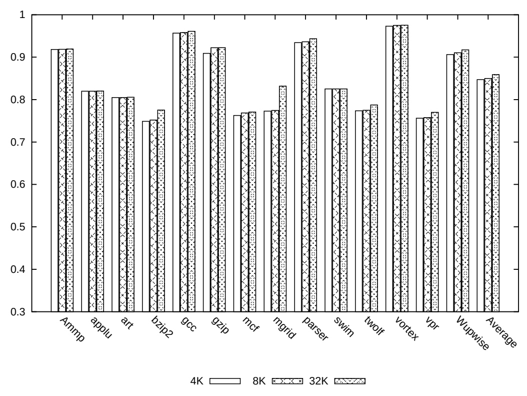


Figure 17: Normalized Performance of 4K,8K, and 32K Sequence Number Cache without Sequence Number Prediction

4.2.1 *Access control*

We simulated all the selected benchmarks with access control enabled. The access control monitor checks every data read/write from and to L1 cache and broadcasts the result after the checking is completed. Instructions reading L1 cache line with pending access control result are allowed to be issued and executed but can be only retired after the result is obtained. Using the proposed access control mechanism, our simulation result does not show performance changes for all the studied benchmarks. Two main possible reasons why the proposed access control does not impact on performance are, 1) it is conducted in parallel with L1 access and taking only very small number of cycles; 2) instructions are allowed to access and use data with pending access control result. Since access control checking is very fast and in parallel with L1 access, access control result is almost certainly ready when the instruction is completed and ready to be retired because there is only one more cycle delay on access control result of L1 read/write. This extra cycle can be tolerated by the access control speculative execution because instructions using either stored data from unchecked store or fetched data from unchecked read can be issued and executed at the same time. If execution of instructions using unchecked data takes at least one cycle to finish, which is certainly true, when the instruction is completed, access control checking result is also ready. Note that access control involves only L1 cache and its performance is not sensitive to cache sizes, L2 performance, and etc.

4.2.2 *Performance improvement using OTP prediction*

Performance results under small L2 cache is critical because protection on data confidentiality is not only deemed for very high end machines but commodity platforms as well. Majority sold processors for regular users have L2 cache of only 256K or even less.

Figure 16 shows normalized IPC performance results of 4K sequence number cache vs. 4K sequence number cache with sequence number prediction. The IPC is normalized to a baseline of ideal situation that there is no miss of sequence number. This represents a perfect scenario that OTP generation for every fetched memory block can be parallelized with memory access. As indicated by the results, OTP prediction improves performance for almost every simulated benchmark. The average improvement is about 11%. But for certain benchmarks, such as bzip2, mgrid, twolf, vpr, the improvement is in the range of 15-25%. The improvement is largely due to the increase of sequence number hit for fetching missed memory blocks.

To show that sequence number prediction improves performance even for relatively large sequence number caches, we did experiment using 8K and 32K sequence number caches and compare the results in figure 17. The results show that although increasing sequence number cache size can improve performance slightly, but the improvement is not proportional to the increase of sequence number cache size. One possible explanation is that for sequence number cache to perform well, is that the processor has to miss on the same memory block many times within a short time window before the sequence number is evicted from the sequence number cache. Due to the temporal locality and memory working set, a processor rarely repeats missing on the same memory block many times in a short time. This suggests that sequence number may have a very large secondary working set.

The slow improvement of sequence number hit rate under increase of sequence number cache suggests that a combination of sequence number prediction and a small sequence number cache is best for reducing sequence number misses. A small sequence number cache with relatively large cache line size can be used to capture spacial locality of L2 misses and repeated conflict misses, while sequence number prediction can be used to reduce misses on data blocks that are not frequently updated.

One parameter used for OTP prediction is prediction range, number of guesses inserted to the prediction queue of OTP generation engine. To study the effect of prediction range, we experimented 3 settings using 4 guesses, 7 guesses, and 10 guesses respectively. The result is shown in figure 18

The results show little difference over the three experimented settings. There is no significant trend of performance improvement using large number of guesses. In fact, per-

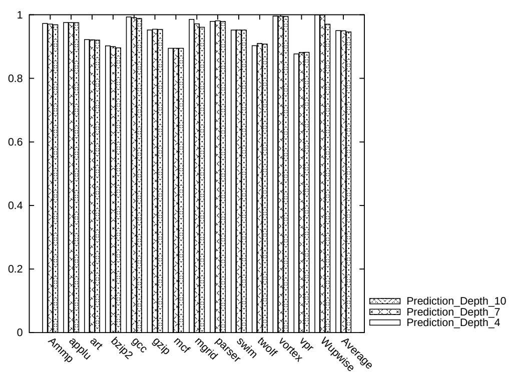


Figure 18: Normalized Performance Under 4, 7, and 10 Guesses For Each Missing Sequence Number

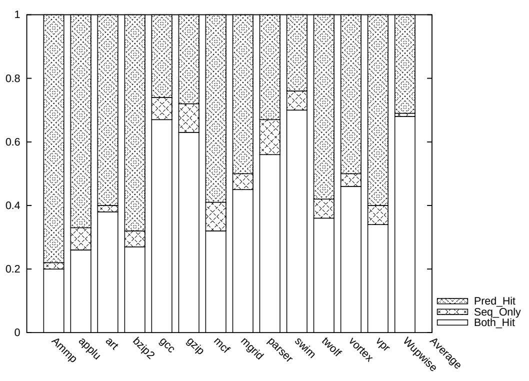


Figure 20: Breakdown of Contribution of Sequence Number Cache, and Sequence Number Prediction

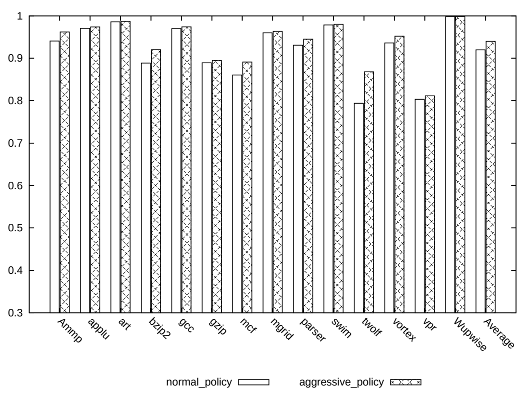


Figure 19: Overall OTP Hit Rate Across 8 Billion Instructions

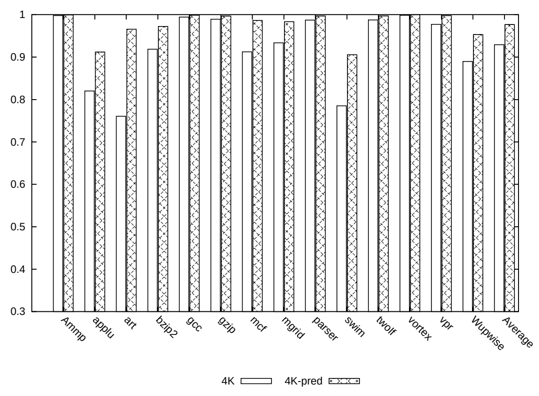


Figure 21: Normalized Performance of Sequence Number Prediction plus Sequence Number Cache vs. Sequence Number Cache Only, 2M L2

formance of some benchmarks decreases slightly when too many guesses are generated. One explanation of why large number of guesses does not lead to improved performance is that it can generate overwhelming number of predictions and cause great resource contention on the OTP generation engine.

4.2.3 *OTP prediction over large execution time*

One concern about OTP prediction is that its performance may decrease over execution time. To answer this question, we simulated performance of OTP prediction over relatively large time window, 8 billion instructions under 256K L2. Since we suspected that OTP prediction performance over a large period may be affected by the sequence number reset threshold, we experimented two settings of reset threshold, called normal setting and aggressive setting. Under normal policy, sequence number associated with a virtual page is reset if twelve of the last sixteen predictions meet sequence number outside the prediction range. Under aggressive setting, threshold is set to eight instead of twelve.

Figure 19 shows the overall OTP hit rate for the two studied settings. First, the results indicate that OTP hit rate does not decrease over time. The overall hit rate is relatively high even after 8 billions of instructions are simulated. Figure 20 breaks the total number of hits under normal policy into three categories, 1)hit both, a sequence number that is in the sequence number cache and can be predicted; 2) prediction only, a sequence number that is missing in the sequence number cache, but can be predicted; 3) sequence cache only, sequence number can not be predicted but available in the cache. The result suggests that sequence number prediction makes a great contribution to improving the overall sequence number hit rate. Secondly, the results show that aggressive setting only improves the overall OTP hit rate slightly.

4.2.4 *Performance improvement on 2M L2*

We also evaluated sequence number prediction under large L2 cache size. Note that sequence number prediction is a technique for reducing latency of fetching OTP encrypted data from memory. If the application does not require a large memory throughput due to large size of L2, OTP prediction will not be expected to improve performance significantly. Many of the memory hungry SPEC2000 benchmarks (e.g., bzip2, mgrid, and mcf) under 256K L2 are no longer so under 2M L2 size because their entire working can be fit into a large 2M L2.

Figure 21 shows normalized performance results using 2M L2 cache under 4K sequence number cache and sequence number prediction. The performance is normalized to the baseline of ideal situation that there is no miss on sequence number. The averaged performance improvement is about

Table 2: Tamper Resistant/Copy Protection System Comparison

System	Confidentiality	Cipher	Access Control
XOM [15]	whole process based	triple-DES	NA
Aegis [9]	whole process based	AES	NA
MESA	fine grain	OTP, OTP prediction	yes
Yang, Zhang, and Gao [23]	whole process based	DES based OTP	NA
Suh, etal [22]	whole process based	AES based OTP	NA
Zhang, Gupta [24]	instruction slices based	block cipher	NA

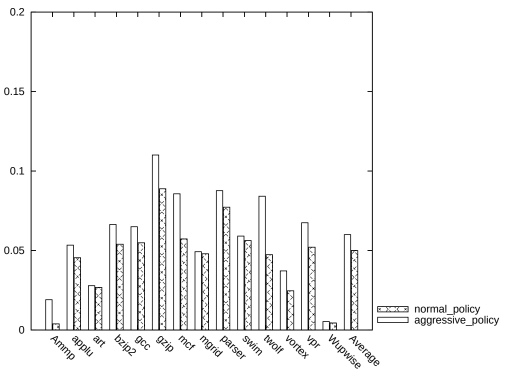


Figure 22: Percentage of Extra L2 Blocks That Have to Be Evicted Due to Sequence Number Reset, 256K L2

5%. For some benchmarks, it is about 10%. The performance improvement is less than the situation of 256K L1 cache. One reason is that 2M is big enough to hold many benchmarks' working set, thus leave a small margin for improvement. Under 4K sequence number without prediction, the averaged IPC is almost 90% of the ideal scenario that assumes no miss of sequence number.

4.2.5 Memory Throughput overhead of OTP prediction

Resetting the sequence number associated with each virtual page as the prediction rate goes down can improve future prediction on frequently evicted memory blocks, but may cause more demand on memory throughput because a non-dirty L2 block of the same page has to be evicted if it is replaced by other memory block. Figure 22 shows the percentage of extra L2 blocks that have to be evicted due to resetting of sequence numbers under the two reset threshold settings (8 billion instructions). As shown by the result, the overhead is relatively small. Under the normal threshold setting, it is below 5%.

5. RELATED WORK

Software protection and trusted computing are among the most important issues in the area of computer security, which have received extensive studies for years. Traditionally, the protections on software are provided through trusted operating systems. The operating systems implement certain mechanisms to ensure that the applications are cryptographically protected and the information spaces of different applications are isolated from one another. Consequently, the software protections are achieved since malicious applications will not be able to access to others without the permission of OS. The trusted computing is ensured as well since an application is protect from tampering from others by the underlying operating system. To improve the security model, some tamper resistant devices are embedded into computer architecture to ensure the loaded operating system is trusted. From there, a chain of trust applications are run, each depending on the underlying layer. A typical example of such computer architecture is the TCPA [2] architecture and the related operating system is known as NGSCB [4]. Another example based on the concept of virtual machine is Terra [10]. Although these systems provide authentication services and prevent simple tamper on the application, they are not designed for protection on software confidentiality. They may be used for digital right protection but it is unlikely that they can prevent physical based attack on copy protection and software confidentiality.

Nevertheless, as history indicated, to the issue of copy protection and software confidentiality, computer hackers/crackers tend to be well-knowledge, highly motivated, and sometimes well supported financially to crack a protected machine and its software. In many cases, the computing devices themselves can be in the hands of adversaries. Consequently, they can launch physical attacks which are even serious than the software based attacks prevented by the previous approaches. The instance of XBOX security key breaking is one such example [13]. This imposes additional challenges and traditional software protection approaches seem not sufficient to address the problem. As a result, new processor architecture, e.g. XOM and Aegis, has emerged, which protects trusted computing with architectural support. With the architectural support, an operating system, called XOMOS [16], is proposed that is able to not only protected applications, but also provide most necessary services available in traditional OS. Another work by Shi et al. [20] proposes a technique for protection of memory integrity in Multiprocessor Systems. Table 2 compares some recently published systems designed for hardware based protection on software confidentiality.

Note that though traditional capability based systems such as Hydra and CAP have mechanisms to manage access on information, they are not the kind of tamper resistant systems that can prevent physical attacks on software integrity and confidentiality. The architecture and the associated OS we proposed is able to address high performance fine-grain protection on software confidentiality with the proposed access control and OTP prediction technique.

6. CONCLUSIONS

This paper describes a high performance memory centric security system that protects software confidentiality using a novel one-time-pad (OTP) prediction technique. Different from the previous process-based tamper-resistant systems designed for the same purpose, the new system allows different software components to be selectively protected or separately protected based on different security requirements, therefore more flexible than the previous process based protection. It supports selective protection on software components in an un-trusted and opened software environment. The novel hardware based access control enforces security protection on memory modules with heterogeneous security requirements. Implementation of memory centric protection requires only small amount of additional hardware resources over process-centric based approach and causes no more performance degradation than process-centric based protection because of proposed security speculative execution and parallel of access control with cache access. The proposed novel OTP prediction technique significantly improves benchmark performance over the previous OTP based designs using sequence number cache. For some memory hungry benchmarks, the gain is in the range of 15% to 25%. The OTP prediction technique when combined with a small 4KB sequence number cache outperforms setting of large 32KB sequence number cache on average by 11%. The prediction technique generates OTP guesses using free idle pipeline cycles of OTP generation logic thus requires little extra hardware resources.

7. REFERENCES

[1] M-TREE: A Fast Secure Architecture for Protecting the Integrity and Privacy of Software. *Submitted for publication.http: //www.cc.gatech.edu/people/home/lulu/Mtree.pdf*, 2004.

[2] The Trusted Computing Platform Alliance. http://www.trustedpc.com. 2003.

[3] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, page 65. IEEE Computer Society, 1997.

[4] Next-Generation Secure Computing Base. http://www.microsoft.com/resources/ngscb/default.mspx.

[5] D. Burger and T.M. Austin. The simplescalar toolset, version 2.0. Technical Report 1342, University of Wisconsin, June 1997.

[6] Carlisle Adams, Steve Lloyd, and Stephen Kent. *Understanding the Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations.* . New Riders Publishing, 1999.

[7] Ellis Cohen and David Jefferson. Protection in the hydra operating system. In *Proceedings of the fifth ACM symposium on Operating systems principles*, pages 141–160. ACM Press, 1975.

[8] Federal Information Processing Standard Draft. Advanced encryption standard (aes). national institute of standards and technology, 2001.

[9] E.G.Suh, D.Clarke, M.van Dijk, B. Gassend, and S.Devadas. Aegis: Architecture for tamper-evident and tamper-resistant processing. In *Proceedings of The Int'l Conference on Supercomputing, 2003*, 2003.

[10] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 193–206. ACM Press, 2003.

[11] Matthias Gries and Andreas Romer. Performance evaluation of recent dram architectures for embedded syhstems. *TIK Report Nr. 82, Computer Engineering and Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich*, 1999.

[12] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 public key infrastructure certificate and CRL profile. RFC 2459, Internet Engineering Task Force, January 1999.

[13] A. Huang. Keeping secrets in hardware the microsoft xbox case study. *MIT AI Memo*, 2002.

[14] Butler Lampson, Martin Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: theory and practice. *ACM Trans. Comput. Syst.*, 10(4):265–310, 1992.

[15] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. Architectual support for copy and tamper resistant software. In *Proceedings of the 9th Symposium on Architectural Support for Programming Languages and Operating Systems*, 2000.

[16] David Lie, Chandramohan A. Thekkath, and Mark Horowitz. Implementing an untrusted operating system on trusted hardware. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 178–192. ACM Press, October, 2003.

[17] R. M. Needham and R. D.H. Walker. The cambridge cap computer and its protection system. In *Proceedings of the sixth ACM symposium on Operating systems principles*, pages 1–10. ACM Press, 1977.

[18] National Institute of Science and Technology. Fips pub 180-2: Sha256 hashing algorithm.

[19] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, October 2002.

[20] Weidong Shi, Hsien-Hsin S. Lee, Mrinmoy Ghosh, and Chenghuai Lu. Architectural support for high speed protection of memory integrity and confidentiality in symmetric multiprocessor systems. In *Proceedings of the 2004 International Conference on Parallel Architectures and Compilation Techniques*, 2004.

[21] E. Suh, B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas. Caches and merkle trees for efficient memory authentication. In *Proceedings of the Ninth Annual Symposium on High Performance Computer Architecture*, February 2003.

[22] E. G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. Efficient Memory Integrity Verification and Encryption for Secure Processors. In *Proceedings 0f the 36th Annual International Symposium on Microarchitecture*, December, 2003.

[23] Jun Yang, Youtao Zhang, and Lan Gao. Fast secure processor for inhibiting software piracty and tampering. In *36th Annual IEEE/ACM International Symposium on Microarchitecture*, December, 2003.

[24] Xiangyu Zhang and Rajiv Gupta. Hiding program slices for software security. In *Proceedings of the 2003 Internal Conference on Code Genration and Optimization*, pages 325–336, 2003.