

Architecture Information Assurance

Brian J. d'Auriol

Department of Computer Engineering, Kyung Hee University, Korea
Ohio Supercomputer Center, Columbus, Ohio, USA
dauriol@acm.org

ABSTRACT

A preliminary model is introduced in this paper whereby data and its associated security properties are treated as a single atomic unit of information in a hardware-only context. Security-tagged data allows each datum to be properly manipulated with a predictable assurance. This paper addresses some of the issues and overhead due to the data size increase of tagged data. This work is a part of a larger issue that instruction set architectures (ISAs) do not consider the information assurance implications in its operational environment.

1 INTRODUCTION

Information assurance typically involves the protection of information. Three common aspects are confidentiality, integrity and availability. Confidentiality deals with the appropriateness of how the information is accessed. Integrity deals with the semantic correctness of the information. Availability ensures that the information is accessible at the time it is needed. Many computer security protocols typically applied in a layering fashion are used to ensure protection of these aspects of the information.

Security in computer architecture has tended to either be management protective as in the case of privileged (kernel) modes of operation or globally protective as in the case of FIPS 140 [1] compliant architectures. The former is a software-hardware solution whereby an operating system manages the protection by utilizing specialized architecture features. The latter represents a hardware-only approach to security in architectures. Hardware-only approaches tend to

be more recent and are motivated by both a concern to better protect encryption/decryption communication processes and by a feeling that the software in the dual-nature solution is subject to compromise and hence protection failures.

The main idea behind the work in this paper is that data and its associated security properties are treated as a single atomic unit of information in a hardware-only context. Security-tagged data allows each datum to be properly manipulated with a predictable assurance in the context of confidentiality, integrity and availability. However, typical architectures are not designed to accommodate the processing of atomic security tagged data nor its memory addressing, data transfer or interprocessor communications. Although tagged data may be logically associated and processed under software control, experience from the hardware-software approaches indicates that doing so is susceptible to potential compromise and protection failures. Applications of security data tagging include screening-out sensitive information from display or removable devices (thereby perhaps preventing similar occurrences of the recent data breach at the U.S. Department of Veterans Affairs [2]), and better control of security over data manipulated by an architecture.

This paper addresses some of the issues and overhead surrounding security property tagging of data. Confidentiality and integrity are considered. A preliminary model is introduced and assessed by two case studies, one applied in a simple MIPS processor design and the other applied in an optical interconnected multiple processor architecture model. This work is a part of a larger issue that instruction set architectures (ISAs) do not consider the information assurance implications in its operational environment.

The remainder of this paper is organized as follows. The next section, Section 2, reviews related work. Section 3 introduces a preliminary model of security property data tagging and discusses implementation and performance implications of the model. Conclusions are given in Section 4.

2 REVIEW

Protection of hardware resources has a long history. Multi-tasking and the more recent multi threading operating systems provide management of resources by using privileged, kernel or supervisory modes, for example, BSD [3]. Operating systems may also provide protection over shared memory especially where shared memory is used to support interpro-

cessor communication (IPC) between processes [3]. Such management is made possible by hardware supported features such as kernel accessible memory and kernel-only registers of the MIPS 32 and 34K core [4]. Multiple operating system interfaces can also be supported in a secured way, for example, the privileged architecture library code of the Alpha AXP 21064 [5]. This dual hardware-software approach provides security over process and thread contexts, usually, so that user applications are isolated from each other or can not otherwise interfere with the integrity of the system. However, the well known (and often exploited) limitation of this approach is the security of the operating system software itself. Compromised software leads to the break-down of system security.

Another approach is that of physical hardware security combined with internal hardware security. A FIPS 140 compliant hardware module [1] provides up to four layers of protection. Specified security features include a finite state model describing correct operations, secure key management and, for Level 4, a detection and response “envelope of protection”. The IBM 4758 processors are certified under Levels 3 and 4 of the FIPS 140-1 (the previous version of the standard, 1994) [6, 7]. Some of the features implemented in the IBM 4758 are the zeroization of some of the RAM memory, processor subsystem reset, termination of RAM-memory refresh, and a state controller which, amongst other actions, control memory allocation. Detected penetrations result in sensitive data destruction.

Hardware level protection is also noticed elsewhere, although, to a much lesser extent. For multiple processor systems, memory management may cache private data in such a way that restricts access to it by the other processors [8]. This provides limited information access as a by-product. The MIPS 34K core also provides the `prefPrepareForStore` variation of the cache prefetch that can zero out a cache line in certain circumstances [4].

The Cell Broadband Engine (Cell BE) [9] is a recent processor especially designed with hardware-only supported security in mind. The processor has nine cores configured as one master and eight slaves. The slaves may be executed in a “secure processing vault” mode where instructions are first downloaded into the slave’s local memory and then the slave core is disengaged from the rest of the chip. Typical operations is when encrypted data is sent to the slave, decrypted, processed in the secure vault mode, re-encrypted and then sent out of the slave. The design also involves runtime verification of an instruction stream complemented by hardware key support.

The hardware only approach addresses the limitation of the dual software-hardware approaches. However, the hardware only approaches surveyed here consider more globally protective mechanisms. In particular, these approaches do not consider data tagged with security attributes specifying allowable operations. Such a model is presented next.

3 ARCHITECTURE

Architectures provide three important functions over data: storage, communication and synthesis. There are many aspects of data storage, but most can be described in terms of register, cache, memory and secondary storage. Com-

munication in a processor-memory context is usually supported by an address bus for memory location information and a data bus for data value transfer. Communication in a processor-processor context may involve the use of dedicated interprocessor links or via an external network using a network interface. Processor-processor communication is often serialized onto a single bit stream. Data synthesis involves the processes of generating new information from given inputs, for example the Algorithmic Logic Unit (ALU) is a primary data synthesis device.

Several simplifying assumptions are made. Data is only obtained from dedicated automated sensor networks and communicated using hardware-only mechanisms directly into storage. This implies that the stored data can be tagged at the point of its synthesis under hardware-only protocols. These assumptions eliminate the possibility of software controlled access to the data prior to the data manipulation in a processor.

Assurance as intended here is stated as follows: *The degree of confidentiality or integrity of data after an operation is no better than that of the operation’s inputs.* In keeping with the appropriateness of information access, the degree of confidentiality ranges from zero indicating public information to larger integer values indicating successively more limited access to the information. In keeping with the semantic correctness of the information, the degree of integrity ranges from zero indicating a lack of correctness to larger integer values indicating successively increased semantic correctness. Operations include storage, data synthesis (e.g. operations implemented in an ALU), and communication (i.e. processor-memory and processor-processor interactions). Let $c \geq 1$ define the bit field representing the category or degree of confidentiality of the datum therefore there are 2^c possible categories; similarly, let $i \geq 1$ define the bit field representing integrity. The confidentiality output from an operation is defined to be the maximum confidentiality over all of its inputs, that is, the result of some operation should have a degree of confidentiality corresponding to the most limiting degree of its inputs. The integrity output from an operation is defined to be the minimum integrity over all of its inputs, that is, the result of some operation should have a degree of integrity corresponding to the least semantic correctness of its inputs.

The size of the tag field is expected to be small. The remainder of this paper assumes $i = c = 4$ bits. A byte register stores c and i respectively in bits 0-3 and 4-7. A comparator combined with a multiplexer can be used to select the lowest/highest tag value. For example (Figure 1), if unsigned integer representation is used, then two separate 4-bit magnitude comparators such as [10] can be used, and, the combined 7-10ns delay time of these devices (which may have less delay time if larger scale integration is considered) could be over-lapped with ALU or other operations performed in parallel.

There are three possibilities of associating the c and i tags with a datum in memory.

1. Extend word size: The tags are stored in a second memory location and accessed atomically with the datum using two bus cycles, one for the datum and one for its tag. Advantages include: use native memory capability, does not affect the normal data sizes, does not require

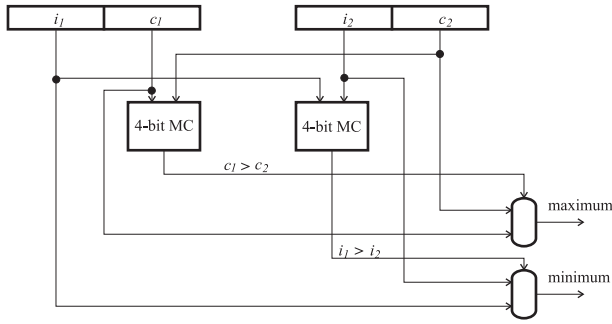


Figure 1: Comparator logic to implement the assurance model.

contiguous memory implementations thereby allowing multiple tags to be packed into words. Disadvantages include: the atomic load/store operations increase bus utilization and access time thereby causing probably disruptions to data throughput, clocked/control-signal control over load/store tag cycle results in additional control logic, and, if tag packing is done, address/offset calculation is needed.

- Utilize same word size: The tags are stored in the high-order bits of the data word. Advantages include: use native memory capability, does not require load/store tag cycle nor atomic operations. Disadvantages include: decreased usable data size, requires contiguous memory implementations, does not provide for tag packing. For 32-bit data architectures, the loss of data bits is not desirable. However, as longer data sizes become supported, architecture design could be enhanced to provide support for this option.
- Utilize additional memory interconnects. The tags are stored in a second memory location and accessed atomically with the datum in a single bus cycle. Advantages include: does not affect the normal data sizes, does not require contiguous memory, does not incur additional bus utilization and access times. Disadvantages include: does not use native memory capability which results in greater wiring and bus control requirements. The additional bus lines could therefore accommodate the transfer without undue burden.

The tags need to be appropriately processed in a processor regardless of the storage and process-memory communication. For this section of the paper, the MIPS architecture [4,11,12] is selected. This paper concentrates on the typical ALU operations for this architecture. The ISA defines a register file consisting of 32 general purpose registers. There are three instruction types. Of these, the R-type instructions generally specify that registers are the ALU operands whereas the I-type instructions generally specify immediate data as one of the ALU operands. The load/store architecture of the MIPS guarantees that memory loads into the register file maintain the c and i values of the respective data items. However, the immediate data is not so tagged. In the absence, it may be considered to have the lowest integrity value akin to claiming its integrity is unknown and

the lowest confidentiality value akin to claiming its fully public nature. The ‘and link’ variations (e.g. the jump and link JAL instruction) result in the program counter (PC) stored in Register 31. A similar absence tagging is assigned to the PC for these instructions. Higher degrees of tags can be assigned to immediate data and the ‘and link’ data if the instruction stream is state machine verified.

A brief example is given. Let a MIPS instruction sequence to carry out the operation of $2(a + b)$ be as follows:

```
add $t0, $s1, $s2
add $s0, $t0, $t0
```

where \$s1 and \$s2 are the registers storing a and b respectively and register \$s0 contains the result of the expression. For simplicity in this example only, assume that $c = i = 1$ and therefore confidentiality and integrity reduce to binary states. Let us assume that a is public but b is private information and furthermore that both a and b are semantically correct. Then, the security tags associated with \$t0 indicate private and correct information which is propagated to the result \$s0.

MIPS makes a distinction between its architecture and an implementation of its architecture, including, required and optional compliance. An academically widely publicized implementation based on dedicated wiring is discussed in [13]. An academic alternative bus-based implementation is discussed in [14].

The dedicated wiring implementation, illustrated in Figure 2, makes distinct the separation of the immediate data available in the instruction stream from data obtained from memory. The register file is the sole input to the ALU’s upper port. However, I-type instructions designate the immediate data as the input to the ALU’s lower port whereas R-type instructions designate the register file as the input. The ‘and link’ data path is not explicitly shown.

The dedicated wiring implementation is enhanced as shown in Figure 2 to include the security tags. There is a separate thirty-three 1-byte register file with thirty-two registers associated with its corresponding general purpose register, and one register corresponding with the immediate data. The comparator logic from the previous discussion is included. For R-type instructions, the two sub-systems act independently with the Read Register ports ‘a’ and ‘b’ (RRa and RRb in the figure) decoding the general purpose and security tag registers. The Tag Comparator determines the maximum confidentiality and minimum integrity and multiplexes these results to the Write Register data port. A similar independent sub-system operation occurs for general I-type instructions that use the ALU for computational purposes. Here, the control logic determines this sub-class of I-type instructions (from the instruction’s op-code) and selects the ‘Immediate’ tag register by way of the appropriate control signal to the multiplexer. The ‘load word’ (lw) instruction obtains the 40 bit data plus tag information from the data cache and writes back the respective bit fields to the appropriate register (the Tag Comparator may or may not operate on its inputs, however, it has a null-effect due to the multiplexer selector). The ‘store word’ (sw) instruction combines the 8-bit tag and 32-bit data fields presenting the 40-bit field to the data input of the cache. The cache and the Memory Management Unit may implement its storage and transfer algorithms based on the earlier analysis. Although

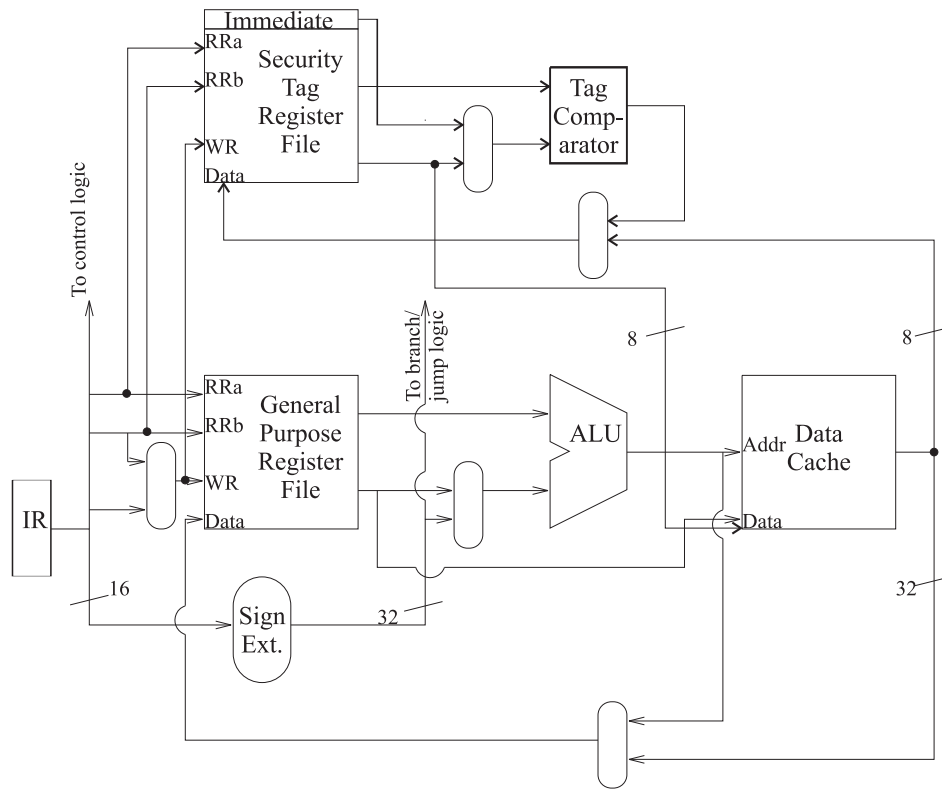


Figure 2: Enhanced simplified dedicated wiring implementation showing the integration of the security tag model implementation.

this paper considers the simplest version of the architecture implementation, the mirror-like operational characteristics of the security tag sub-system together with the lesser delays incurred by this sub-system suggests its candidacy for more realistic pipelining implementations.

The bus-based implementation makes the immediate value available on the bus and therefore available to either of the ALU inputs, to the register file or to memory. Compliance with the MIPS architecture, especially the ISA, will ensure that the MIPS semantics of the instructions are maintained. Again due to the mirror-like characteristics, the bus-based version may also be extended to accommodate the security tag model. However, should a super-set ISA be implemented on this implementation, say, involving a direct immediate data to memory copy, then the security tag model will likewise require enhancing to ensure correct propagation of the tag information.

Processor-processor communication is typically serialized and multiplexed onto a single bit stream. For this part of the study, the LARPBS(p) [15] and LARPBS [16] optical bus parallel computing models are selected due to the author's future interest in integrating the security tag model into these architectures. Multiple processors are interconnected with an optical fiber waveguide with a typical index of refraction of 1.469. Typical data communication rates are on the order of 10Gbps. Due to the addressing and routing requirements of the optical bus models, the entire message must be buffered in the fiber between connecting processors. The security tags add an additional 25% overhead; 8

bits of information to the 32 bits of data communication. For a 10Gbps data rate, 32 data bits require 0.65m of inter-processor fiber requiring 3.2ns; the additional 8 tag bits require an additional 0.163m and 0.8ns. However, again due to the addressing and routing requirements, a worst case communication requires on the order of $2N$ interprocessor links to complete, where N denotes the number of processors. For a 10 processor system, the overheads accumulate to 3.29m and 16.1ns, however for a desired 1000 processor system, the overheads accumulate to 346.75m and 1.7ms. Therefore, as the system approaches desired capabilities, the overheads incurred in the serialized security tag communications become a matter of concern.

4 CONCLUSION

The proposed security tag model is successfully integrated into a simple architecture implementation in this paper. The discussions included implementation aspects related to the storage requirements, processor operations and processor-memory as well as processor-processor communications. This now provides a hardware-only mechanism to track the confidentiality and integrity of data. Moreover, a simple algebra that combines confidentiality and integrity is implemented in hardware. Although this paper emphasized simplistic implementations, the proposed solutions appear very reasonable for more realistic implementation scenarios.

However, the preliminary work in this paper does not provide realistic implementations. Nor, does it consider ap-

plications that may benefit from this approach. Therefore, the work in this paper lays the foundation for both of these future-work investigations. There is a need to: consider 5-stage and 8-stage MIPS architecture pipelines (the 5-stage is the typical academic implementation however the 8-stage is the currently defined architecture), control and timing issues, caching and memory management. Hardware applications include the appropriate handling of storage and transfer of tagged information. For example, it is possible to implement a removable hard disk controller that disallows data storage based on a confidentiality threshold value; this may be one solution to restricting sensitive data from being stored on lap-tops or other mobile platforms. However, beyond these types of applications, the tagging of each datum in a large-scale database environment imposes significantly greater storage and communication overheads. Yet, it is likely that subsets of data can be abstracted by single tags, thereby compressing the storage and communication requirements; the computational processing of the tags may also need to be so modified.

REFERENCES

- [1] *Security Requirements for Cryptographic Modules*, National Institute of Standards and Technology, May 2001.
- [2] J. Vijayan, "Massive data breach puts VA's IT policies under a microscope," May 2006. [Online]. Available: <http://www.computerworld.com>
- [3] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman, *The Design and Implementation of the 4.4 BSD Operating System*. Addison-Wesley Longman, Inc., 1996.
- [4] *Programming the MIPS32 34K Core Family*, MIPS Technologies, Inc., September 2005, revision 01.05.
- [5] R. L. Sites, "Alpha AXP architecture," *Communications of the ACM*, vol. 36, no. 2, pp. 33–44, February 1993.
- [6] *IBM 4758 Models 2 and 23 PCI Cryptographic Coprocessor*, IBM, May 2004.
- [7] *IBM PCI Cryptographic Coprocessor, General Information Manual*, 6th ed., IBM, May 2002.
- [8] D. A. Patterson and J. L. Hennessy, *Computer Architecture A Quantitative Approach*, 2nd ed. Morgan Kaufmann Publishers, Inc, 1996.
- [9] K. Shimizu, "The cell broadband engine processor security architecture," IBM, April 2006. [Online]. Available: <http://www-128.ibm.com/developerworks/power/library/pa-cellsecurity>
- [10] "74F85 4-bit magnitude comparator," September 1994.
- [11] *MIPS32 Architecture for Programmers. Volume I: Introduction to the MIPS32 Architecture*, MIPS Technologies, Inc., July 2005, revision 2.50.
- [12] *MIPS32 24K Processor Core Family Software User's manual*, MIPS Technologies, Inc., June 2005, revision 03.05.
- [13] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design The Hardware/Software Interface*, 3rd ed. Morgan Kaufmann Publishers, Inc, 2004.
- [14] "6.823 computer system architecture: Bus-based mips implementation," September 2005, available as part of assignment Problem Set 1, Course 6.823 Computer System Architecture, Fall 2005, Instructors in-charge: Arvind and Krste Asanovic, catogrizred in MIT's OpenCourseWare under Electrical Engineering and Computer Science. [Online]. Available: <http://ocw.mit.edu>
- [15] B. J. d'Auriol and R. Molakaseema, "A parameterized linear array with a reconfigurable pipelined bus system: LARPBS(p)," *The Computer Journal*, vol. 48, no. 1, pp. 115–125, January 2005.
- [16] Y. Pan and K. Li, "Linear array with a reconfigurable pipelined bus system — concepts and applications," in *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '96), Vol. III*, H. Arabnia, Ed., Sunnyvale, California, USA, August 1996, pp. 1431–1441.