Author Retrospective

AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing

G. Edward Suh*, Christopher Fletcher†, Dwaine Clarke†, Blaise Gassend†, Marten van Dijk‡, Srinivas Devadas†

* Cornell University, ECE – suh@csl.cornell.edu
† Massachusetts Institute of Technology, CSAIL – {declarke, gassend}@alum.mit.edu, {cwfletch, devadas}@mit.edu
‡ University of Connecticut, ECE – vandijk@engr.uconn.edu

ABSTRACT

AEGIS is a single-chip secure processor that can be used to protect the integrity and confidentiality of an application program from both physical and software attacks. We briefly describe the history behind this architecture and its key features, discuss main observations and lessons from the project, and list limitations of AEGIS and how recent research addresses them.

Original paper: http://dx.doi.org/10.1145/782814.782838

Categories and Subject Descriptors

C.1 [**Processor Architectures**]: Miscellaneous; D.4.6 [**Operating Systems**]: Security and Protection

Keywords

Certified execution, software licensing, secure processors

1. HISTORY

The AEGIS project began with the question of what hardware's role in security should be. At the time, security was a relatively new topic to computer architects and there were only a few previous studies in the area of architectural support for security. One notable work from the academic community was XOM [10], which proposed to only trust processor hardware to provide a protected execution environment. XOM needed this level of security to support software copy protection. We found the idea of only trusting a single processor chip to be particularly promising, especially given the trend of remote computation on the Internet such as peer-to-peer or cloud computing and the proliferation of embedded and mobile devices. In this sense, XOM largely inspired the security model and the high-level approach of AEGIS.

We started our work in this direction by developing mechanisms to protect off-chip memory [5], which we believed were essential to enable a "single-chip" secure processor but were missing at the time, including in [10]. This experience and further investigation of potential application scenarios helped us identify key high-level design considerations for AEGIS. First, we found that the cost of protection mechanisms could be substantial and that application scenarios had varying security requirements. Based on these efficiency concerns, we felt that the secure processor should to be able to provide flexible protection that matches each application's needs. Second, to be useful in remote computations, we found that the security of the computation must be verifiable externally in addition to being protected by the processor. Finally, there was a question about how much software needs to be trusted especially because the hardware complexity of completely removing trust in an operating system turned out to be non-trivial.

The AEGIS paper in the 2003 ICS presented the high-level architecture that would enable an application to choose its protection levels and verify its computation externally. The paper also investigated multiple design choices based on the level of trust in system software. In that sense, the main contribution of the paper was to provide a framework for single-chip secure processor designs by identifying key security capabilities, to describe the protections necessary for each type of attack, and to discuss how best to implement those protections. In a subsequent paper, we presented a detailed implementation of one particular design point [13].

AEGIS explicitly identified two protection levels for secure execution environments. First, *tamper-evident computing* is intended to ensure the integrity of a computation. In this environment, any tampering from outside the protected program such as unintended changes to program memory should be detected. Second, *private tamper-resistant computing* requires an additional confidentiality guarantee that an adversary cannot learn about the user's secret data. This distinction means that additional protection overheads are paid only when necessary, depending on application requirements. For example, crowd sourcing applications like SETI@home require integrity, but seldom require confidentiality. On the other hand, users usually expect confidentiality when they outsource computation with their own sensitive data.

The AEGIS paper investigated the tradeoffs based on how much trust is placed on software components, in particular operating system (OS) functions. General-purpose computing systems often rely on OS support to provide the ability to time-share resources (e.g., CPU and memory) and manage multiple users, among other responsibilities. Yet, OS code is often quite complex, which results in vulnerabilities that at-

ICS 25th Anniversary Volume. 2014
ACM 978-1-4503-2840-1/14/06.
http://dx.doi.org/10.1145/782814.782838.

tackers use to break the security of the system. For security, some or all of the OS functions can either be moved to or checked by hardware so the software trusted computing base shrinks. However, more security-related functions in hardware come at the cost of reduced flexibility and increased design complexity. To investigate this tradeoff, AEGIS presented two designs: a version that trusts a small piece of critical OS code, and a version that does not trust any OS code.

AEGIS also described which protection mechanisms are necessary given different attack types. For example, conventional virtual memory and privilege levels are sufficient to protect the memory contents from a pure software attack. On the other hand, to prevent hardware attacks that physically read memory by tapping the main memory bus, encryption is necessary. Similarly, the design that assumes an untrusted OS requires additional protection mechanisms that are not necessary with trusted OS components.

2. OBSERVATIONS AND LESSONS

Hardware Root-of-Trust.

For systems that are physically exposed to potential adversaries, hardware must provide a root of trust that cannot be duplicated or altered. Thus, a system's root keys should be kept secret in hardware. The most fundamental security functions should also be implemented as ROM code or custom hardware to ensure their integrity. For example, today's mobile devices such as smartphones and tablets rely on a secret key inside the processor hardware to authenticate the device and a secure booting mechanism in ROM code to check the integrity of a boot loader.

Single-Chip Trusted Computing Base.

History also suggests that the single-chip approach of minimizing trusted hardware has important benefits in building secure systems in practice. Having multiple chips in the hardware trusted computing base turned out to pose challenges in both securing inter-chip communication channels from physical attacks and coordinating multiple manufacturers. As an example, Intel's TXT technology [2] relies on both an Intel processor and a third-party (external) chip called a TPM [15] to provide hardware-based security functions. TXT+TPM has been successfully attacked by probing or tampering with the processor-TPM bus [12, 16]. While TXT+TPM has been available for several years, the technology has not been widely utilized.

In 2013, Intel announced the next-generation of TXT called Software Guard Extensions (SGX) [9]. SGX works by maintaining one or more enclaves, where each enclave is a memory region that is inaccessible to software running at any non-enclave privilege level. Like TXT, code/data loaded into an enclave is measured and the result/starting state can be signed before it is returned to a user. Interestingly, SGX takes the third-party TPM out of the picture and adopts a single-chip approach, only relying on the main processor. SGX also performs automatic encryption and integrity verification on all data that leaves the processor. AEGIS proposed these techniques to protect against physical attacks.

Similarly, game consoles and mobile devices also integrate their basic security functions into the main processing chip instead of utilizing an auxiliary chip. These decisions are likely to have been taken in order to provide a proper level of security against physical attacks.

Hardware's Role in Reducing the Software TCB.

It is now widely accepted that a complex piece of software cannot be trusted to be secure. For example, a large monolithic OS has been repeatedly shown to have security vulnerabilities. In that sense, hardware support to reduce the software trusted computing base is necessary to build a secure system.

However, exactly how much software should be trusted and how critical security functions should be implemented are open design choices. In the AEGIS project, after investigating two design points with and without trusted software, we implemented the one with a trusted security kernel to get the flexibility of software in executing certain tasks, and the security of hardware for the remaining operations. While the ability to tolerate a completely untrusted OS was attractive, full hardware protection added significant design complexity and could only support a very restrictive protection model. We remark that Intel's SGX also chose to rely on a special piece of system software, which Intel provides, to implement TPM-like functionality.

On one hand, cryptographic primitives are good candidates for hardware implementations because they are relatively static, seldom requiring a patch once implemented correctly, and their efficiency can often benefit significantly from parallelism in hardware. Indeed, AEGIS implemented encryption and hashing functions for memory protection in hardware for performance reasons. Further, a cryptographic operation implemented in hardware is nearly always more resistant to attacks compared to the same operation in software. For example, Intel's hardware AES-NI extensions prevent cache side-channel attacks on AES [8].

On the other hand, for infrequent and control-intensive functions, the added complexity and reduced flexibility of a hardware implementation is often difficult to justify. For example, a context switcher in hardware is likely to be implemented as a sequential state machine and is in a sense equivalent to a fixed microcode routine. Therefore, if the integrity of the software routine can be ensured by, for example, checking the hash at run-time, both hardware and software implementations arguably provide comparable security and performance, and a trusted software implementation will be preferable. In fact, the AEGIS implementation relied on ROM code for infrequent and complex security functions such as entering or exiting a protected execution environment.

In essence, architects need to consider the overall system complexity of both hardware and software combined, in addition to efficiency and security, to make a decision on where to place a security function. For certain cases, we found that the complexity can be reduced by having a trusted component verify the operations of an untrusted one. For example, the AEGIS design with an untrusted OS had a hardware verifier, which is much simpler than the context switcher itself, to check the context switching operation in the OS. However, simply moving one function from software to hardware may not necessarily reduce the overall complexity of the trusted computing base.

3. LIMITATIONS

Untrusted Programs and Side/Covert Channels.

The AEGIS architecture was designed to protect a program from external software and physical attacks, but assumed that the protected program itself is *trusted* and *well-written*. A malicious program, on the other hand, may produce a bogus result or intentionally leak the user's private

data. Also, AEGIS did not provide protection against side-channel or covert-channel attacks. As a result, a protected program may still leak confidential information if its observable behaviors, such as memory access patterns [7] or execution time [1], depend on the confidential data.

AEGIS made the above assumptions because it mainly considered application scenarios where a program was written by a trusted party. In fact, these assumptions are fairly common for many secure processor designs. For example, Intel SGX assumes software running inside the enclave is trusted, and requires the user program to wipe register values upon a normal enclave exit if those registers stored confidential data.

However, the assumptions of trusted and well-written programs pose limitations on applicable application scenarios and on the level of security provided by AEGIS. For example, in many cloud computing scenarios today, users do not own or see the program that processes their data (e.g., the algorithms in Google Docs are kept secret by Google). Further, these types of programs are frequently updated. As a result, it is difficult (if not impossible) for users to vet these programs. Second, being "well-written" to prevent unintentional leakage is difficult and/or incurs large overhead. In fact, the only "fool proof" way to quell unintentional leakage channels is to use Fully Homomorphic Encryption [6], which currently has over a billion times overhead when running even simple programs.

To address these issues, a new single-chip secure processor called Ascend is being designed to run *untrusted* programs [3]. The key idea behind Ascend is *program obfuscation*: an adversary should not be able to tell what program or data is running inside Ascend by observing the Ascend chip's external pins. To achieve this security level, Ascend is built with a primitive called Oblivious-RAM (ORAM) that keeps main memory encrypted and additionally shuffles memory to prevent leakage through the program's address pattern [11]. Further, Ascend uses information-theoretic principles to bound leakage over timing channels (e.g., program runtime and ORAM access rate) [4]. On the other hand, Ascend's security comes with a more limited usage model. Currently, Ascend runs programs on bare metal and does not allow sharing resources with other users. Ascend can only run programs whose IO is limited to Ascend's ORAM — such as batch programs and stream computations [17].

Secure User IO.

The AEGIS proposal is also limited in its support for secure IO channels. AEGIS relies on a program itself to secure its communication channels using cryptographic protocols. This approach is relatively easy to apply for network communications where both sides can be authenticated and trusted, but difficult to use for local user interfaces such as displays and keyboards.

In fact, providing secure yet rich user interfaces is a challenge that still needs further investigation [18]. In traditional systems, user interfaces involve multiple components on a system such as a processor, a south bridge, and an IO device. As a result, the processor needs to be able to secure communications among multiple distributed components and also maintain consistent security policies across them. Modern system-on-chip (SoC) designs mitigate this problem by integrating most of the IO components into a single chip. However, even for SoCs, secure user IO requires device drivers to be trusted, adding to the TCB.

Virtualization Support.

AEGIS was designed in the context of traditional systems that run a single operating system. Today, however, virtualization has become a norm especially for cloud computing, allowing a single processor to run multiple operating systems managed by a hypervisor. Recent studies (e.g., [14]) showed that the general secure processor approach in AEGIS can be applied to the virtualization context, protecting a virtual machine (or an application) even from untrusted hypervisors, thereby reducing the trusted computing base.

4. CONCLUSION

In retrospect, the high-level approach in AEGIS that relies only on a single processor chip to protect a program's execution turned out to be a promising direction which continued in more recent systems. The architecture had some limitations and much research still remains to be done to fully realize the promise of single-chip secure processors.

5. REFERENCES

[1] D. J. Bernstein. Cache-Timing Attacks on AES. Technical report, 2005.

[2] David Grawrock. The Intel Safer Computing Initiative: Building Blocks for Trusted Computing, 2006.

[3] C. Fletcher, M. van Dijk, and S. Devadas. Secure Processor Architecture for Encrypted Computation on Untrusted Programs. In *STC; an extended version is located at http://csg.csail.mit.edu/pubs/memos/Memo508/memo508.pdf (Master's thesis)*, 2012.

[4] Fletcher, Christopher and Ren, Ling and Yu, Xiangyao and Van Dijk, Marten and Khan, Omer and Devadas, Srinivas. Suppressing the Oblivious RAM Timing Channel While Making Information Leakage and Program Efficiency Trade-offs. In *HPCA*, 2014.

[5] B. Gassend, G. E. Suh, D. Clarke, M. van Dijk, and S. Devadas. Caches and Merkle Trees for Efficient Memory Integrity Verification. In *HPCA*, 2003.

[6] C. Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *STOC*, 2009.

[7] O. Goldreich and R. Ostrovsky. Software Protection and Simulation on Oblivious RAMs. In *J. ACM*, 1996.

[8] S. Gueron. Intel Advanced Encryption Standard (AES) New Instructions Set, 2012.

[9] Intel. Software Guard Extensions Programming Reference, 2013.

[10] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. Architectural Support for Copy and Tamper Resistant Software. In *ASPLOS*, 2000.

[11] Ling Ren and Xiangyao Yu and Christopher Fletcher and Marten van Dijk and Srinivas Devadas. Design Space Exploration and Optimization of Path Oblivious RAM in Secure Processors. In *ISCA*, 2013.

[12] E. Sparks. A Security Assessment of Trusted Platform Modules. Technical Report 597, 2007.

[13] G. E. Suh, C. W. O'Donnell, I. Sachdev, and S. Devadas. Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions. In *ISCA*, 2005.

[14] J. Szefer and R. B. Lee. Architectural Support for Hypervisor-Secure Virtualization. In *ASPLOS*, 2012.

[15] Trusted Computing Group. TCG Specification Architecture Overview Revision 1.2. http://www.trustedcomputinggroup.com/home, 2004.

[16] Winter, Johannes and Dietrich, Kurt. A Hijacker's Guide to the LPC Bus. In *EUROPKI*, 2012.

[17] Yu, Xiangyao and Fletcher, Christopher W and Ren, Ling and van Dijk, Marten and Devadas, Srinivas. Generalized External Interaction with Tamper-Resistant Hardware with Bounded Information Leakage. In *CCSW*, 2013.

[18] Zhou, Zongwei and Gligor, Virgil D. and Newsome, James and McCune, Jonathan M. Building Verifiable Trusted Path on Commodity x86 Computers. In *IEEE S&P*, 2012.