Intel® Technology Journal

JUNE 2009

Advances in Internet Security

Intel Technology Journal

Publisher
Richard Bowles

Managing Editor
David King

Content Architect
David Durham

Program Manager
Stuart Douglas

Technical Editor
Marian Lacey

Technical Illustrators
Richard Eberly
Margaret Anderson

Technical and Strategic Reviewers
Ernie Brickell
Joe Cihula
David Durham
Howard Herbert
Eric Mann
Prasanna Mulgaonkar
Ravi Sahita
Jessie Walker
Rasheed Yasser

Intel Technology Journal

Copyright © 2009 Intel Corporation. All rights reserved.
ISSN: 1535-864X
ISBN 978-1-934053-22-5
Intel Technology Journal
Volume 13, Issue 2

This book is printed on acid-free paper. ♾

Publisher: Richard Bowles
Managing Editor: David King

Library of Congress Cataloging in Publication Data:

Printed in the United States

10 9 8 7 6 5 4 3 2 1

First printing June 2009

INTEL® TECHNOLOGY JOURNAL
ADVANCES IN INTERNET SECURITY

Articles

FOREWORD

David Durham
Principal Engineer
Security & Cryptography Research
Intel Labs

"The Internet now faces threats that are fundamentally unique to the virtual world."

The Internet remains full of promise but also peril. As the world becomes increasingly interconnected, barriers are breaking down: information can travel virtually anywhere in the blink of an eye and be accessible to almost anyone. However, as commerce, content, and personal information move en masse on-line, the motives for malice follow. The Internet now faces threats that are fundamentally unique to the virtual world. While the physical world of brick and mortar deals effectively with malicious individuals who have to abide by the constraints of space and time, in the virtual world, botnets are forming vast overlay networks of zombie machines ready to do the bidding of a single master. Blended threats combine the best-known methods for individual attacks into entirely new composite forms, constantly changing to stay a step ahead of security solutions. Meanwhile, the inherent need for information replication, search, and dissemination creates ample opportunities for eavesdropping and identity theft. The vastness of the Internet requires an equally vast solution, one that makes the old archetypes of the past seem quaint in comparison. This issue of the Intel Technology Journal describes some of the steps Intel is taking to help stem the tide of attack.

"Like a series of airlocks, partitioning and compartmentalizing software components reduces exposure to a single failure, helping to fundamentally contain a point of compromise."

The first task before us is redefining the network endpoint itself. No longer just a machine at the other end of the wire, the network endpoint becomes a composition of independently measured and protected software services, establishing a basis of good citizens in the online community. By leaving nowhere for malware to hide, security solutions can detect the stealthy rootkits and viruses that would otherwise infect and then lie dormant, waiting for commands to distribute spam, spread malware, steal information, or launch denial-of-service attacks. New models for attestation can directly validate individual programs thereby enabling remote entities to trust the specific software services with whom they are communicating. Finally, like a series of airlocks, partitioning and compartmentalizing software components reduces exposure to a single failure, helping to fundamentally contain a point of compromise.

Intel is also aggressively improving the power and performance of computing in general and cryptographic operations and algorithms in particular. Securing every network connection is becoming a real possibility. Data can be cost-effectively protected in transit and while at rest. New cryptographic instructions, simultaneous multithreading, and optimized cryptographic algorithms help to make the choice between no security and security obvious.

"Securing every network connection is becoming a real possibility."

Another challenge is scaling trust within the vastness of the Internet. Intel is developing new algorithms that provide anonymous attestation, preserving an individual's privacy while still establishing trust at a distance. Revocable group identities can vouch for systems and software anonymously, scaling trust by removing the need for establishing individual identities for everything in the Internet. Also, even as attacks become increasingly distributed, so can the solutions. Intel's research demonstrates that enlisting a broad array of endpoints to detect, report, and analyze anomalies in traffic patterns may be the answer to botnets in the Internet. Finally, community-based security solutions improve awareness and establish reputations in ad hoc infrastructures, absent of central administration.

While the vision of a completely safe Internet will likely remain elusive, much progress is being made. Steps are being taken in hardware to break the cycle and end the arms race between malware and security solutions, finally giving the good guys the upper hand. Endpoints are becoming more robust, enabling better software practices. Information can be kept private, even when distributed broadly, without the performance penalties of the past. Finally, scalable security solutions are being designed to work across the vast scale of the Internet, providing trust of and for the masses. It is my real pleasure to work with Intel Labs with a great team of researchers creating innovative solutions to the Internet's security challenges, now on display in this issue of the Intel Technology Journal.

"Steps are being taken in hardware to break the cycle and end the arms race between malware and security solutions, finally giving the good guys the upper hand."

ENHANCED DETECTION OF MALWARE

Contributors

Carlos Rozas
Intel Corporation

Hormuzd Khosravi
Intel Corporation

Divya Kolar Sunder
Intel Corporation

Yuriy Bulygin
Intel Corporation

Index Words

Cloud Computing
Anti-Virus
Malware
Rootkits
Virtualization
Runtime Integrity

Abstract

A significant development in the malware landscape in recent years is the ability of hackers to monetize compromised platforms by (1) gathering valuable information that can be sold, (2) using the platform's resources to aid in an illicit or unwanted activity, or (3) holding information contained on the platform for ransom. Since the attacker's potential monetary reward is increased the more the malware is undetected, a re-emergence of malware that can mask its presence from traditional security agents has occurred. This type of malware is referred to as stealth malware.

Researchers and industry have found novel uses for cloud computing to detect malware. In this article, we present an overview of these uses and identify their shortcomings. We present a cloud-computing-based architecture that improves the resiliency of the existing solutions, and we describe our prototype that is based on existing Intel platforms. We examine the new firmware that makes the existing architecture more robust. Our new platform-based tool can be utilized by security providers to help them keep pace with stealthy malware.

Introduction

Over the last three years, malware has evolved to support the new goal of malware writers and developers: to profit from their exploits. This for-profit goal has sparked the development of malware that can mask its presence on a platform. Some malware will go so far as to remove less stealthy malware from an infected computer to help avoid detection of that malware.

The cost of malware to businesses worldwide has been estimated to be in the tens of billions of dollars each year: 14.3 billion dollars in 2006 alone [1].

IT security faces a number of different challenges in combating the threat of malware. First of all there has been an explosion in malware samples. Panda Security reported that an average of 35,000 malware samples were detected each day in 2008, with the total count exceeding 15 million samples [2]. McAfee Inc. reported that the number of malware samples in their collection doubled from 10 million in March 2008 to 20 million in March 2009 [3]. This explosion in the number of samples underscores the reality that no client can have an up-to-date list of known malware at any given time. Moreover, security agents are required to spend ever more resources to test files against the multitude of known malware signatures. In certain situations, security agents consume 50-60 percent of the CPU resources [4].

"This for-profit goal has sparked the development of malware that can mask its presence on a platform."

Considering the ubiquity of malware samples, academia and industry have identified opportunities to use cloud computing to detect malware [5, 6]. There are a number of possible cloud-computing solution models. Figure 1 shows a generic system architecture for a cloud-based, anti-virus service. One model is a service model, where a host runs a lightweight process that collects relevant samples (such as files) and sends them to a network service. The network service performs the analysis to determine if the sample contains malware, and if so, it directs the lightweight process to quarantine the sample. Another approach is where the host agent maintains only a subset of the known malware signatures and a list of common software applications.

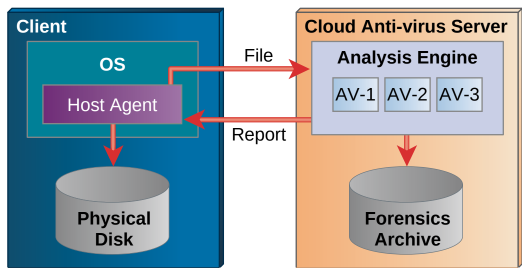


Figure 1: Cloud-based Anti-virus Service
Source: Intel Corporation, 2009

Cloud computing provides a number of benefits to malware detection. It reduces the amount of storage and computational resources on the client, and it simplifies the management of signature files, as it is centrally located. Moreover, whenever a previously unidentified malware sample is presented to the cloud, the security vendor can apply much more sophisticated and computationally expensive heuristics to determine the threat profile of the software.

Cloud computing, however, does not protect host agents from malware. Host agents need mechanisms to prevent or detect the agents that have been disabled or subverted. A number of proposals have been put forth to provide a better protection mechanism for host agents [7, 8, 9, 10, 11, and 12]. A number of these approaches center on the use of virtualization to provide an isolated execution environment for security agents. In this article, we examine platform features that can be used to isolate the host agent in order to provide protection against different threat vectors.

Organization of this Article

We start out by discussing threats to host agents. We then outline a generic architecture for malware detection, based on enhanced cloud computing. We continue with a description of how Intel platform technologies can be used to enhance computing solutions, and we end with a threat analysis of the approaches discussed.

Threats to Host Agents

The host agent on the platform must provide reliable information to the cloud service to be effective, just as host-only malware detection systems have to do to be effective. If malware is able to exploit vulnerability in the system (for example, a buffer overflow in a browser plug-in) and subvert the host agent, it can execute undetected.

These are some ways the host agent can be subverted:

- Tampering with the host agent. The host agent executable is modified so that it no longer poses a threat to the malware sample. Such tampering can be as simple as no longer sending files to the cloud service, or as elaborate as allowing the malware agent to filter the files that are sent to the cloud service.

"If malware is able to exploit vulnerability in the system and subvert the host agent, it can execute undetected."

*"Another method discussed by security researchers is to install a malicious virtual machine monitor (VMM) to **hyperjack** an operating system."*

- *Disabling the host agent.* The malware modifies the system configuration to either no longer launch or to suspend execution on the agent.
- *Input filtering.* The malware filters the information provided to the host agent by hooking the invocation of the system API and inserting malicious code to filter the results. Well-known hook points include the import table and the system call table. However, many more hook points exist; Wang et al. identified 41 potential file-hiding kernel hook points for the Red Hat Fedora core [10, 13].

In the last few years, malware has evolved to focus on more subversive methods of breaching system security. One such method was used by Shadow Walker wherein the interrupt descriptor table (IDT), page-fault handler was hooked. This caused the processor to return certain values when reading memory as data and other values when reading memory as code [14]. Another method discussed by security researchers is to install a malicious virtual machine monitor (VMM) to *hyperjack* an operating system (OS) [15]. The VMM affords the researcher the ability to observe the system without requiring any modification or hooking of the OS.

Enhancing Cloud-based Malware Detection

We illustrate a system architecture to enhance cloud-based, anti-virus services in Figure 2.

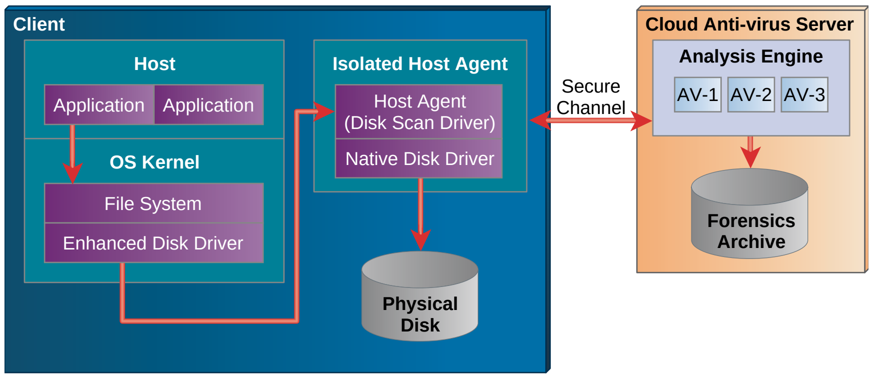


Figure 2: Enhanced Cloud-based, Anti-virus Solution
Source: Intel Corporation, 2009

"Since the host agent partition does not need to support general-purpose computing, it can be configured to be more secure."

By isolating the host agent from the host environment and by providing direct access to platform resources, such as storage and memory, malware in the host can no longer attack or manipulate the host agent directly. It must instead attack the host agent partition. Since the host agent partition does not need to support general-purpose computing, it can be configured to be more secure resulting in a more robust solution. A description of the architectural components follows:

- *Isolated host agent environment.* An isolated execution environment contains the host agent. It supports an interface from which the host can send requests. It provides direct access to host storage, and host access; disk I/O requests can be directed to this environment.

- *Isolated host agent.* The host agent maintains a secure, authenticated channel with the cloud-anti-virus service. The host agent monitors the host-disk I/O, and if necessary, sends the files over the secure channel to the cloud-anti-virus network service for evaluation. The host agent contains the file system logic, corresponding to the host file system, and the agent can periodically scan the physical disk to find out what files have changed; it can then send the changed files over to the cloud-anti-virus network service.
- *Enhanced disk driver.* An enhanced disk driver can also be used to forward disk IO requests by the file system, from the primary partition to the host agent, running in the secure container, for further processing.
- *Native disk driver.* The native disk driver provides direct access to the host disk hardware from the isolated partition.

Figure 3 illustrates how a cloud-anti-virus service can be extended to provide kernel rootkit detection capabilities, in addition to disk/file scan capabilities for malware. A description of the architectural components follows:

"The two issues that come up in remote memory integrity operations are security and network latency."

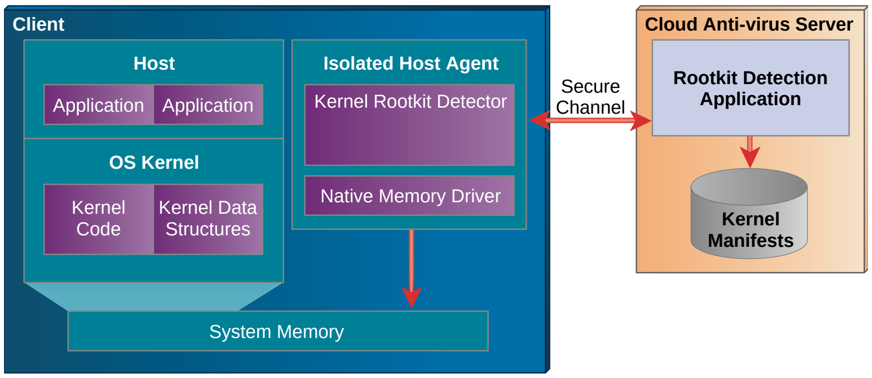


Figure 3: Cloud-based Kernel Rootkit Detection
Source: Intel Corporation, 2009

- *Kernel rootkit detector.* A local rootkit detector [9], running inside the client isolated partition, exposes secure remote interfaces to the rootkit detection application that is running on the cloud-anti-virus software. In this way, the rootkit application is able to access kernel memory pages and perform basic hash comparison operations on kernel memory regions that can be used to perform integrity checks. The integrity validation operations are run on the remote server. The kernel hashes are also stored in the cloud-anti-virus server and provided to the kernel rootkit detector on the client PC, if needed.
- *Native memory driver.* The native memory driver running in the isolated partition provides secure access to the area of system memory containing the kernel memory regions of the host OS.

The two issues that come up in remote memory integrity operations are security and network latency. We address the network security concerns by using the secure channel between the client PC and the cloud-anti-virus service, by providing interfaces for memory hash comparisons, and by restricting remote memory accesses. Network latency issues for memory validation are mitigated by the fact that most of the kernel memory sections that are checked for integrity reside in non-pageable memory on the client platform.

"We take advantage of this DMA engine to access memory regions."

The kernel rootkit detector helps mitigate unknown threats or in-memory threats by detecting commonly used attack methods such as import table hooking, kernel code and static data modifications, IDT, system call table hooking, and direct kernel object manipulation.

Prototype Architectures for Combating Stealth Malware

We developed two prototypes of the system just described to validate the system design. Prototype 1 is based on the Intel® Management Engine (Intel® ME) and Prototype 2 is based on a virtual machine monitor (VMM), both of which provide additional isolation from the OS. Because they are secluded from the host OS, it is harder for an attacker to compromise these environments.

Prototype 1: Based on the Intel® Management Engine

"At regular intervals, it verifies the integrity of the run-time image of the host agent."

When Intel® Active Management Technology (Intel® AMT) [16, 17], and platforms running Intel® vPro™ technology were introduced, the platforms contained an embedded microcontroller, called Intel ME. Intel ME appears as a separate integrated device on the PCI bus. It integrates different hardware engines such as bus controllers, crypto accelerators, DMA engines, and so on. Intel ME runs firmware that consists of a real-time operating system (RTOS), drivers operating the hardware engines, and manageability applications. In our prototype we take advantage of this DMA engine to access memory regions.

We first implemented an agent to scan the memory in the firmware. This agent is the traditional blacklist-based scanning agent. Because of the restrictions in Intel ME, both in terms of compute power and storage, we could only implement a limited scanning agent in Intel ME firmware. We were limited in the size of the blacklist that could be securely stored (192KB) and in the frequency of scanning operations. Considering these restrictions, we implemented a host agent in the host OS to scan the blacklist-based memory. In our prototype, we add to the Intel ME agent integrity firmware to verify the integrity of this host agent. Additionally, the Intel ME out-of-band (OOB) interface can communicate with any remote cloud-anti-virus service to notify the software if the host agent is modified at run time. Intel ME maintains the hash of the host agent in its storage area, and at regular intervals, it verifies the integrity of the run-time image of the host agent. In our paper, *Runtime Kernel Rootkit Detection* [9], we describe the manifest generation process and the 3-phase algorithm deployed to verify the run-time integrity of the host agent. For our prototype, we measured the integrity of the host agent by using Intel ME: the process was completed in the order of milliseconds. In future work, we propose to explore event-driven, host-agent scanning to address any timing attacks. Our prototype architecture is shown in Figure 4.

"In future work, we propose to explore event-driven, host-agent scanning to address any timing attacks."

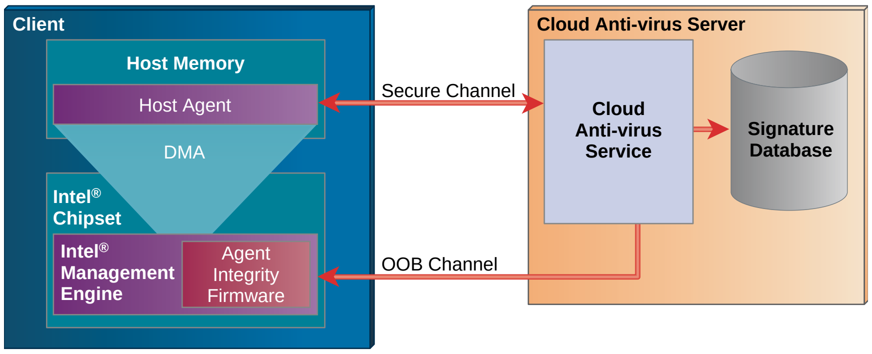


Figure 4: Intel® Management Engine Architecture
Source: Intel Corporation, 2009

Prototype 2: Based on the Virtual Machine Monitor

In this, the second of our prototypes, we considered a virtual machine (VM) as an isolated environment and also utilized the extension of our first prototype to measure the integrity of a VMM.

In order to understand why we chose VMM as an isolated environment, we first present a brief overview of a virtualization-based system that uses hardware virtualization. We utilized Intel® Virtualization Technology (Intel® VT) on our platform. Virtualization refers to the technique of partitioning the physical resources of a processor or a chipset into VMs and inserting a higher privilege executive under the OS. This executive is known as a VMM. The privilege level is called as VMX-root mode in Intel® Virtualization Technology (Intel® VT) for IA-32, Intel® 64 and Intel® Architecture (Intel® VT-x). A control transfer into the VMM is called a VMExit, and the transfer of control to a VM is called a VMEntry. A VM can explicitly force a VMExit by using a VMCALL instruction. A guest OS runs in VMX non-root mode that ensures that critical OS operations cause a VMExit. This allows the VMM to enforce isolation policies. We enhanced the prototype described in [9] by adding a light-weight VMM to this prototype. This VMM provides us the capabilities to monitor system events as required and to create shadow page tables as needed, in order to intercept paging events and modifications to data structures. System components, manifest generation, and an integrity verification algorithm are discussed in detail in [9]. We added additional Intel ME firmware to our research prototype to verify the integrity of the VMM itself [11]. The architecture is shown in Figure 5.

"Virtualization refers to the technique of partitioning the physical resources of a processor or a chipset into VMs and inserting a higher privilege executive under the OS."

"This VMM provides us the capabilities to monitor system events as required and to create shadow page tables as needed."

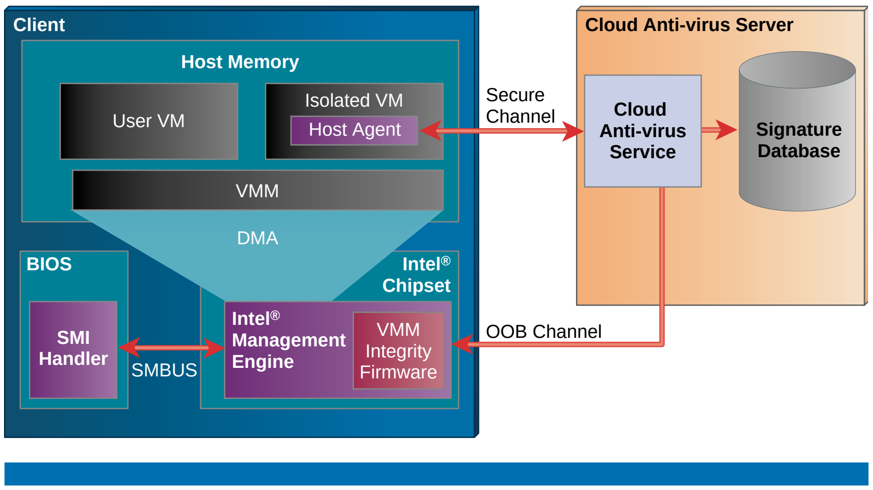


Figure 5: Virtualized Environment Architecture
Source: Intel Corporation, 2009

"SMM offers a distinct and easily isolated processor environment that is transparent to the OS or to the executive and software applications."

The key architectural components of our second prototype based on Deep Watch, as described in [11], are the Intel ME firmware with an integrity verification module and a VMM integrity application service inside the cloud-anti-virus service. With simple support from the BIOS system management interrupt (SMI) handler, the processor state and register information can be ascertained, and from these, Intel ME can reconstruct the virtual memory page tables for the VMM. System management mode (SMM) is a special-purpose operating mode that handles system-wide functions such as power management, system hardware control, or proprietary OEM-designed code. The main benefit of SMM is that it offers a distinct and easily isolated processor environment that is transparent to the OS or to the executive and software applications. When SMM (SMI handler) is invoked through an SMI, the processor saves the current state of the processor (the processor's context), then switches to a separate operating environment contained in system management RAM (SMRAM). This processor state can be gathered in the SMI handler and communicated to Intel ME via a hardware interface. The processor state obtained by Intel ME can be utilized to reconstruct memory page tables to verify the run-time integrity of the VMM. Additionally, Intel ME can communicate all the information (processor state and memory pages) to the cloud-anti-virus service. The remote anti-virus service can then verify the run-time integrity of the VMM, thus overcoming the computational limitations of Intel ME. We also built a similar research prototype to measure the integrity of host OS drivers from the PCI DMA device as described in [12].

Threat Analysis

We assume that the attacker has full access to and control of the OS, including the kernel, and is able to insert, modify, or delete kernel drivers; however, we assume the attacker is not able to modify Intel ME firmware or SMRAM. Our assumption implies that the attack space is large in scope and ranges from simple user-space attacks to the kinds of attacks that seek to modify critical kernel data structures so as to compromise the user OS or VMM itself. Examples of some of these kinds of attacks include hooking of the import table, IDT, or system call table, kernel code and static data modifications, and direct kernel object manipulation. For an overview of kernel rootkit techniques please refer to [18].

Following are the threat vectors we address in Prototype 1:

- *Threats to the host agent.* Kernel code modifications, import tables, IDT, and system call-table hooking are mitigated by the kernel rootkit detector in Intel ME. Intel ME has an OOB interface to read memory through its DMA interface, and thus it guarantees that the rootkit detector has an unobstructed and unmodified view of memory.
- *Unknown kernel attacks.* If the kernel rootkit detector in Intel ME detects any suspicious behavior or pattern, then it can communicate with the cloud-anti-virus server for a detailed scan.

"Our assumption implies that the attack space is large in scope."

In Prototype 2 we address the same threats as in Prototype 1 as well as VMM attacks:

- *Threats to the host agent.* The host agent is protected against attacks from malware by the VMM.
- *Unknown kernel attacks.* The host agent, enhanced with our kernel root detector, provides the ability to detect any suspicious behavior or pattern.
- *VMM attacks.* With direct access to memory and the Runtime Kernel Rootkit Detection (RKRD) system in Intel ME, a compromised VMM can be detected.

Summary

In this article we describe the motivation for using cloud computing in the fight against malware, as proposed by both academia and industry. We examine the threats against cloud-based, anti-virus services, which are primarily directed towards the host agents running on the clients that provide input to the cloud-anti-virus engine. We then propose some platform-based features, based on Intel architecture, that can be used to mitigate threats against host agents. Our research prototypes use a combination of Intel virtualization technology and Intel chipset technologies, such as Intel ME, to effectively mitigate most of the threats against host agents in cloud-anti-virus service environments. Thus, these new usages of our technologies can help bring the benefits of cloud-anti-virus services to our end customers.

"These new usages of our technologies can help bring the benefits of cloud-anti-virus services to our end customers."

References

[1] "The Economic Impact of Viruses, Spyware, Adware, Botnets and other Malicious Code." *Computer Economics, 2007 Malware Report*.

[2] Annual Report PandaLabs 2008. At ***http://www.pandasecurity.com***

[3] F. Paget. "Avert Passes Milestone: 20 Million Malware Samples." March ;10, 2009. At ***http://www.avertlabs.com***

[4] T. Watson. "Antivirus Vendors Push Toward Cloud Computing." *Dark Reading*, September 17, 2008. At ***http://www.darkreading.com***

[5] J. Oberheide et al. "Cloud AV: N-Version Anti-virus in the Network Cloud." In *Proceedings of the 17th Usenix Security Symposium*, pages 91-206, July 2008.

[6] McAfee, Inc. "Artermis Technology—Always–on, Real–Time Protection." *Whitepaper*, 2008. At ***http://www.mcafee.com***

[7] N. Petroni Jr. et al. "Copilot–a coprocessor–based kernel runtime integrity monitor." In *Proceedings of the 13th Usenix Security Symposium*, pages 179-194, August 2004.

[8] VMware, Inc. "VMsafe Security Technology." A set of web pages. At ***http://www.vmware.com***

[9] S. Grover et al. "RKRD: Runtime Kernel Rootkit Detection." *SECRYPT*, 2008. To be published by *Springer*.

[10] X. Zhao et al. "Towards Protecting Sensitive Files in a Compromised System." In *Proceedings of the Third IEEE international Security in Storage Workshop*, December 13, 2005.

[11] Y. Bulygin et al. "Chipset based detection and removal of virtualization malware." *Black Hat USA*, 2008.

[12] R. Sahita et al. "OS Independent Run-Time System Integrity Services." Intel Corporation 2005. *Whitepaper.* At ***http://www.intel.com***

[13] Z. Wang et al. "Countering Persistent Kernel Rootkits Through Systematic Hook Discovery." *11th International Symposium on Recent Advances in Intrusion Detection (RAID)*, Boston, MA, September 15–17, 2008.

[14] S. Sparks and J. Butler. "Shadow Walker: Raising the bar for rootkit detection." *Black Hat Japan*, 2005.

[15] J. Rutkowska. "Subverting Vista Kernel for fun and profit." *Black Hat USA*, 2006.

[16] Intel Corporation. "Built-in Manageability and Proactive Security." *Whitepaper*, 2006. At ***http://www.intel.com***

[17] O. Levy et al. "Advance Security Features of Intel® vPro Technology." *Intel Technology Journal*, Volume 12, Issue 04, December 2008.

[18] Skape and Skywing. "A Catalog of Windows Local Kernel-mode Backdoor Techniques." *Uniformed Journal*, Volume 8, September 2007. At ***http://www.uniformed.org***

Acknowledgments

We acknowledge our reviewers and contributors, among them Yasser Rasheed, Ernie Brickell, and David Durham.

Author Biographies

Carlos Rozas is a Senior Staff Security Researcher at Intel Labs in Hillsboro, Oregon. Carlos has 13 years research and development experience in the security area, including content protection, tamper resistant software, and software integrity. For the last four years, he has led research efforts in trustworthy virtualization that combines trusted computing technologies and virtualization. Carlos has a B.S. degree in Computer Engineering and Mathematics and a M.S. degree in Computer Engineering from the University of Michigan. His e-mail is carlos.v.rozas at intel.com.

Hormuzd Khosravi joined Intel in 1999 and currently works as a Software Architect at Intel Labs in Hillsboro, Oregon. His areas of specialization are security, networking, and manageability, and he has been involved with Intel® Active Management Technology (Intel® AMT) architecture since 2005. He holds seven patents in this area and has more pending. Hormuzd holds a B.S. degree in Electronics Engineering from Mumbai University, India and an M.S. degree in Computer Engineering from Rutgers University, New Jersey. His e-mail is hormuzd.m.khosravi at intel.com.

Divya Kolar Sunder is a Network Software Engineer at Intel Labs in Hillsboro, Oregon. Her research interests are in the areas of platform security, networking, and manageability. She joined Intel Corporation in 2005 and has been an active researcher in various security and manageability technologies such as Intel® Active Management Technology. Her current research focus is in chipset- and platform-based security technologies, and she has played an integral role in building proof-of-concept demonstrations from research concepts. She received her M.S. degree in Computer Science from Portland State University in 2006. Her e-mail is divya.kolar at intel.com.

Yuriy Bulygin is a Technical Lead in Intel's Security Center of Excellence. He is responsible for vulnerability analysis of Intel processor and chipset technologies. His primary interests are vulnerability analysis, exploit development, reverse engineering, and cryptography. Yuriy's education is in cryptography and applied math and physics from Moscow Institute of Physics and Technology. His e-mail is yuriy.bulygin at intel.com.

Copyright

PROTECTING CRITICAL APPLICATIONS ON MOBILE PLATFORMS

Contributors

Ravi Sahita
Intel Corporation

Ulhas Warrier
Intel Corporation

Prashant Dewan
Intel Corporation

Index Words

Application Isolation
Virtualization
Anti-Malware
Remote Attestation
TCB
Runtime Integrity

Abstract

The size and complexity of the privileged kernel of current operating systems (OSs) have been increasing at an alarming rate. Moreover, there is a direct correlation between vulnerability and the size and complexity of the software base on a PC platform. Stealth malware takes advantage of this complexity in the way it attacks PCs. Recently, malware has been increasingly used for automated and targeted attacks that steal user data from applications. Anti-virus software is limited to well-known signatures and does not address low-level rootkits that subvert the OS and all the services that security software depends on.

In this article, we describe our research prototype, P-MAPS, a processor-measured service layer that dynamically reduces the trusted computing base (TCB) and verifiably improves the runtime security of user's applications, without interrupting the typical operation of the user OS. We describe the P-MAPS architecture that was built using current Intel processors. Our dynamic usage approach reduces the execution footprint of P-MAPS, making it feasible to protect critical applications on power-sensitive mobile platforms. We also discuss some security usages that can benefit from P-MAPS.

Introduction

We first discuss the critical user applications that need to be protected; next, we describe the threat vectors by which malware can install itself, and we then outline our research goals, looking at them from a security and usability perspective.

Motivation

Regular reports from security vendors reveal that malware is becoming increasingly stealthier and more polymorphic. Most malware countermeasures are reactive, such as anti-virus scanning. These measures are no longer very effective in today's computing environment. Intrusion prevention systems address some threats, but also are susceptible to attacks themselves, since these software systems operate at the same privilege level as that of the attacks. Our approach to mitigating this present-day scourge is to protect the critical user applications, such that malware, while still continuing to execute, will not have any negative impact on the security of the application.

"Malware is becoming increasingly stealthier and more polymorphic."

Threat Vector

The threat vector we address with this research is software-based, automated malware attacks. Malware can install itself on the platform via any of the following vectors:

- *Internet downloads.* Unsuspecting users can be motivated into installing user-space or kernel-space malware on their platforms under the pretext of other useful software. An example of such malware is scare-ware. Scare-ware can spoof anti-malware software, software that masquerades as a custom codec for custom video formats. This type of malware is typically downloaded from the Internet. Web drive-by attacks are a subset of this attack vector where an infected web server can infect a client that visits it.
- *Buffer overflow.* A buffer overflow can be used to execute malware in the context of a supervisor or user process. The root cause of this infection vector is software vulnerabilities.
- *Network-based infection.* Automated worms propagate malware payloads via instant messaging, peer-to-peer networks, shared drives, and e-mail services.
- *Dropped by other malware.* Malware toolkits allow malware to extend its behavior by allowing the installation of variants or other malware payloads on an already infected computer.

"The threat vector we address is software-based, automated malware attacks."

Note that once malware is installed on the platform, it can use a combination of the following methods to attack PCs:

- *Code tampering.* Malicious software can tamper with application code thus changing the behavior of the application; for example, not encrypting sensitive data before sending them to the network.
- *Unauthorized data access.* Malware can snoop data from the application's memory or may modify data without authorization.
- *Screen scraping.* Malware can read the application screen buffer, extracting information from it.
- *Key logging.* Malware can hook kernel keyboard handlers thus allowing it to access user input.
- *Man-in-the-middle.* Malware can replace a valid application or a valid library with a malicious version thereby launching a man-in-the-middle between the user and a remote server.
- *Circumvention attacks.* Malware can obfuscate an application's resources thus ensuring the application does not run securely. This class of attack is called a circumvention attack, since the attack does not tamper with the application directly, but instead it attacks the environment the application interacts with.
- *DOS attacks.* Malware can prevent an application from running at all, or can prevent access to key resources the application may need, such as network I/O.

Research Goals

Our goal is to protect applications from software-based attacks that may originate from the infection vectors just listed. The types of attacks that P-MAPS can mitigate are application code tampering, unauthorized access of application data, screen scraping (protected in a limited manner where the application renders the screen buffer itself), and man-in-the-middle (for example, by running a secure network connection from the protected application). P-MAPS can address circumvention attacks if the library used by the application is also protected. Any use of untrusted libraries by an application are not protected by P-MAPS. Note that P-MAPS does not address DOS attacks on the application. Malware can prevent a P-MAPS-protected application from running, but the unprotected application will not be able to access the resources that P-MAPS has control over; for example, the unprotected application will not have access to secrets provisioned on the platform by a trusted third party (TTP).

"The types of attacks that P-MAPS can mitigate are application code tampering, unauthorized access of application data, screen scraping, and man-in-the-middle."

Security Goals

Our security goals center on the following activities being carried out:

- *Runtime authentication of applications.* To ensure that only valid (authenticated) applications are protected, we perform runtime measurement of the application to verify its integrity before affording it any protection. This goal is not specific to the application being protected but it ensures that the P-MAPS capability cannot be used by rogue software.
- *Runtime, in-place protection of applications.* Once the application is authenticated, we protect its code and data memory in-place within the OS. This approach is in contrast to approaches that isolate the applications into a separate OS or virtual machine [1].
- *Reduction of trusted computing base (TCB).* The OS is a general-purpose environment where users can install unknown and potentially malicious kernel modules that can attack a user's applications. Hence, we reduce the large TCB [2] that trusted third parties depend on by a significant factor by removing the OS services from the protected application's TCB. The applications that can restrict their use of system services to memory allocation and de-allocation benefit the most from this TCB reduction.
- *Remote verification of protected execution.* The platform should be able to report the state of protected applications. An independent remote verifier should be able to verify the authenticity of the attestation report and its contents.

Usability Goals

Any security capability that conflicts with usability is typically not used. Hence, we have the following set of usability goals:

- *The existing programming model should not be changed.* It should be possible to use P-MAPS on existing applications making only minor changes to the application.
- *Low-power and performance impact when protection is active.* In order to use P-MAPS on low-power platforms, such as notebooks and mobile Internet devices (MIDs), the expected power overhead of P-MAPS must be minimal and must not impact the power performance of the device when protection is not active.

- *No impact on application interaction with the OS.* P-MAPS should not impact the OS-scheduled execution of protected and unprotected applications that are executing on the OS. Note that protected applications can still use system services, and the services will have access to only the data that the protected application exposes. However it is important to note that such interaction should be limited to operations that are expected to be untrusted.
- *Co-existence with other hardware-based security solutions.* P-MAPS can co-exist with other software components that use the hardware capabilities it uses—Intel® Virtualization Technology (Intel® VT) and Intel® Trusted Execution Technology (Intel® TXT). P-MAPS uses these capabilities in a dynamic manner: it uses Intel VT controls while an application is being protected, and it relinquishes Intel VT controls when the application is turned off.

Software Architecture

Overview

At a high level, the two stable states of the platform, when using P-MAPS, are shown in Figure 1. The platform starts with a commodity OS (currently the *host*) running on hardware, as shown in state **A** in Figure 1. P-MAPS is instantiated via user launch of an application that requires P-MAPS services. The resulting state of the platform is as shown in state **B** in Figure 1: only the P-MAPS core, the CPU, the verified chipset, and BIOS are in the TCB. Note that the OS is running in *guest* mode.

In the rest of this section we describe how the architecture of P-MAPS achieves the smaller TCB, shown in state **B**, as well as how the application is added to this TCB at runtime. The primary contribution of our research is on-demand reduction of the TCB to one or more independently protected applications that execute without interrupting the operation of other unprotected applications or services executing on a commodity OS.

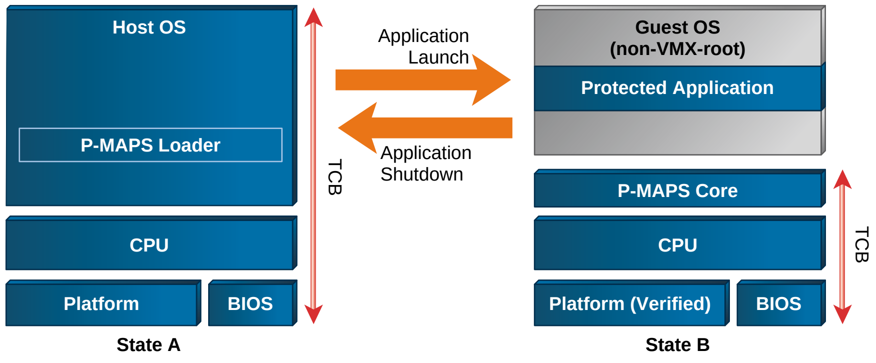


Figure 1: TCB States Before and After P-MAPS Launch
Source: Intel Corporation, 2009

Components

Intel® Trusted Execution Technology

Intel TXT is a set of CPU and platform extensions that provide a measured and controlled launch of system software that can then establish a protected environment for the system software and any additional software that it may execute. The Intel TXT use of the term *trusted* denotes a successful measurement of the provided software module to a reference measurement that is protected by a hardware trusted platform module (TPM) and is pre-provisioned on the platform. The software environment that is measured and launched is called the measured launch environment (MLE). MLEs may be system software, such as an OS kernel or a virtual machine monitor (VMM). MLEs can use different launch mechanisms and therefore use different types of measurement schemes. One measurement is made when the platform boots, by using a root of trust for measurement (RTM) that executes on each platform reset; the RTM creates a chain of trust that extends from platform reset to the measured environment. As the measurement always executes at platform reset, this type of RTM is called a static RTM (SRTM). Maintaining a chain of trust for a length of time may be challenging for an MLE that operates in an environment that is, under normal operation, exposed to unknown software entities, such as device drivers. P-MAPS relies on a small, static code base and runs the OS in a de-privileged mode. Running an MLE, an extra layer of code on power-sensitive platforms, incurs extra overhead and therefore is not a desirable means of addressing this issue. Intel TXT provides another RTM called a dynamic root of trust for measurement (D-RTM), also called a *late launch*. Using D-RTM, the launch of the measured environment can occur at any time without a platform reset. An Intel-signed, chipset-verified code module (known as an authenticated code module or ACM) is used to verify the state of the CPU and chipset, to ensure a secure state of the platform when an attempt is made to launch the MLE. It is therefore possible to launch an MLE, execute it for some time, terminate the MLE, and then launch the same or a different MLE again. An Intel TXT chipset and a Trusted Computing Group standards-based TPM (available from various vendors) are required to ensure correct operation of the D-RTM model. The chipset, enabled with Intel TXT, implements TXT *Heap* memory, which is a region of physically contiguous memory set aside by BIOS for the use of Intel TXT hardware and software. The software that launches the MLE passes data to the SINIT ACM and to the MLE by using the Intel TXT Heap memory. This heap region allows for secure handoffs to occur between the BIOS and the OS, between the OS and the SINIT ACM, between the SINIT ACM and the MLE, and finally between the OS and the MLE. The structure of the data passed between the OS and the MLE is system software specific. We shall describe the format used for P-MAPS later in this article. The other protection aspect of the Intel TXT chipset comes from DMA devices via Intel® Virtualization Technology (Intel® VT) for Directed I/O (Intel® VT-d)[3]. The key aspects of the TPM used by Intel TXT are also described later on in this article.

*"**Trusted** denotes a successful measurement of the provided software module to a reference measurement that is protected by a hardware trusted platform module."*

"Using D-RTM, the launch of the measured environment can occur at any time without a platform reset."

Trusted Platform Module

The trusted platform module (TPM) [4] provides the hardware root of trust for storage (RTS) and the D-RTM. The TPM contains the following capabilities that allow secure MLE measurement and recording of the MLE measurement:

Locality of access. TPM localities are essentially access levels that can be mapped to the privilege level of the software or hardware entity by using the TPM. Localities can also be used in access control lists for objects managed by the TPM. For example, trusted software, such as the MLE, is assigned a higher locality than untrusted software, whereas hardware is assigned a higher locality than the MLE (which is measured by the hardware). P-MAPS uses TPM Locality 2 to associate operations with the P-MAPS core. This binding is used for remote attestation of the applications protected by the P-MAPS. This operation is described in detail later in this article.

Platform configuration registers (PCR). PCRs are registers maintained in the TPM hardware that capture the software state of the system. The TPM exposes an *extend* operation on PCRs, which is an order-sensitive, one-way cryptographic hash operation. Additionally PCR state can be quoted via the TPM (that is, signed by the TPM with a key that is only known to the TPM) such that the PCR quotes can be verified by a remote verifier that can then attest to the software state of the system. Intel TXT uses PCR 17 and PCR 18, where PCR 17 holds the hash of the ACM (along with other static fields, explained in the *MLE Writers Guide* [5], section 1.9.1), and PCR 18 holds the hash of the MLE.

Endorsement key. The endorsement key is the root key pair provisioned in the TPM. It can be used to identify the TPM as a hardware TPM and also to derive additional key-pairs that can be used for attestation operations.

Storage root key. The TPM has a separate storage root key to protect its local non-volatile memory. This key can be used to seal (and subsequently unseal) data to the platform and the platform's software state (via TPM PCRs). P-MAPS uses the root key to bind data to the integrity of the application it is protecting.

Launch control policy (LCP). LCP is a local verification mechanism that is used to ensure that the MLE to be launched meets specific measurement criteria. The measurement criteria, or policy, may be defined by the platform owner, or as a default set by the platform supplier. LCP is enforced by the chipset ACM and the policies are stored in the TPM. A simple policy is a list of valid MLEs. When the ACM is executed (via the GETSEC[SENTER] CPU instruction), the LCP engine in the ACM reads the LCP from the TPM and compares the measurement of the MLE whose launch is being requested against the platform policy. If the policy matches, the measured environment is then launched.

Hardware Virtualization

Intel VT-x provides hardware support to virtualize the CPU and it allows a VMM to configure the events that transfer control to the VMM, via a VMexit control field. This capability is used by the P-MAPS core to virtualize the CPU translation lookaside buffer (TLB), which caches the virtual-to-physical address mappings. The VMM can use the Intel VT-x hardware capability to selectively transfer control to the VMM, when the OS performs memory management operations such as loading control registers, flushing the TLB (invlpg/invd), as well as page fault exceptions. Please refer to Intel software developers' manuals [6] for more information on Intel VT and Intel TXT.

P-MAPS Architecture

The P-MAPS module consists of the OS-specific P-MAPS loader and an OS-independent P-MAPS core (Figure 2). The P-MAPS loader uses the Intel TXT dynamic launch capability to authenticate and bootstrap the P-MAPS core (the MLE). The P-MAPS core then uses Intel VT to extend the protected environment that the P-MAPS MLE executes within. Thus, the P-MAPS core executes in the highest privilege mode (VMX root mode), which ensures hardware separation between the protected (target) applications and itself. The P-MAPS core provides three local properties for the application it protects. The P-MAPS core:

"Runtime protection must be able to attest to the protection state of the application to a remote verifier."

- isolates the program's memory from other software executing on the platform, even software with a higher privilege level, such as the OS;
- ensures encapsulation of application data memory such that only code in measured application pages can access application data; and
- prevents circumvention of any function entry points exposed by an application (such as a shared library).

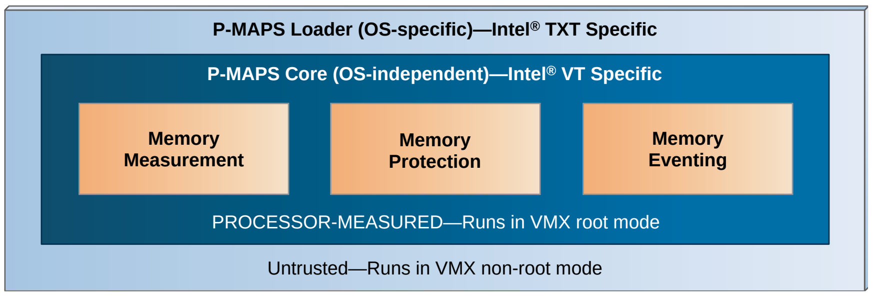

P-MAPS Loader (OS-specific)—Intel® TXT Specific
P-MAPS Core (OS-independent)—Intel® VT Specific
Memory Measurement
Memory Protection
Memory Eventing
PROCESSOR-MEASURED—Runs in VMX root mode
Untrusted—Runs in VMX non-root mode

Figure 2: P-MAPS Architecture
Source: Intel Corporation, 2009

The runtime protection for the application is important not just for the three local properties just listed, but also because it must be able to attest to the protection state of the application to a remote verifier. Intel TXT uses the TPM that provides us with the necessary storage and reporting mechanisms to ensure that the P-MAPS core that is loaded, is the one that was provisioned by the platform owner into the TPM LCP (that is, the whitelist of MLEs). Additionally, the hardware platform must ensure that virtualization is turned on only when virtualization is used after a successful measured launch. Moreover, the hardware platform ensures that the

P-MAPS core is measured and verified before it can enable Intel VT. By building the P-MAPS core to be run on-demand for protection of applications, we can leverage this approach on power-sensitive devices. The additional power used to run P-MAPS is not consumed until the application protection is needed.

P-MAPS Memory Services

The P-MAPS core provides memory services for measurement, protection, and eventing. The capabilities of each of these submodules are as follows:

Memory measurement. The P-MAPS core applies memory measurement to identify applications, based on an application integrity manifest. In essence, the application integrity manifest provides a signed list of integrity check values over the contents of the application's code and data. If there are relocation symbols in the pplication (for example, a dynamically loadable library) then those are captured in the manifest to aid in runtime measurement. The integrity manifest can be created for both executable and linkable format (ELF) and Windows* Portable Executable (PE) format applications. The software measurement schemes are described in detail in [7].

Memory protection. P-MAPS applies Intel VT hardware to virtualize OS page-table management. We have implemented OS-independent memory protection by forcing VMexit control events in order to be able to access control registers, invalidate page instruction usage, and page fault exception occurrences. We have designed a shadow page-table partitioning algorithm to gain access control to the application's memory in order to prevent it from being tampered with. Our shadow page-table partitioning approach is called virtualization-enabled integrity services (VIS) and is described in more detail in [8, 9].

The P-MAPS core manages two sets of page tables:

- *Active page table (APT).* This is the page table created and managed by the P-MAPS core in response to the creation and manipulation of the guest page table (owned and managed by the OS).
- *Protected page table (PPT).* This is the page table created and managed by the P-MAPS core in response to a registration by a software module running in the guest OS. In response to the registration, the software module is measured as described in the "P-MAPS Memory Services" section of this article, and a PPT is created for the software application such that the rest of the OS code (running via the APT mappings) cannot execute within the address space defined by the PPT. The setup of the PPT is shown in Figure 3. (The interaction between the APT and PPT for protecting a particular application during its execution is described later in the "P-MAPS Steady State: Application Protection" section of this article.)

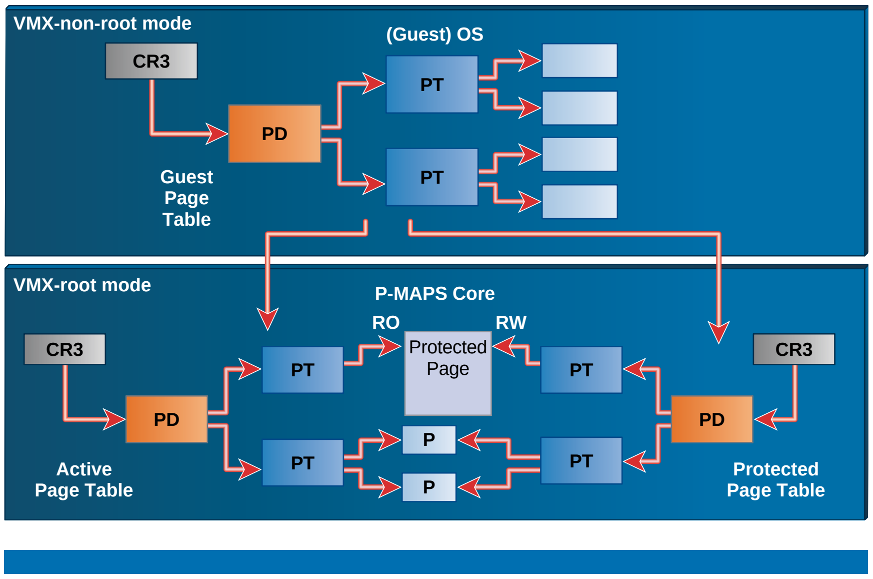


Figure 3: APT and PPT Managed by the P-MAPS core
Source: Intel Corporation, 2009

Memory eventing. The page-based access control system is used to report memory access events to a protected auditing agent that, in turn, may be used to apply policies to application memory accesses or to record events for audit log purposes.

P-MAPS Initialization and Launch

A high-level view of a trusted launch process is shown in Figure 4. Note that in that figure, on the left, the OS is first in host mode, that is, running natively. The OS is then temporarily quiesced when the P-MAPS core loader runs. Finally, the OS is in guest mode, and the applications interact with the P-MAPS core for protection. The pseudo code for the launch is described in detail in this section.

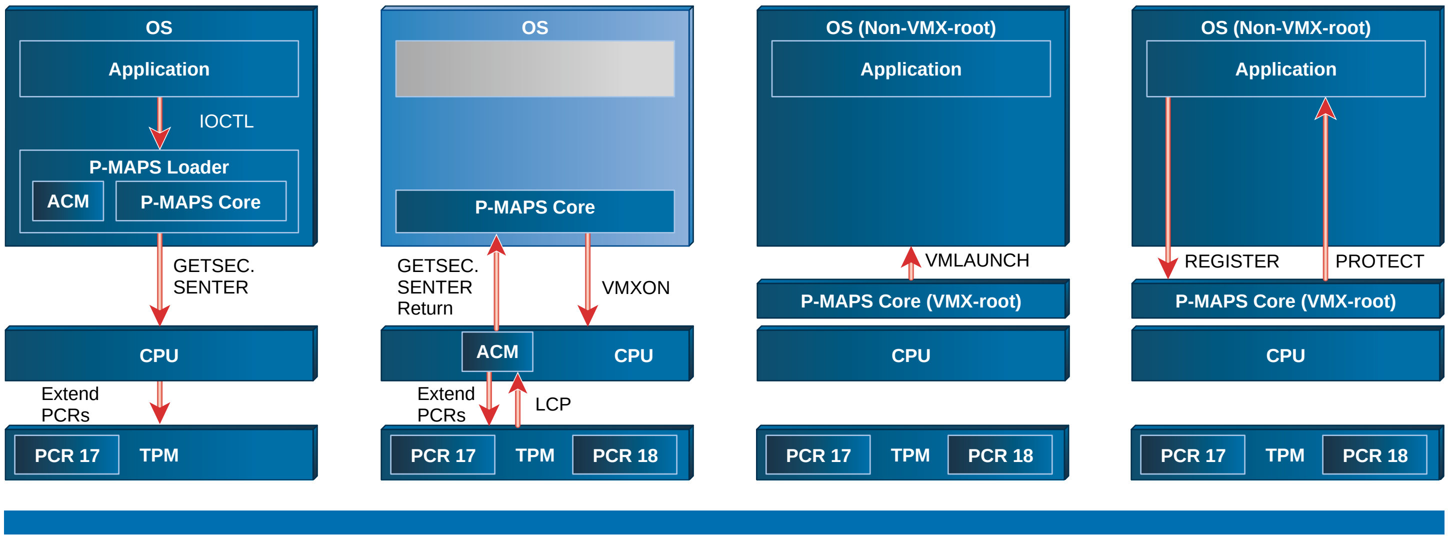


Figure 4: Trusted Launch Process of P-MAPS
Source: Intel Corporation, 2009

The top-level pseudo code for the P-MAPS core launch is shown in Table 1.

```
    //OS is in "host" mode
    //current mode of operation of this code is untrusted
1.  Disable Interrupts
2.  Save Segment Registers
3.  Save Stack Pointer
4.  Save all GPRs
5.  Save EFlags
6.  Launch P-MAPS (pseudo code for this step is described in detail in the next
    section)
    //Execution should resume at point 7 after launch with
    //OS in "guest" mode and Active and Protected Page
    //Tables managed by P-MAPS core.
7.  Restore EFlags
8.  Restore all GPRs
9.  Restore Stack Pointer
10. Restore Segment Registers
11. Restore Interrupts
```

Table 1: Top-level Pseudo Code for Launch of P-MAPS Core
Source: Intel Corporation, 2009

The P-MAPS loader loads the chipset SINIT ACM together with the P-MAPS core into memory, along with the supporting components. The P-MAPS loader also restores MTRRs that are saved in the os_mle_data (which is a data structure located in the TXT Heap). The os_mle_data used for P-MAPS core operation is shown in Table 2.

OS_MLE_DATA (Data used from OS by P-MAPS loader and core)
//os state saved (untrusted)
MTRR STATE
MSR STATE
OS CR3
OS STACK
OS RETURN VIRTUAL ADDRESS
OS RETURN PHYSICAL ADDRESS
OS GDT VIRTUAL ADDRESS
OS GDT PHYSICAL ADDRESS
OS TSS VIRTUAL ADDRESS
OS TSS PHYSICAL ADDRESS
//p-maps setup
POST-SENTER PAGE TABLE MEMORY (scrubbed before use)
P-MAPS CORE ENTRY PAGE PHYSICAL (measured code)
P-MAPS CORE ENTRY PAGE VIRTUAL (retained from OS)
P-MAPS GDT (measured data)
P-MAPS STACK PHYSICAL BASE (retained from OS)
P-MAPS STACK VIRTUAL BASE (retained from OS)
P-MAPS CORE PHYSICAL BASE (scrubbed before use)
P-MAPS CORE EXIT PAGE PHYSICAL (measured code)
P-MAPS CORE EXIT PAGE VIRTUAL (retained from OS)

Table 2: The os_mle_data Structure
Source: Intel Corporation

The memory allocated via OS services is not trusted. The P-MAPS loader allocates additional memory to stage the launch of the P-MAPS core. This includes memory for the following elements:

- *MLE page table.* Used by the processor to map the memory elements that will be measured by the GETSEC[SENTER] instruction.
- *MLE header.* Holds the P-MAPS code entry-point linear address (as interpreted by the MLE page table). After measurement of the P-MAPS core, the ACM transfers control into this entry-point, in protected non-paged mode.
- *Post-SENTER trampoline code and data.* This code is measured as part of the MLE and is responsible for switching to the measured global descriptor table (GDT), restoring the memory type range registers (MTRRs), and setting up the post-SENTER page table.
- *Post-SENTER page table.* The P-MAPS core is relocated in memory to execute from an identity memory map that is created via the post-SENTER page table. This page table is created by the measured relocator code. The mapped memory area is pre-allocated by the P-MAPS loader and is passed in the os_mle_data.

- *Post-SENTER GDT.* This GDT is used in the post-SENTER trampoline code. The GDT is prepared and measured in memory as part of the launch measurement performed by the processor.
- *Post-SENTER relocator code.* This (measured) code scrubs the memory into which it relocates the P-MAPS core. The base address of the P-MAPS core is passed to this relocator via the os_mle_data. This code library is pre-compiled at a well-known (static) virtual address base. The post-SENTER code that creates the post-SENTER page table maps this code at the well-known virtual address.
- *Un-relocated P-MAPS core.* This measured code is the P-MAPS core that is relocated and executed in VMX root mode to provide the application protection service.
- *Pre-allocated memory.* The memory for the P-MAPS core, the P-MAPS core-managed heap, and the P-MAPS stack are all pre-allocated and are cleared by the trusted P-MAPS loader before usage.

The P-MAPS core (MLE) memory layout is shown in Figure 5.

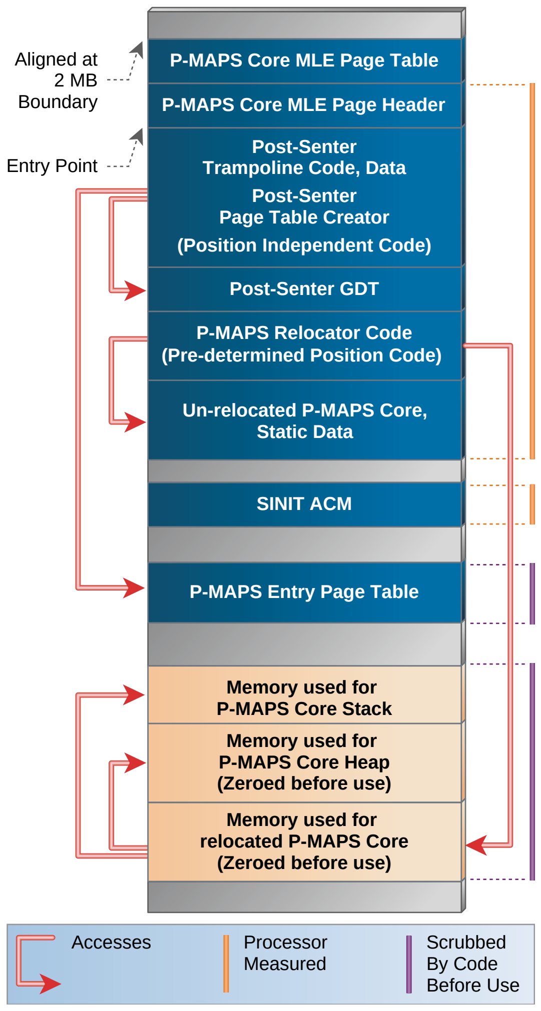

Aligned at 2 MB Boundary
Entry Point
P-MAPS Core MLE Page Table
P-MAPS Core MLE Page Header
Post-Senter Trampoline Code, Data
Post-Senter Page Table Creator
(Position Independent Code)
Post-Senter GDT
P-MAPS Relocator Code (Pre-determined Position Code)
Un-relocated P-MAPS Core, Static Data
SINIT ACM
P-MAPS Entry Page Table
Memory used for P-MAPS Core Stack
Memory used for P-MAPS Core Heap (Zeroed before use)
Memory used for relocated P-MAPS Core (Zeroed before use)
Accesses
Processor Measured
Scrubbed By Code Before Use

Figure 5: P-MAPS Physical Memory Layout
Source: Intel Corporation, 2009

The *Launch P-MAPS* step is described in more detail in Table 3. At launch time, the system is considered to be untrusted.

```
//current mode of operation is untrusted
1.  Allocate memory to stage P-MAPS for measurement.
2.  Load chipset SINIT ACM.
3.  Load P-MAPS (unrelocated) core binary image.
4.  Create MLE page table that maps part of P-MAPS loader and P-MAPS core that is to be measured and compared against platform
    launch control policy (LCP).
5.  Issue processor instruction GETSEC[SENTER].
//the above instruction causes the processor to verify the ACM, which then verifies the P-MAPS loader and unrelocated P-MAPS core
//against the LCP in the TPM
//control resumes at item 6 after GETSEC[SENTER] in protected non-paging mode following operations are trusted (that is, measured)
6.  P-MAPS loader loads measured GDT.
7.  Clear TXT error and status registers.
8.  Restore MTRRs from state saved in os_mle_data (located on the TXT Heap).
9.  Create post-SENTER page table that will be used to enter P-MAPS core. (Note: paging is not turned on yet). The mapping created
    in this page table is described in detail below.
10. Switch to post-SENTER page table.
11. Establish stack from (scrubbed) allocated memory.
12. Invoke relocator module to relocate measured P-MAPS core to scrubbed memory (allocated and passed via os_mle_data).
13. Push data needed for OS resume on stack. This includes the OS's original CR3, stack, and return EIP. These data are retrieved from
    the os_mle_data in the TXT Heap.
14. Push reference to P-MAPS handoff structure in memory on stack (P-MAPS handoff memory mapped in Step 9).
15. Invoke P-MAPS core entry. The P-MAPS core initialization is described in Figure 7(a).
//After Step 15, the P-MAPS core activates VMX and transitions the "host" OS into a "guest" configuration OS.
//Execution resumes at Step 16 (with any error information in GPRs).  If successful, the CR3 used by the guest references an active page
//table managed by the P-MAPS core.
16. Check GPRs for any error information.
17. If no error, restore OS resume data from stack.
18. Switch to OS guest CR3. Note that this action now causes a VMexit that is handled by the P-MAPS core that creates an active page
    table corresponding to the guest page table used by the OS. This APT ensures that the OS mapping cannot tamper with any of the
    P-MAPS core memory. The P-MAPS core memory includes the active and protected page tables.
19. Jump to the OS return EIP (virtual address mapped in guest page table, and therefore in active page table).
```

Table 3: Launch P-MAPS Pseudo Code
Source: Intel Corporation

The sequence of operations for the creation of the post-SENTER page table is shown in Table 4.

1.	Map (measured) entry trampoline pages as an identity and an OS-mirrored virtual address range.
2.	Map (measured) relocator module pages to static (well-known) virtual address.
3.	Map (measured) unrelocated P-MAPS core (identity mapped).
4.	Map (scrubbed) P-MAPS core stack pages (identity mapped).
5.	Map OS GDT and IDT. These are used only for creating the guest VMCS. The P-MAPS core uses its own (memory protected) GDT and IDT.
6.	Map OS TSS. These are used only for creating the guest VMCS.
7.	Map (measured) exit trampoline pages as an identity and an OS-mirrored virtual address range.
8.	Map (scrubbed) memory where P-MAPS core is relocated into (identity mapped).

Table 4: Create Post-SENTER Page
Source: Intel Corporation

P-MAPS Steady State: Application Protection

Once the P-MAPS core is in place (shown by the P-MAPS steady state in Figure 6), applications can register with it for protection. The registration interface is implemented via a parameterized VMCALL into the P-MAPS core, where VMCALL is an Intel VT instruction. The initial registration received by the P-MAPS core is untrusted. The P-MAPS core verifies the measurement of the runtime memory state of the application, based on the integrity manifest provided by the application. Once the application memory passes the measurement checks, the P-MAPS core creates a PPT for the application. All page tables managed by the P-MAPS core are in the P-MAPS heap, which is allowed to be mapped in any APT or PPT created for OS execution.

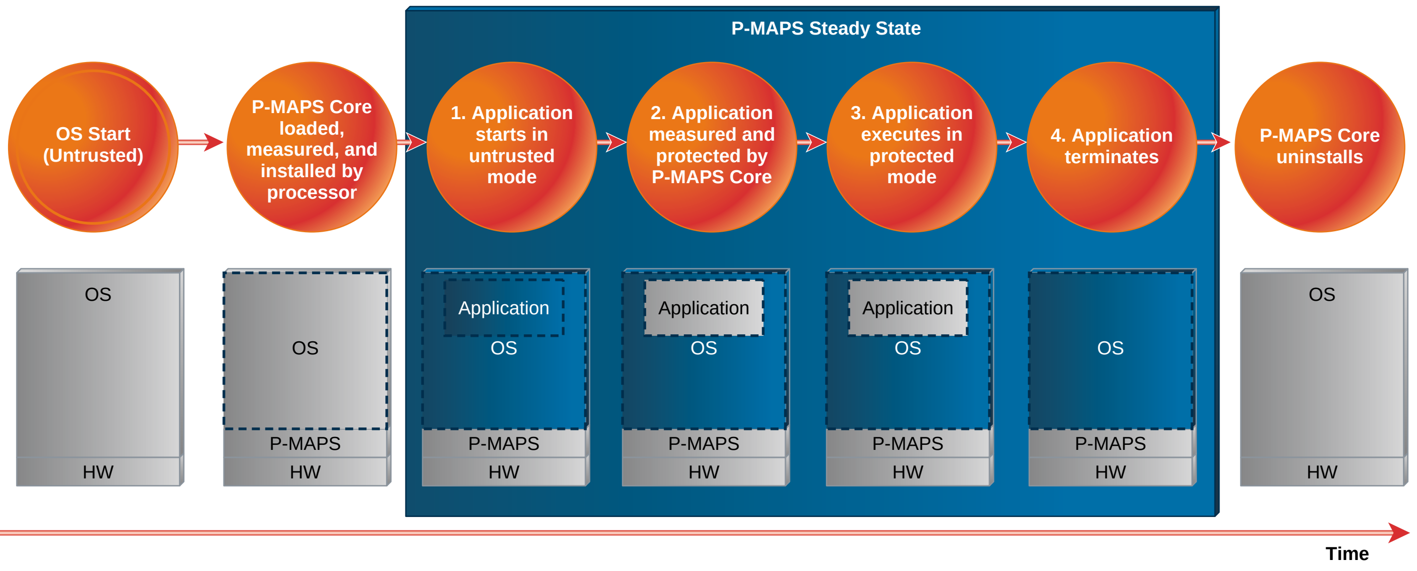

P-MAPS Steady State
OS Start (Untrusted)
P-MAPS Core loaded, measured, and installed by processor
1. Application starts in untrusted mode
2. Application measured and protected by P-MAPS Core
3. Application executes in protected mode
4. Application terminates
P-MAPS Core uninstalls
OS
HW
P-MAPS
Application
Time

Figure 6: Application Usage of P-MAPS
Source: Intel Corporation, 2009

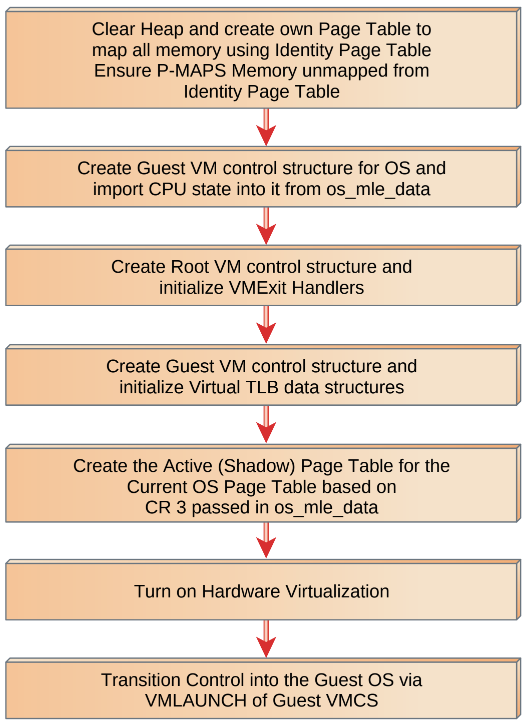

Clear Heap and create own Page Table to map all memory using Identity Page Table Ensure P-MAPS Memory unmapped from Identity Page Table
Create Guest VM control structure for OS and import CPU state into it from os_mle_data
Create Root VM control structure and initialize VMExit Handlers
Create Guest VM control structure and initialize Virtual TLB data structures
Create the Active (Shadow) Page Table for the Current OS Page Table based on CR 3 passed in os_mle_data
Turn on Hardware Virtualization
Transition Control into the Guest OS via VMLAUNCH of Guest VMCS

Figure 7(a): P-MAPS Core Initialization
Source: Intel Corporation, 2009

The pages for the application that are successfully measured are isolated by the P-MAPS core into a PPT. The protected application may allocate new memory that can be inserted by the P-MAPS core into the PPT after scrubbing. When the protected application is executed, it is not necessary to mask interrupts or interfere with OS operation of other unprotected or unknown applications. If an interrupt occurs, the OS interrupt service routine execution causes the execution to fault into the P-MAPS core; the P-MAPS core verifies if a protected application was executing (via a PPT), and if so, transfers control to the active page table to let the (unprotected) OS interrupt service routine complete. Additionally, the P-MAPS core records the interrupt point of the application so that it can verify that it is being resumed from the correct point.

Further, paging of the application pages is not affected; any access to P-MAPS protected pages from the OS is considered equivalent to an attack, so the affected pages are subjected to an integrity check (in the P-MAPS fault routine), and they are un-linked from the PPT. When the page is swapped back in, and code from the protected code page is executed, the fault is internal to the PPT, and the P-MAPS core verifies the integrity check value on the page contents before linking the page to the PPT. This allows the OS operation to continue unhindered but does not affect the security of the protected application.

The P-MAPS core allows the following policies to be enforced for a protected application:

- Code pages cannot be written.
- Code or data pages may be entirely hidden.
- Data pages may be read/write or hidden.
- Specific data pages may be shared between trusted and untrusted code.
- The code page can be executed only from specific entry points.

The events handled by the P-MAPS core for memory management of the protected application are best shown in flowcharts shown in Figure 7(a) and 7(b).

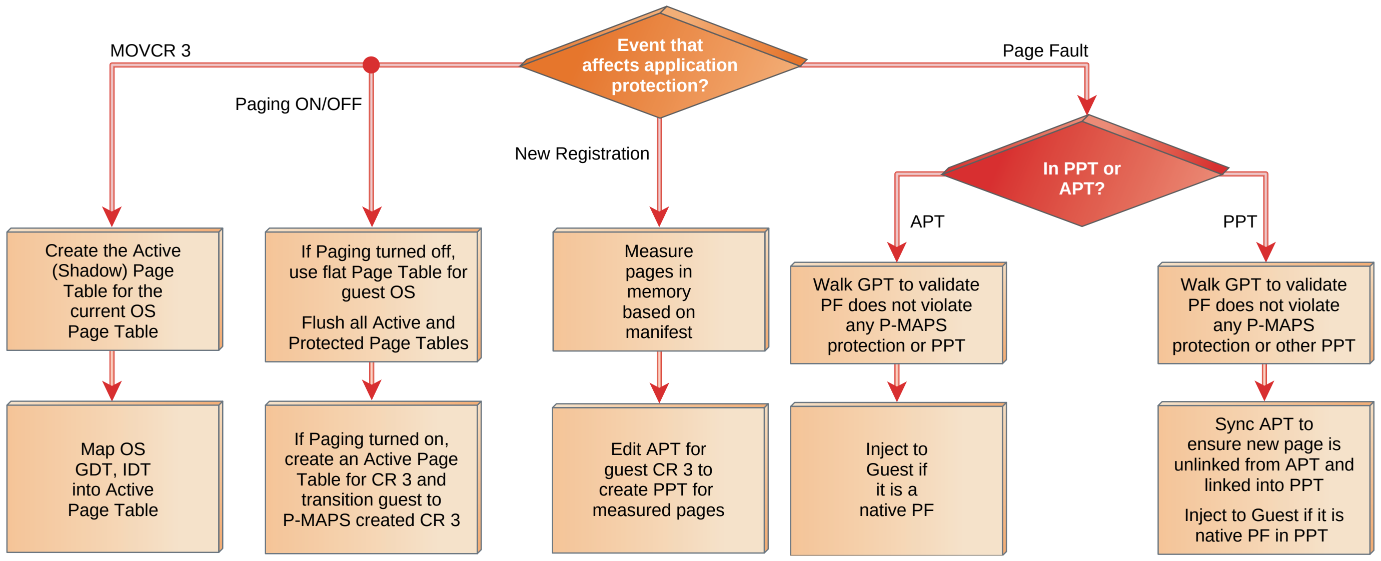

Event that affects application protection?
MOVCR 3
Paging ON/OFF
New Registration
Page Fault
In PPT or APT?
APT
PPT
Create the Active (Shadow) Page Table for the current OS Page Table
Map OS GDT, IDT into Active Page Table
If Paging turned off, use flat Page Table for guest OS
Flush all Active and Protected Page Tables
If Paging turned on, create an Active Page Table for CR 3 and transition guest to P-MAPS created CR 3
Measure pages in memory based on manifest
Edit APT for guest CR 3 to create PPT for measured pages
Walk GPT to validate PF does not violate any P-MAPS protection or PPT
Inject to Guest if it is a native PF
Walk GPT to validate PF does not violate any P-MAPS protection or other PPT
Sync APT to ensure new page is unlinked from APT and linked into PPT
Inject to Guest if it is native PF in PPT

Figure 7(b): P-MAPS Core Memory Management Events
Source: Intel Corporation, 2009

P-MAPS Teardown

The P-MAPS teardown is achieved as shown in Figure 8. Before issuing a VMXOFF VT instruction that exits VM root mode, the P-MAPS core ensures that there are no more applications being protected by P-MAPS and that the teardown request arrived from the protected service that launched the P-MAPS core.

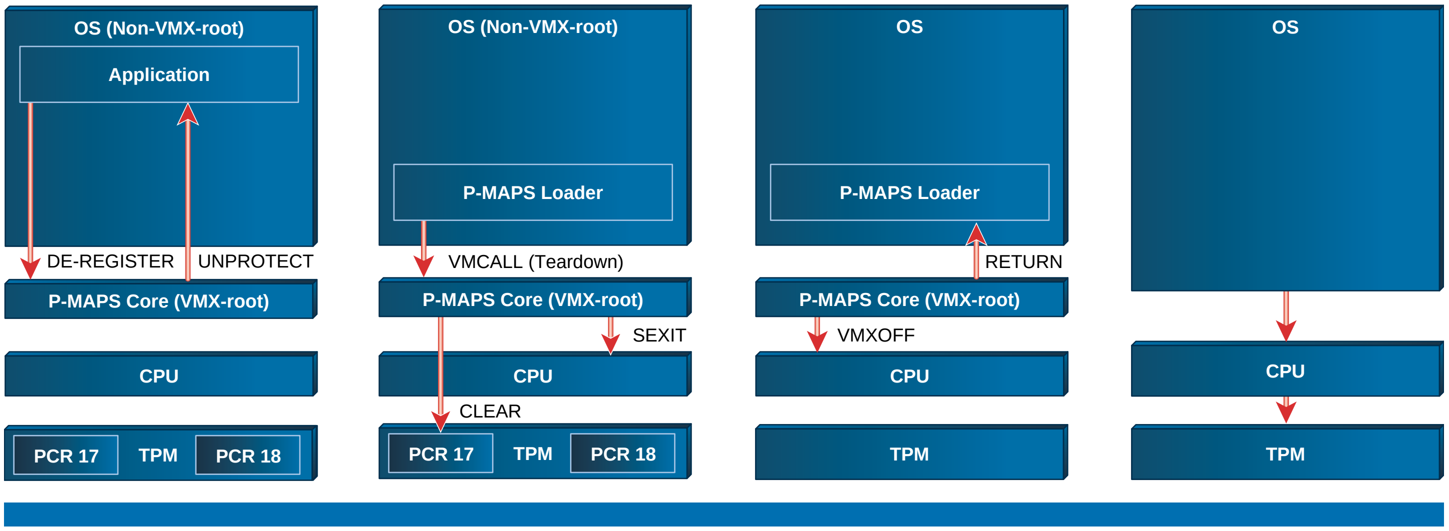


Figure 8: P-MAPS Teardown
Source: Intel Corporation, 2009

If these conditions are satisfied, the P-MAPS core scrubs any secrets that were held in protected memory, caps TPM PCRs, issues a VMXOFF to relinquish Intel VT hardware control, and issues a GETSEC[SEXIT] to exit trusted mode (to allow a subsequent measured launch to take place). The P-MAPS core transfers control back into the untrusted portion of the P-MAPS loader, which in turn de-allocates the P-MAPS memory (if required). The P-MAPS loader may also keep the memory allocated until system shutdown to allow subsequent launches, if such an action is required. As shown in Figure 8, the OS resumes in *host* mode after P-MAPS teardown.

Remote Attestation

A protected application typically involves the handling of secret data that are provisioned by an entity (provisioning server) in the network. The protected application must assure the remote entity that the application is indeed executing in the specified protected environment before receiving the secret data. A set of trusted entities participate to enable this mechanism.

"A protected application typically involves the handling of secret data that are provisioned by an entity in the network."

Trusted Entities and Their Roles

Here are some of the trusted entities and their roles.

- *Trusted platform module (TPM) and its owner (e.g., an end user or an IT administrator).* The owner sets the TPM authentication password and is responsible for password protection.

- *Endorsement certificate authority (CA).* The TPM device is provisioned with the endorsement key (EK) and an EK certificate from the endorsement CA at manufacture and ship time. The certificate provides attestation for the TPM manufacturer, signed by a TTP, such as VeriSign*.
- *Privacy CA server.* This is a TTP used by the provisioning server to verify the EK certificate from a TPM with an assurance of keeping the identity of the TPM host confidential.
- *Intel TXT components (CPU/Chipset, ACM).* The ACM works in concert with the CPU and chipset to verify hardware conformance; for example, it verifies that the TPM being used is physically attached to the platform. The ACM also extends the TPM PCR registers to record the measurement of the P-MAPS core—this property is used during the operation of the P-MAPS core to associate application credentials to the local TPM.
- P-MAPS core. The core enforces protection via page table changes. The P-MAPS core uses TPM to generate attestation identity keys (AIKs). These keys are used to sign (appropriately tagged) application-specific data and to sign the TPM's current PCR values (TPM_Quote). The provisioning server verifies the TPM quote (based on PCRs and locality) to ensure the platform has the necessary software posture before sharing confidential data.

As part of the Intel TXT dynamic launch, PCR 17 is updated with the identity of the ACM, and the P-MAPS core measurement is recorded in PCR 18. When the P-MAPS core is launched, it protects (virtualizes) TPM access and denies host OS access to TPM at locality 2. The P-MAPS core requests the TPM to generate an AIK pair and to associate this AIK with PCRs 17 and 18 and locality 2. It provides the TPM's EK certificate to the privacy CA and requests a certificate for this AIK. When the P-MAPS core needs to attest its state to a remote server it provides a TPM quote signed by the AIK and includes values of PCRs 17 and 18.

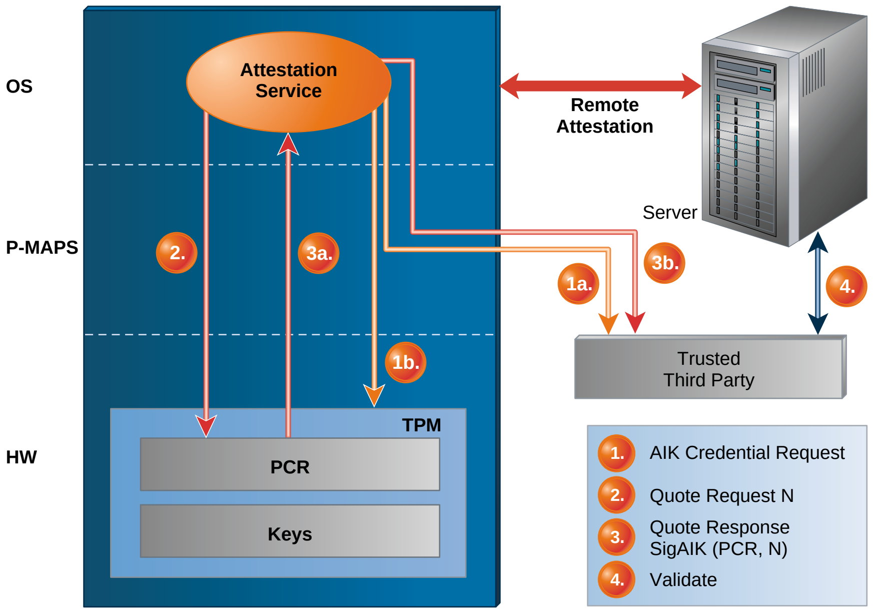


Figure 9: Remote Attestation of Protected Applications
Source: Intel Corporation, 2009

The remote server can use the privacy CA to verify the AIK. The AIK can be used by the P-MAPS core to send the public portion of an RSA key pair. The above mechanism follows a standard protocol recommended by the TCG. The remote server can use the public key to encrypt a secret before sending it to the P-MAPS core for provisioning. This interaction is illustrated in Figure 9.

Seal and Unseal Secrets

Once provisioning is complete, an application may need to store a secret (which may be a key) that is subsequently required during steady-state operation. The application sends the secret to the P-MAPS core for protection, and the core uses the TPM to seal the secret to PCRs 17 and 18 and locality 2. The encrypted secret is given back to the application to store as it pleases. When the secret is needed, the application requests the P-MAPS core to unseal the secret and deposit it into protected memory.

Usages

The P-MAPS core can be used to protect critical applications. Applications are deemed critical, either from a user-data perspective or from a security perspective: for example, banking applications, security software, such as anti-virus or rootkit prevention, are all critical applications. Additionally, P-MAPS can be used to extend hardware services to integrity-verified drivers thus creating protected hardware extensions in software.

Performance Evaluation

We implemented P-MAPS on an Intel mobile platform enabled with Intel VT and Intel TXT. Our Intel TXT loader is written for Windows* XP* and is based on the Trusted Boot Project [10]. The platform hardware configuration, previously codenamed Montevina [11], consists of an Intel® GM45 Express Chipset, an Intel® Core™2 Duo Processor P8600 (3M Cache, 2.40 GHz, 1066 MHz FSB), 2GB RAM, and an Infineon* TPM [12]. We measured the time required to launch the P-MAPS core, via a Windows XP kernel service, to be 300 msec on average. This includes the time taken from the GETSEC[SENTER] instruction to the instruction run after control comes back into the OS-specific launcher (from the measured P-MAPS core). A large portion of the time is spent in interaction with the TPM over the serial LPC bus, and in reconfiguring the MTRRs. Table 5 breaks out the time spent in the different activities that occur during the launch and teardown processes.

Launch: from GETSEC[SENTER] to resume	**300 msec**
GETSEC[SENTER]: ACM verification, execution (entry to trampoline)	
Trampoline: execution (entry to P-MAPS core)	
P-MAPS core: setup, guest creation and resume	
Tear Down: from VMCALL to resume	**0.54 msec**

Table 5: Initialization and Teardown for P-MAPS Core
Source: Intel Corporation, 2009

For further details on Intel TXT, the reader is referred to the Intel technical reference book for Intel TXT [13].

Conclusion

"We describe how this system can be used to provide protection without interfering with the typical scheduling and operation of the OS, including unprotected applications."

We have demonstrated via a research proof-of-concept how Intel TXT and Intel VT hardware can be used to reduce the TCB of current PC systems, on-demand (dynamically), from the full OS software to a substantially smaller P-MAPS core module that provides runtime protection for applications. We describe how this system can be used to provide protection without interfering with the typical scheduling and operation of the OS, including unprotected applications. We can use this application protection mechanism to make a whitelist of critical applications and thus mitigate 0-day software attacks on these protected applications. We continue to analyze different applications of the P-MAPS core.

References

[1] R. S. Cox et al. "A Safety-Oriented Platform for Web Applications." In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. 2006.

[2] Source lines of code. At ***http://en.wikipedia.org***

[3] "Intel® Virtualization Technology for Directed I/O." At ***http://download.intel.com***

[4] TPM Specification, Version 1.2. At ***http://www.trustedcomputinggroup.org***

[5] *Intel® Trusted Execution Technology—Measured Launched Environment Developer's Guide*. At ***http://www.intel.com***

[6] *Intel® 64 and IA-32 Architectures Software Developer's Manual*. At ***http://www.intel.com***

[7] "OS Independent Run-Time System Integrity Services." *IT Innovation and Research*, November 2005, Intel Corporation. At ***http://blogs.intel.com***

[8] Ravi Sahita et al. "Mitigating the lying endpoint problem in network access control frameworks." *IEEE/IFIP DSOM*, 2007. At ***http://www.springerlink.com***

[9] Ravi Sahita et al. "Towards a Virtualization-based Framework for Information Traceability." *Advances in Information Security—Insider Attack and Cyber Security* ISBN 978-0-387-77321-6. At ***http://www.springerlink.com***

[10] Joseph Cihula et al. "Trusted Boot project on sourceforge.net." At ***http://sourceforge.net***

[11] Intel Montevina Platform. At ***http://ark.intel.com***

[12] Infineon Trusted Platform Module. At ***http://www.infineon.com***

[13] David Grawrock. "The Intel Safer Computing Initiative." *ISBN-10: 0976483262*. At ***http://www.intel.com***

Acknowledgments

We thank our colleagues who contributed to various facets of this research project and the proof-of-concept development: David Durham, Joseph Cihula, Andy Anderson, Michael Kinney, Ranjit Narjala, and Ansuya Negi.

Author Biographies

Ravi Sahita is a Senior Researcher at Intel Labs. His research interests are software and systems security, network security, and distributed systems. Ravi has contributed to Intel NetStructure® products, the open-source Intel Common Open Policy Services (COPS) client SDK, and Intel® AMT System Defense Manager. Ravi is a contributing member of the Internet Engineering Task Force (IETF) and the Trusted Computing Group (TCG) standards bodies. He received his B.E. degree in Computer Engineering from the University of Bombay and an M.S. degree in Computer Science from Iowa State University. His e-mail is ravi.sahita at intel.com.

Ulhas Warrier is a Senior Architect in Intel's Mobile Platforms Group, working on architecture definition for next-generation mobile platforms. He works in the areas of platform security, manageability, virtualization, and architectural support for services. Ulhas has contributed to in-circuit emulator products, Intel® ProShare® related products, platform partitioning, and consumer networking projects. He was the chair of the Internet Gateway Working Group in the Universal Plug and Play (UPnP) Forum. He received his M.S. degree in Computer Science from Oregon State University. His e-mail is ulhas.warrier at intel.com.

Prashant Dewan is a Security Research Scientist at Intel Labs. His research interests are network and platform security, virtualization, and decentralized networks. He has a Ph.D. degree in Computer Science from Arizona State University and has been working at Intel since 2004. His e-mail is prashant.dewan at intel.com.

Copyright

PROVIDING A SAFE EXECUTION ENVIRONMENT

Contributors

Kirk Brannock
Intel Corporation

Prashant Dewan
Intel Corporation

Frank McKeen
Intel Corporation

Uday Savagaonkar
Intel Corporation

Index Words

Attestation
Security
Software
Integrity
Virtualization

"It is extremely important that an SEZ provide an execution environment that has a small and manageable TCB."

Abstract

Providing a safe execution environment for an application requires meeting a number of complex requirements.

In today's connected computer environment, it is necessary to provide a trusted environment to a wide variety of applications. This article describes research on creating a secure execution zone (SEZ) for executing software agents in a secure manner on a computer system. Security in this case includes integrity, confidentiality, access control, as well as other domain-specific requirements, all of which are described in this article. We also discuss the creation of an SEZ and outline some examples.

Introduction

The purpose of a secure execution zone (SEZ) on a computing platform is to provide a place where software can execute as intended without being effected by malicious external agents. Providing such isolation on an open platform, such as an x86-64 platform, is a challenging task. A typical x86-64 system consists of multiple hardware, firmware, and software components, a large number of which are capable of altering the computational outcome of software executing on that platform. The trusted computing base (TCB) is "a small amount of software and hardware that security depends on and that we distinguish from a much larger amount that can misbehave without affecting security" [1]. Any vulnerability in the TCB of a software component can potentially be exploited by an attacker to alter the behavior of the software in an unexpected fashion. Consequently, it is extremely important that an SEZ provide an execution environment that has a small and manageable TCB.

Organization of this Article

We first provide an overview of the problems associated with an SEZ. We then describe the current computer ecosystem and the need for an SEZ. We further explain the issues with the current ecosystem and continue with a description of the requirements of an SEZ. We then look at some of the current solutions and discuss the advantages and disadvantages of those, and we end with a description of an ideal SEZ.

Ecosystem Description

The Internet and the connected environment have brought substantial changes to the nature of applications. Applications are evolving to bring valuable data to the client platform. Applications can be distributed across many platforms. Each platform performs a particular task of the application. The application interacts with the user of a computer and other applications. Applications today can dynamically download information, and information can be forwarded from one computer to another without any user intervention. This allows applications to take advantage of remote and local computational resources.

"Applications today can dynamically download information, and information can be forwarded from one computer to another without any user intervention."

Many applications contain intellectual property (IP) that may be valuable to outside parties or even the owner of the machine (for example, getting a look at an earnings report before it is released). In some cases this IP is not sold to the computer owner but rather rented (for example, when you rent a movie). When IP is rented, the computer user does not own the rights to it, but rather is provided access for some defined period.

In addition, database applications provide the ability to distribute records to many machines. These databases store millions of records, such as financial and medical, as well as other valuable and important records, and keeping track of this information has proved to be problematic. There are many reports of lost laptops containing thousands of records.

There are also numerous reports of servers being overrun with malware that removes many files that contain personal information.

Ecosystem Issues

The computer ecosystem today is dominated by open platforms. These platforms are constructed with a myriad of software components from different vendors, and all of the components vary in quality. A computer may include components from as many as a dozen manufacturers. Many of these components use privileged drivers that are inside the TCB of the application; they have access to the application's data.

"Database applications provide the ability to distribute records to many machines."

Computer platforms contain hundreds of settings and parameters that affect the security and integrity of the platform. It takes a panel of experts to understand how to secure such a platform, given the implications of all the various settings; yet, for the most part, platform users are untrained in software or security. The result is that platforms are often mis-configured with respect to security protection.

Moreover, the current ecosystem encourages the download of both code and data. There is no way for users to tell if the files they download contain only the code they expect. For instance, many users are deceived by a download that purports to provide a service but at the same time deposits other code in their system—code that is in effect malware. Today's platforms allow remote download of code that executes at privileged levels. A user can accidentally download code that will alter the operating system (OS) configuration.

All of these factors make machines vulnerable to attacks that can result in the release of confidential data.

"A computer may include components from as many as a dozen manufacturers."

Why a Secure Execution Zone is Needed

As the Internet has evolved, innovation has resulted in new applications that require the ability to securely store data and protect them from unauthorized usage and tampering. The value of data has risen in recent years. For example, the theft of files containing personal information leads to identity theft. Data are also a company's or a manufacturer's intellectual property (IP); these data contain trade secrets of the application. Data in applications, such as those shown in Table 1, require protection. Many applications have legal requirements to protect data. For instance, medical records must meet protection standards imposed by the Health Insurance Portability and Accountability Act (HIPPA). The Sarbanes-Oxley Act of 2002 imposes protection of corporate financial data to prevent insider trading and the compromise of a company's integrity.

Application Category	Protected Items
Premium content	Content provided by physical or digital distribution
Medical records	Patient information, HIPPA compliance
E-commerce	Credit card information
Device authorization	Theft protection, computer leasing
Military applications	Targets, threat locations, resource location
Enterprise digital rights management	Document protection and control
Network subscriptions	Network access, 3G, WiMAX
Network keys	Protocol protection
Context-aware applications	Targeted ads, user location

Table 1: Applications That Require Data Protection
Source: Intel Corporation, 2009

In all cases, the application or data owner requires that core portions of the application (i.e., the code and data) be kept secret and the data not be compromised.

Secure Execution Zone Requirements

In this section we delve into the requirements and properties of an SEZ. We first discuss how the security properties of an SEZ depend on the types of attack; we look at the required properties of an SEZ as well as the properties that make an SEZ more effective. We also touch on SEZ technology and how it works with mutually distrusting applications. We end the section by evaluating a system management mode (SMM) and a virtual machine monitor (VMM) as candidates for an SEZ.

Attack Models

The security properties desired of an SEZ heavily depend on the attack model. In the least severe of all the attack models, the adversary only has access to the platform via a network port. In such a model, the process separation provided by the OS typically provides a sufficiently secure place to execute, and no special SEZ is required. It should, however, be noted that such models typically have very large TCBs, and any vulnerability in any component inside the TCB can compromise the software trust model. The insufficiency of such an attack model is amply demonstrated by frequent security advisories affecting the major operating systems; consequently, system architects must consider more advanced attack models.

"In the least severe of all the attack models, the adversary only has access to the platform via a network port."

The second attack model assumes that the attacker has compromised the software stack on the platform and has access to the most privileged ring available to commercial software. However, the attacker does not have any direct access to the hardware. In such situations, hardware-based access control (for example, range-register-based access control and paging-structure-based access control) can provide a sufficiently trusted SEZ. The hardware, firmware, or software inside the TCB can cryptographically protect code and data as they leave the access-controlled environment. However, no special (cryptographic or otherwise) protection is needed on code or data living inside the access-controlled environment, while they travel on the system buses.

The most severe attack model assumes that the attacker can not only compromise the software stack, but also has some level of physical access to the platform hardware and is capable of launching simple hardware attacks, such as DIMM-removal and/or snooping the buses. This attack model is applicable in situations where the computer platform is stolen, or where the platform user might be interested in circumventing the security of the system (for example, to circumvent digital rights management (DRM) protections). This kind of attack represents one of the most challenging security problems and requires cryptographic protections on internal buses, in addition to various access-control mechanisms.

Cloning and Replay Attacks

A special kind of attack model is the cloning and replay attack. Such an attack involves a corrupted software stack that attacks an in-band SEZ. The attack involves recording the state of the SEZ and restarting the SEZ repeatedly with the same state. An attacker can change the input data to the SEZ and record the output of the SEZ. We use the term replay attack to indicate that a partial portion of the SEZ is being replayed. A cloning attack refers to attacks that replay the entire SEZ space.

"The cloning and replay attack involves a corrupted software stack that attacks an in-band SEZ."

A cloning or replay attack is more easily mounted against an in-band SEZ, where

"Code integrity is an absolute requirement of any SEZ environment."

the SEZ is dependent on the software stack to provide resources. In order to prevent this sort of attack the SEZ must have special protections against replay. Replay and cloning attacks can be mounted remotely, if the software stack contains an exploitable vulnerability.

Required Properties of an Effective Secure Execution Zone

An SEZ environment has to have many properties to make it effective. We describe in detail these properties in this section.

Code Integrity

Code integrity refers to the ability of an SEZ environment to prevent the software running inside an SEZ container from being modified by entities outside the TCB of that container. Code integrity is an absolute requirement of any SEZ environment—any SEZ environment must protect software code running inside a container from malicious tampering by an attacker outside the container, under the applicable attack model.

Control-flow Integrity

"The SEZ must enforce controlled entry points into the protected environment."

Just as code integrity is an important property of an SEZ environment, so is control flow integrity. Therefore, any SEZ environment must provide this property under the applicable attack model. It should be noted that, in an x86-64 programming environment, protecting the binary code image of the software component from malicious modification is not sufficient—there are a number of other factors that can affect the outcome of the execution. For example, an x86-64 execution environment does not enforce any alignment restrictions on executable instructions. Consequently, jumping into the same binary page at an offset different from the intended offset can lead to a completely different set of instructions being executed. Additionally, x86-64 platforms allow multiple levels of address indirections, including segments, page tables, extended page tables (EPTs), and QPI-based routing/decoding. Typically, these address indirections are controlled by more than one firmware and software entity. If the attacker is allowed to control any of these levels of translation in an unrestricted fashion, then the attacker can deterministically change the outcome of the software execution. For example, the attacker can jump into unspecified offsets (by modifying the segment base), or the attacker can completely change the execution order (by changing page tables or EPTs). Thus, the SEZ must enforce controlled entry points into the protected environment. Additionally, the SEZ must protect software executing inside it by either disabling the translation, controlling the translation, checking the final translation, or any combination of these.

Data Integrity

Data integrity, another required property of an SEZ environment, refers to the ability of the SEZ environment to prevent modifications of the static and dynamic data belonging to the SEZ container from entities outside the container. Every SEZ environment must be able to protect the data belonging to an SEZ container from malicious tampering by an attacker outside the container, under the applicable attack model. It should be noted however, that under some usage models, SEZ containers need to ensure data integrity on a portion of their data only—the attacker may be allowed to modify the remainder of the data belonging to the SEZ container. For example, in the case of integrated graphics devices, the attacker is allowed to modify the intermediate-surface data for protected surfaces that belong to the graphics device, without affecting the trust properties, such as the high-definition digital content protection (HDCP), of the graphics device.

"Data integrity refers to the ability of the SEZ environment to prevent modifications of the static and dynamic data belonging to the SEZ container."

Data-flow Integrity

Data-flow integrity refers to the property that stipulates that the data belonging to an SEZ container always serve their intended purpose within that container. For example, most of the executable programs developed in native runtime environments such as C/C++/assembly associate their data with the logical address of the data. However, as explained earlier, the logical addresses go through multiple layers of address translation and redirection before being consumed by the hardware. Consequently, if an attacker controls any of the intervening translation layers, it can point the logical address of one data block to a completely different physical data block belonging to the same container, causing the data to be used in an unintended fashion. For example, consider a small web server running inside an SEZ container that maintains lists of hosts that are allowed and disallowed to establish a connection. If the attacker can modify the logical-to-physical address translations without any oversight from the SEZ, then the attacker might be able to persuade the web server to interpret the list of disallowed hosts as the list of allowed hosts, thereby circumventing the intended purpose of these lists. An SEZ environment may be required to provide a binding between the logical address and the actual data. The binding can either be provided cryptographically or by disabling, controlling, or checking the translation (or a combination of all three).

"If an attacker controls any of the intervening translation layers, it can point the logical address of one data block to a completely different physical data block."

Semantic Data Integrity

"The SEZ permits substitution within a semantically equivalent set."

Semantic data integrity is a weaker property than data-flow integrity. In data-flow integrity, data are tied to their addresses within the SEZ container and can only be accessed by that container at those addresses. In semantic data integrity, the data are grouped into semantically equivalent sets. The SEZ permits substitution within a semantically equivalent set. However, substitution across semantically different sets is not permitted. For example, on the integrated graphics device, the intermediate protected surfaces generated by the graphics device that belong to the same application context are semantically equivalent, and an attacker can substitute one surface for another. However, protected surfaces belonging to different application contexts cannot be substituted for one another. It should be noted that the substitution of semantically equivalent surfaces does not compromise the graphics trust model; although such substitution may result in the graphics device displaying garbage on the screen. For many protection models, semantic data integrity is sufficient and can be implemented at a much lower cost than full data-flow integrity.

Data Confidentiality

"If the attack model allows the attacker to have physical access to the platform, then the SEZ may require hardware-based encryption on the platform buses."

Data confidentiality refers to the ability of an SEZ to prevent attackers from accessing the designated data in the clear. The measures required to protect the data depend on the level of access the attacker has to the system. Typically, if the attacker does not have physical access to the system, then data confidentiality can be guaranteed via access control at the hardware level, and via software and firmware managed encryption when the data are moved out of the access-controlled region. However, if the attack model allows the attacker to have physical access to the platform, then the SEZ may require hardware-based encryption on the platform buses.

Code Confidentiality

Code confidentiality is where the attacker is denied access to the executing code. Again, the exact mechanism depends on the level of access the attacker has. In simple situations, mode-based access control might be sufficient. In more involved cases, SRAM-level protection, or memory encryption, might be necessary. Code confidentiality might be desired when the code contains intellectual property, such as a trade-secret algorithm or copyrighted software.

Attestation

"Attestation is used for establishing trust with remote entities after the platform has been shipped to the user."

Attestation refers to the ability of the software running inside an SEZ container to prove to external entities that it is in fact running inside the SEZ container. This property is extremely important for most SEZ environments, and attestation is used for establishing trust with remote entities after the platform has been shipped to the user. Without an attestation framework, the SEZ cannot provide provable protection to software that was not provisioned at the time the platform was shipped to the user.

Resistance to Denial of Service

Some platform services require guaranteed platform resources such as memory, CPU cycles, network access, and so on. Without these resources, the services could be starved and therefore unable to perform their basic intended functions. Examples of such services include anti-virus or malware detection and battery management services, among others. Such services need an SEZ that can either protect the platform services from denial of service attacks or detect that a denial of service attack is occurring and signal an external agent.

"Some platform services require guaranteed platform resources."

Mutually Suspicious Applications

The same SEZ technology could be used to create multiple simultaneous instances of an SEZ environment running on the same platform. Such instances are typically mutually distrusting, and the SEZ should assure the isolation between the different instances. Each instance of an SEZ is called an SEZ container.

Other Properties

As well as required properties, an SEZ will be much more effective if it also has these properties:

- Friendly to operating environments
- Scalable threading, expandable memory, scalable CPU resources
- Access to system resources such as network stack, display, system services

System Management Mode as a Secure Execution Zone

We now examine system management mode (SMM) and its viability as an SEZ.

"When memory cycles reach the memory controller they can be distinguished between SMM and non-SMM operation of the processor."

Background

The SMM of the CPU has a number of attributes that make it an interesting candidate for an SEZ. When running in a properly configured platform, SMM code enjoys isolation from the host OS and from DMA agents in the platform. Furthermore, SMM code has access to all host-accessible hardware resources in the machine. Figure 1 shows an architectural layout of SMM with respect to the rest of the system software.

SMM is entered via a system management mode interrupt (SMI), which can be generated by a platform's chipset hardware or by the CPU itself. The SMI is serviced by the SMI handler, which is typically the exclusive domain of the BIOS and is configured early in pre-boot BIOS execution. The associated configuration bits are normally locked to prevent non-BIOS code from changing the configuration once it is set. The SMI handler executes in a region of sequestered memory known as SMRAM. All memory cycles from the CPU are tagged in a manner to indicate whether the cycle originated from code running in SMM or not. Therefore, when memory cycles reach the memory controller they can be distinguished between SMM and non-SMM operation of the processor. The memory controller will route cycles to SMRAM, only if they originate from a CPU that is running in SMM. Additionally, all DMA cycles to SMRAM are denied.

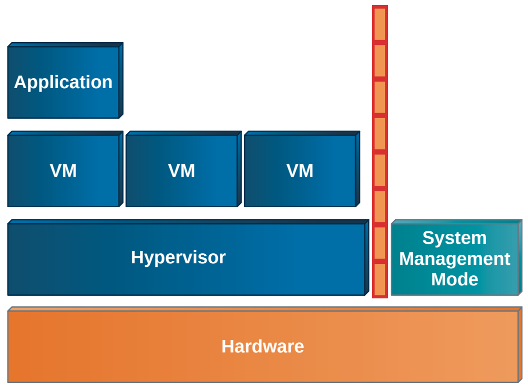


Figure 1: System Management Mode Diagram
Source: Intel Corporation, 2009

"The SMI handler is resistant to tampering from ring 0 host software and seems like a natural place to implement SEZ applications."

SMM is typically used to provide platform-specific runtime services (for example, enabling ACPI mode during OS boot), to implement BIOS workarounds for hardware issues, and in some cases to emulate hardware (for example, PS/2 mouse and keyboard emulation for USB devices).

System Management Mode as an SEZ Host

Hardware locks prevent software from changing configuration in a way that would expose SMRAM or inhibit SMIs from occurring. Therefore, if implemented properly, the SMI handler is resistant to tampering from ring 0 host software and seems like a natural place to implement SEZ applications.

All SEZ applications have a confidentiality requirement (protection of some secret) and an integrity requirement (tamper resistance); otherwise, they would simply be written in the context of a normal OS. Furthermore, assurance or attestation of the environment is commonly required. Any candidate SEZ environment must be evaluated based on its ability to provide these properties.

Upon careful analysis, with these requirements in mind, there are a number of issues with SMM as an SEZ host:

- The SMI handler is, by definition, platform specific. Each platform may have several BIOS revisions, and each BIOS revision may change the SMI handler. Therefore, the total number of SMI implementations is very large. Gaining any reasonable level of assurance that an arbitrary SMM implementation provides sufficient confidentiality or is tamper resistant is problematic, if not intractable.
- Secret protection and assurance of secret destruction is incomplete, even with a correctly implemented SMI handler. While it may be possible to devise schemes that can defend secrets from a software attack with BIOS inside the SEZ TCB, these schemes are clearly insufficient for some classes of reasonably trivial physical attacks that are a concern for many SEZ applications.
- SMM does not scale well. OS environments tend to be very sensitive to the amount of time spent in SMM. If the time spent servicing an SMI exceeds about 300 microseconds, OS visible artifacts may become a problem. This short duration leaves very little time for any meaningful work to be done. To work around this time problem, any SMM-hosted SEZ environment would need to be scheduled by the OS. This is still likely to be insufficient, because SMM runs with all interrupts inhibited, which significantly complicates OS scheduling algorithms. Furthermore, basing SEZ time on an OS scheduler permits ring 0 denial of service attacks on SEZ applications. Finally, if the SEZ application needs access to system resources (for example, the network stack, display, storage), the device must be either dedicated to SMM or arbitrated with the OS.

"SMM does not provide sufficient scalability, attestability, or assurance to meet the security demands of SEZ applications."

- While SMM runs X86 code, no common software environment or services exist to support SEZ applications. While the Unified Extensible Firmware Interface (UEFI) may partially address this issue (heap management, for example), UEFI is far from universal, nor does it include a sufficient set of services to provide a base for SEZ applications [2]. This lack of a common software support environment implies that an SEZ application must itself implement all of the platform support it requires. This is an ecosystem problem that has no clear solution.

While SMM provides BIOS a robust environment to implement runtime functions and services and has proven many times to be a valuable tool, it is not sufficient for SEZ usage. SMM does not provide sufficient scalability, attestability, or assurance to meet the security demands of SEZ applications.

An SEZ Based on a Virtual Machine Monitor

Since virtual machine monitors (VMMs) that use hardware support run at a higher privilege level than the OS, VMMs can be used to create in-band SEZs. VMMs are used to isolate the memory and I/O of the SEZ from the OS, by trapping on certain operations. Intel® Virtualization Technology (Intel® VT) can be used to virtualize OS page-table management, for example. OS independent memory isolation can be provided by inserting a layer of software under the OS, called a VMM. The VMM runs at a higher privilege level. As a result, it can force the OS to fault into the VMM and to control access to the memory ranges the OS is allowed to access. Intel® Trusted Execution Technology (Intel® TXT) provides a mechanism for a trusted boot of the VMM. Intel VT allows the VMM to trap on various paging events (for example, control-register accesses, translation look-aside buffer invalidation, and so on), which enables the VMM to install its own page tables that also conform to the OS's context-separation requirements. There are a number of variants of such VMM-based, page-table-management algorithms. They are commonly referred to as page-table shadowing algorithms. Intel's shadow page table partitioning approach is called virtualization-enabled integrity services (VIS) and is described in more detail in [3, 4, 5, and 6].

"VMMs are used to isolate the memory and I/O of the SEZ from the OS, by trapping on certain operations."

In summary, the VIS core manages two sets of page tables:

- *Active page table (APT).* This is the page table created and managed by the P-MAPS core in response to the OS's creation and manipulation of the guest page table (owned and managed by the OS).
- *Protected page table (PPT).* This is the page table created and managed by the P-MAPS core in response to a registration by a software module running in the guest OS. In response to the registration, the software module is measured, and a PPT is created for the SEZ, such that the rest of the OS code (running via the APT mappings) cannot execute within the address space defined by the PPT.

"There are a number of variants of such VMM-based, page-table-management algorithms."

The setup of VIS is shown in Figure 2.

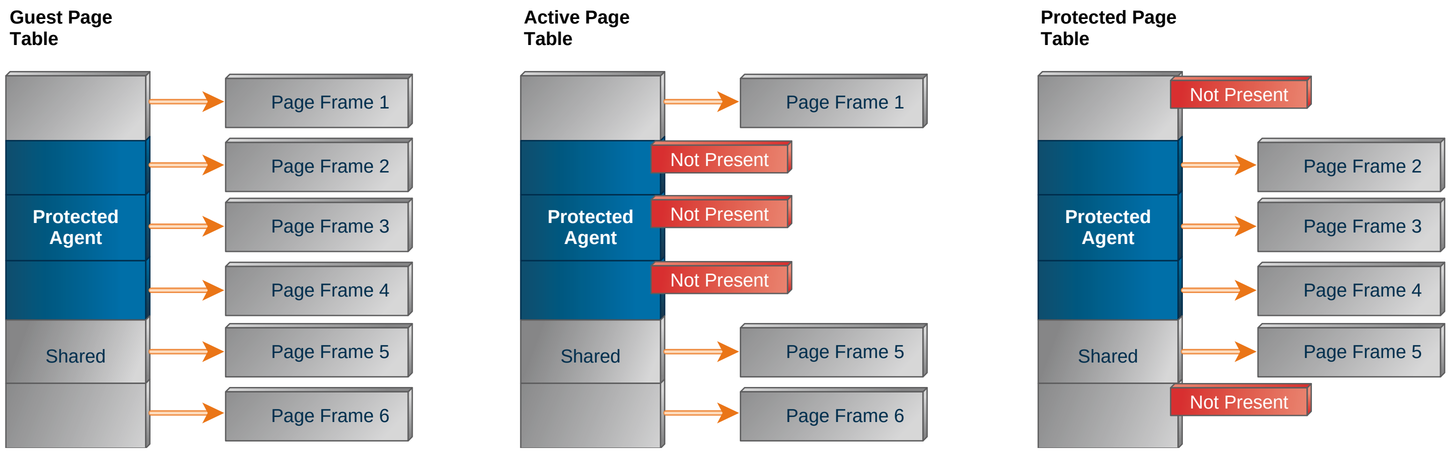


Figure 2: APT and PPT Managed by the P-MAPS Core
Source: Intel Corporation, 2009

The Ideal Secure Execution Zone

There has been considerable research into creating an SEZ for executing software agents in a secure manner on a PC.

The ideal SEZ is a measurable, tamper-resistant environment that executes code and returns the result to entities outside the SEZ, together with a proof that the result was actually generated inside the SEZ. The fundamental trade-offs for an SEZ are these:

- Threat models and spectrum of addressed use cases
- Size of the TCB—the enforcement entity
- Complexity of implementation, validation, deployment, and support
- Liability

SEZs can be divided into three main categories: in-band, out-of-band executing on the host, and those executing at a remote site.

In-band Secure Execution Zone

An in-band SEZ on a PC is created within the confines of the host OS. The SEZ is essentially a compartment that runs inside the linear address space of the OS processes. The SEZ selectively uses OS services but it is immune to interference from agents running at the OS privilege levels or the user privilege levels. The SEZ is enforced by entities on the platform that run at a higher privilege level than the OS itself. These enforcement entities may be CPU subsystems that have the privilege to access CPU resources that are not directly available to the OS (for example, CPU subsystems have access to internal CPU states that are not accessible to the OS) or software entities that run at higher (than OS) privilege modes as provided by the CPU (for example, a micro-vmm in VMX-root mode). Figure 3 shows a typical SEZ in an open execution environment.

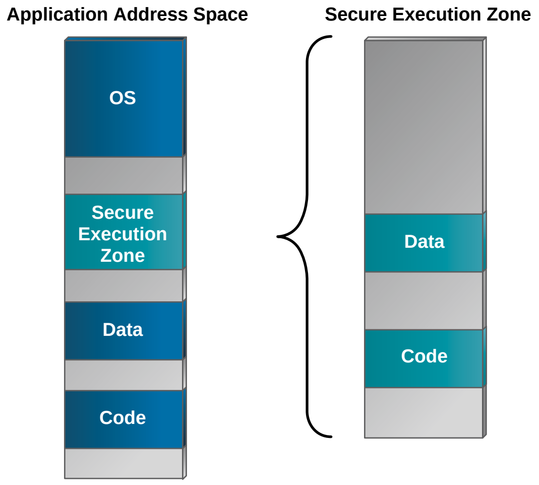


Figure 3: In-band Secure Execution Zone
Source: Intel Corporation, 2009

The SEZ runs within the confines of the OS; that is, the SEZ is in the linear address space of the OS (or its processes) and is scheduled by the OS. Since the SEZ generally sends a signal to its enforcement entity when the computation is completed, a DoS attack can be detected (but cannot be prevented) in most cases.

Out-of-band Secure Execution Zone

An out-of-band SEZ is a zone that is created in parallel with the OS. It may execute on the main CPU or on a processor adjacent to the main CPU. The IBM* 4758 Cryptographic Coprocessor [7] is an example of such an out-of-band SEZ. It may run at the same privilege level as the OS, but it enjoys higher trustworthiness than the main OS by virtue of its limited usage and its tightly access-controlled agents that run inside these SEZs. Some of these SEZs also include enhanced hardware protection. In other words, since this kind of SEZ executes only trusted code in a controlled fashion, it enjoys higher trustworthiness than the OS. Like an in-band SEZ, an out-of-band SEZ is also created by an entity running at a higher privilege level than the host OS. Figure 4 shows a typical out-of-band SEZ.

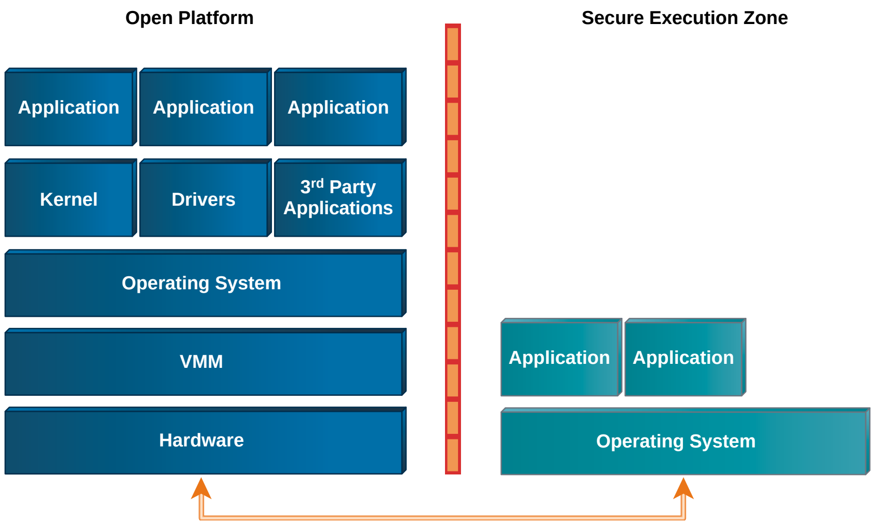


Figure 4: Out-of-band Secure Execution Zone
Source: Intel Corporation, 2009

One advantage of an out-of-band SEZ is that it is less susceptible to DoS attacks than an in-band SEZ, since the out-of-band SEZ does not have to depend on the host OS for any of its functionality. In addition, since this kind of SEZ runs on bare metal hardware or over an entity that emulates bare metal hardware, it runs its own OS, and as a result, is not dependent on the host OS. Running an OS inside an SEZ also has its pros and cons. In its favor, the OS can be suitably modified for the needs of the agents inside the SEZ and can be stripped down to a bare minimum, as a result enabling the agents to trust the OS services. Running against it, however, is the fact that any OS is likely to be greater than 10K lines of code, and as such, will have a larger attack surface than an in-band SEZ running over an untrusted OS. The out-of-band SEZ can still be functional when the host OS is not running.

"Any OS is likely to be greater than 10K lines of code, and as such, will have a larger attack surface than an in-band SEZ running over an untrusted OS."

An out-of-band SEZ can also be hardened against hardware attacks to whatever extent is deemed necessary to protect valuable data, such as keying material. This can be done without the expense of hardening the entire system.

The disadvantage of an out-of-band SEZ is that it is restricted to the limited performance and functionality offered in the SEZ environment. Most out-of-band SEZs do not provide a general environment for third-party applications to run their code.

"Since this SEZ is off the platform, a mutual trust relationship has to exist between the client and the cloud."

Out-of-band Remote Site SEZ

With the advent of cloud computing, out-of-band, remote-site SEZs are likely to come to the fore. These kinds of SEZs follow a client-server model, wherein the client packages code and initialization data for computation to a remote entity in the cloud (after mutual authentication) and receives the result of the computation with a proof that the computation was done in an SEZ in the cloud with the attributes of an SEZ. Since this SEZ is off the platform, a mutual trust relationship has to exist between the client and the cloud that can subsequently be enforced by using various cryptographic mechanisms. A remote site-based SEZ also has advantages and disadvantages.

The advantage is that clients do not need additional hardware or system software for creating SEZs; the SEZ does not eat into the client's resources, and the client can access the latest resources (for example, fixed function blocks, algorithms, and so forth) available in the cloud but not available in the client hardware.

The disadvantages of a remote site-based SEZ are that the underlying trust relationships are hard to create and even harder to enforce. These kinds of SEZs can be a liability when something does not end up as expected. A sub-problem of remote site-based SEZ trust relationships is privacy protection: the client, the user, or both might be identifiable with certain provable attributes. It is often hard to protect the remote SEZ without a local SEZ. Without protected authentication and data transfer, the remote SEZ is subject to attacks from client software. Further, performance limitations may preclude some applications due to the bandwidth limitations.

"The ability of an agent to access a certain resource on the platform is proof of the fact that it is running inside an SEZ."

Secure Execution Zone Interfaces

All SEZs need a bidirectional set of interfaces to interact with the entities outside the SEZ in order to receive workloads, deliver results, and use services that are not available inside the SEZ. The design of these interfaces is one of the most challenging aspects of SEZ design. An SEZ has to provide a proof of execution to the requestor. Ideally an SEZ will communicate directly with some other trusted entity such as another SEZ or a trusted hardware device. If the requestor runs in an unprotected environment, generally in the host OS, then the ability of the requestor to validate the proof of execution before trusting the results of the execution is very limited.

Therefore, each SEZ needs an off-platform entity to be able to validate the proof of execution provided by the SEZ. Since servers running in data centers are considered to be more trustworthy by virtue of strict physical and digital access control, they offer the appropriate environment needed for this verification. Consequently, the platform has to be provisioned with a secret that is only usable by an SEZ and possession of which can be validated by a remote entity (attestation). Alternatively, an SEZ can control the resources on a platform, and the ability of an agent to access a certain resource on the platform is proof of the fact that it is running inside an SEZ. For example, if the SEZ controls a network interface device (NID) and a remote entity receives a packet from the NID, the remote entity automatically assumes that the packet has been either sent by an agent running in the SEZ or by its delegate.

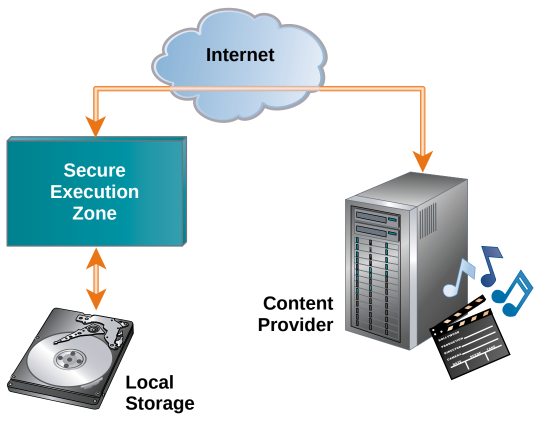


Figure 5: Remote Attestation Diagram
Source: Intel Corporation, 2009

Provisioning and Attestation

As shown in Figure 5, an SEZ needs a mechanism for provisioning a secret into the SEZ and a mechanism for proving the possession of a secret to an off-platform verifier (attestation). Provisioning and attestation [8] are two tightly-bound problems. Remote provisioning needs a platform to report its identity to its membership in a group before it can receive a secret. As a result, the platform has to be provisioned with a root secret during the time of manufacture, assembly, or by a trusted entity that has physical possession of the platform.

The root secret provisioned in the platform is likely to get compromised in the field. The compromised root secret of one instance of a platform should not reveal the root secret of another instance: it should not lead to a *break once run anywhere* (BORE) attack.

Summary

SEZs protect critical applications from various attacks. The selection of an SEZ is dependent on a number of factors including the threat protection requirement, the type of execution environment, and the resources available to the application.

References

[1] B. Lampson, M. Abadi, M. Burrows and E. Wobber. "Authentication in Distributed Systems: Theory and Practice." *ACM Transactions on Computer Systems, page 6*, 1992.

[2] "Unified Extensible Firmware Interface (UEFI)."
At *http://www.uefi.org/home*

[3] Ravi Sahita and Uday Savagaonkar. "Towards Virtualization-based Framework for Information Traceability." In *Insider Attack and Cyber Security: Beyond the Hacker*. Springer Book Company, USA, pages 113-132, August 2008.

[4] Ravi Sahita, Uday Savagaonkar, Prashant Dewan, and David Durham. "Mitigating Lying Endpoint Problem in Virtualized Network Access Frameworks." *Lecture Notes in Computer Science: Managing Virtualization of Networks and Services*, Volume 4785/2007, Springer Book Company, Berlin, Germany, pages 135-146, September 2007.

[5] Ravi Sahita, Ulhas Warrier, and Prashant Dewan. "Protecting Critical Applications on Mobile Platforms." *Intel Technology Journal*, Volume 13, Issue 1, 2009.

[6] David Grawrock. "Dynamics Of A Trusted Platform: A Building Block Approach."

[7] At *http://www-03.ibm.com/*

[8] US Patent 6990579. "Platform and method for remote attestation of a platform." Inventors: Howard C. Herbert, David W. Grawrock, Carl M. Ellison, Roger A. Golliver, Derrick C. Lin, Francis X. McKeen, Gilbert Neiger, Ken Reneris, James A. Sutton, Shreekant S. Thakkar, and Millind Mittal. Assignee: Intel Corporation.

Acknowledgments

The authors thank Joe Cihula and David Grawrock for their feedback and input in preparing this article.

Author Biographies

Kirk Brannock is a Principal Engineer in the Mobile Platforms Group and is focused primarily on platform security concerns. He began his career at Intel in 1994 after graduating from the Computer Science program at Portland State University. He has focused on BIOS and platform-level software interactions throughout his career, including BIOS engineering and architecture in desktop, workstation products, software and services group, and mobile products. His work on BIOS included the creation of two completely new BIOS bases, one of which became UEFI, in wide usage today. More recently, Kirk has been involved with the security community contributing to TXT and other related technologies. His e-mail is kirk.brannock at intel.com.

Prashant Dewan is a Research Scientist at Intel Labs. His research interests are in the area of network and system security, distributed systems, and reputation management. He received a Ph.D. degree in Computer Science in 2004 and an M.S. degree in Computer Science in 2002 from the Department of Computer Science, Arizona State University. His e-mail is prashant.dewan at intel.com.

Frank McKeen is a Research Scientist at Intel Labs. His research interests are in the area of system security and microprocessor architecture. He received a BSEE degree in 1978 from Northeastern University. His e-mail is frank.mckeen at intel.com.

Uday Savagaonkar received a Master of Technology degree in Electrical Engineering from the Indian Institute of Technology, Mumbai, in 1998, and he received a Ph.D. degree in Electrical Engineering from Purdue University, Indiana, in 2002. Since 2002, Dr. Savagaonkar has been working for Intel Corporation. Currently he is a Sr. Research Scientist at Intel Labs. Dr. Savagaonkar has published several research articles in refereed journals and conferences, has applied for several patents, and has contributed book chapters in a variety of areas, among which are applied game theory, network pricing, and platform security. His current research interests include network endpoint security, secure execution environments, cryptographic memory protections, and applied cryptography. His e-mail is uday.r.savagaonkar at intel.com.

Copyright

NEW PROCESSOR INSTRUCTIONS FOR ACCELERATING ENCRYPTION AND AUTHENTICATION ALGORITHMS

Contributors

Shay Gueron
Intel Corporation

Michael E. Kounavis
Intel Corporation

Index Words

Authenticated Encryption
Advanced Encryption Standard (AES)
Galois Counter Mode (GCM)
New Processor Instructions

Abstract

We present a new set of processor instructions for accelerating Advanced Encryption Standard (AES) encryption and decryption, and for accelerating AES-Galois Counter mode (AES-GCM) authenticated encryption. Four instructions are used for accelerating AES, and a fifth instruction that computes the carry-less product of 2 64-bit operands is used for accelerating the GCM mode of operation. In addition to performance acceleration, these instructions help protect the implementations from software side-channel attacks. In this article, we describe the instructions and how they are used for speeding up AES-GCM encryption.

Firstly, we examine modes of operation, such as counter mode (CTR), that can be sped up by processing multiple data blocks in parallel. Then, we present a novel technique for efficiently computing Galois hashes whereby a reduction method in the Galois field GF (2^{128}) can be used in cases where the field's reduction polynomial is sparse. The use of the new instructions, combined with algorithms and software techniques, offer a comprehensive solution for speeding up AES-GCM authenticated encryption.

Introduction

Message confidentiality and integrity are key to the security of applications, operating systems, and the network infrastructure of the Internet in the future. As a result, improving the performance and security of encryption and authentication has significant benefits for today's computer platforms. In this article we describe new tools that Intel offers in this area.

First, we focus on instructions and software techniques for supporting high-performance *encryption and decryption* (for confidentiality) by using the Advanced Encryption Standard (AES), and for supporting the Galois Counter Mode (GCM), which is used (for integrity) in the AES-GCM authenticated-encryption protocol.

AES is the Federal Information Processing Standard for symmetric encryption and is defined by FIPS Publication #197 (2001). It is widely used in a large variety of security applications.

GCM is a message authentication protocol that was endorsed by the US Government in April 2006, and it is typically used, together with AES, for authenticated encryption. The GCM is also used by the IEEE 802.1ae standard, where its usage is recommended for forwarding rates higher than 10 Gbps. Other usage models of GCM include IPsec (IPsec RFC 4106), the storage standard P1619, and security protocols over fiber channels (ISO-T11 standard).

"Improving the performance and security of encryption and authentication has significant benefits for today's computer platforms."

We describe how AES can be accelerated with the new processor instructions that Intel is introducing to the ISA, and we look at how GCM can be accelerated with another new instruction that computes the carry-less product of two 64-bit operands. This new instruction is used by a reduction algorithm that takes advantage of the fact that in GCM, the reduction polynomial of the associated GF (2^{128}) Galois field is sparse [2]. This algorithm uses carry-less multiplications, implemented by this new instruction, and due to its efficiency, there is no need to add field-specific reduction logic to the processor architecture: the generic carry-less multiplication primitive can do the computation.

"AES-128, AES-192, and AES-256 process the data block in 10, 12, or 14 iterations of pre-defined sequences of transformations, which are also called AES rounds."

The AES Instructions

What is AES?

AES is a block cipher that encrypts a 128-bit block (plaintext) to a 128-bit block (ciphertext), or decrypts a 128-bit block (ciphertext) to a 128-bit block (plaintext). AES uses a cipher key whose length can be 128, 192, or 256 bits, respectively. Hereafter, encryption/decryption with a cipher key of 128, 192, or 256 bits is denoted as AES-128, AES-192, AES-256, respectively. AES-128, AES-192, and AES-256 process the data block in 10, 12, or 14 iterations of pre-defined sequences of transformations, which are also called AES rounds (hereafter referred to simply as *rounds*). The rounds are identical except for the last one, which slightly differs from the others (by skipping one of the transformations). They operate on two 128-bit inputs: *state* and *round key*. Each round from 1 to 10/12/14 uses a different round key. The 10/12/14 round keys are derived from the cipher key by the *key expansion algorithm*. This algorithm is independent of the processed data, and can therefore be carried out independently of the encryption/decryption phase. (Typically, the key is expanded once and is thereafter used for many data blocks by using some cipher mode of operation). The data block is processed serially; initially, the input data block is XOR'd with the first 128 bits of the cipher key to generate the state (an intermediate cipher result). Subsequently, the state passes, serially, through 10/12/14 rounds, with each round consisting of a sequence of transformations operating on the state and using a different round key. Code 1 illustrates the AES encryption flow, for a single 16-byte block, by using the terminology of the FIPS197 document, which defines AES (see also [1] for details).

"The input data block is XOR'd with the first 128 bits of the cipher key to generate the state."

```
AES encryption flow

Input:

Data: 16 bytes to encrypt
Round_Key_Encrypt: array of 11-15 16-byte blocks which are the expanded
cipher key

Tmp = AddRoundKey (Data, Round_Key_Encrypt [0])
For round = 1-9 or 1-11 or 1-13:
        Tmp = ShiftRows (Tmp)
        Tmp = SubBytes (Tmp)
        Tmp = MixColumns (Tmp)
        Tmp = AddRoundKey (Tmp, Round_Key_Encrypt [round])
end loop
Tmp = ShiftRows (Tmp)
Tmp = SubBytes (Tmp)
Tmp = AddRoundKey (Tmp, Round_Key_Encrypt [10 or 12 or 14])

Output:
Tmp : (16 bytes)
```

Code 1: AES Encryption of a Single Block
Source: Intel Corporation, 2009

Instruction Specification

A new set of instructions will be introduced in the next generation of the Intel® processor family to facilitate secure and high-performance AES encryption and decryption. The instructions are described by using the terminology found in FIPS197; in this document, the details of the transformations, the encryption/decryption flows, and key expansions are provided in full. See also [1] for details on the AES instructions and their usages.

AESENC xmm1, xmm2/m128
Tmp := xmm1/m128
RoundKey :=xmm2/m128
Tmp := ShiftRows (Tmp)
Tmp := SubBytes (Tmp)
Tmp := MixColumns (Tmp)
xmm1:= Tmp xor RoundKey
	AESENCLAST xmm1, xmm2/m128
	Tmp := xmm1/m128
	RoundKey := xmm2 /m128
	Tmp := ShiftRows (Tmp)
	Tmp := SubBytes (Tmp)
	
	xmm1:= Tmp xor RoundKey

AESDEC xmm1, xmm2/m128
Tmp:=xmm1/m128
RoundKey := xmm2/m128
Tmp := InvShiftRows (Tmp)
Tmp := InvSubBytes (Tmp)
Tmp := InvMixColumns (Tmp)
xmm1:= Tmp xor RoundKey
	AESDECLAST xmm1, xmm2/m128
	Tmp:= xmm1/m128
	RoundKey := xmm2/m128
	Tmp := InvShiftRows (Tmp)
	Tmp := InvSubBytes (Tmp)
	
	xmm1:= Tmp xor RoundKey

AESKEYGENASSIST xmm1, xmm2/m128, imm8	
Tmp := xmm2/m128	
RCON[31-8] := 0; RCON[7 -0] := imm8;	
X3[31-0] := Tmp[127 -96];	X2[31-0] := Tmp[95-64];
X1[31-0] := Tmp[63 -32];	X0[31-0] := Tmp[31-0];
xmm1 := [RotWord (SubWord	(X3)) XOR RCON, SubWord (X3),
Rotword (SubWord	(X1)) XOR RCON, SubWord (X1)]
AESIMC xmm1, xmm2/m128	
RoundKey := xmm2/m128;	
xmm1 := InvMixColumns (RoundKey)	
Examples:	
xmm1 =	7b5b54657374566563746f725d53475d
xmm2 =	48692853686179295b477565726f6e5d
AESENC result:	a8311c2f9fdba3c58b104b58ded7e595
AESENCLAST result:	c7fb881e938c5964177ec42553fdc611
AESDEC result:	138ac342faea2787b58eb95eb730392a
AESDECLAST result:	c5a391ef6b317f95d410637b72a593d0
xmm2 =	7b5b54657374566563746f725d53475d
AESIMC	result: 627a6f6644b109c82b18330a81c3b3e5
xmm2 =	3c4fcf098815f7aba6d2ae2816157e2b; imm8 = 1
AESKEYGENASSIST result: 01eb848beb848a013424b5e524b5e434	

Table 1: The Six New AES Instructions
Source: Intel Corporation, 2009

The AES architecture offers six instructions to support AES (see Table 1 and Code 2). AESENC and AESENCLAST support encryption. AESDEC and AESDECLAST are building blocks suitable for decryption that use the Equivalent Inverse Cipher (see FIPS197 for definition). AESIMC and AESKEYGENASSIST support the key expansion. AESIMC facilitates the conversion of the encryption round keys to a form suitable for the Equivalent Inverse Cipher. AESKEYGENASSIST uses an immediate byte as part of the input (used as RCON).

"The instructions execute the AES transformations efficiently by holding the operands in the SIMD registers of the IA architecture, and by using dedicated hardware."

```
AES encryption flow
(using the new AES instructions)

Input:
Data: 16 bytes to encrypt
Round_Key_Encrypt: array of 11-15 16-byte blocks
which are the expanded cipher key

Tmp = XOR128 (Data, Round_Key_Encrypt [0])
For round = 1-9 or 1-11 or 1-13:
        Tmp = AESENC (Tmp, Round_Key_Encrypt [round])
end loop
Tmp = AESENCLAST (Tmp, Round_Key_Encrypt [10 or 12 or 14])

Output:
Tmp: (16 bytes)
```

Code 2: AES Encryption with New AES Instructions
Source: Intel Corporation, 2009

The AES processor instructions are designed based on the structure of AES, a structure that includes transformations in a GF(2^8) Galois field and byte shuffling transformation. The instructions execute the AES transformations efficiently by holding the operands in the SIMD registers of the IA architecture, and by using dedicated hardware.

"Software side channels are vulnerabilities in the software implementation of cryptographic algorithms."

Protection against Software Side-Channel Vulnerabilities

An important security advantage of using AES instructions is the protection it provides against software side-channel attacks (by other Ring 3 malicious applications).

Software side channels are vulnerabilities in the software implementation of cryptographic algorithms, and they emerge in multiple processing environments (cores, threads, and operating systems).

Cache-based software side-channel attacks exploit the fact that it takes time for a particular piece of data to be accessed, if the data are not in the cache. Because of this time lag, malicious code can potentially detect the memory addresses that are being accessed during encryption or decryption. In software implementations of AES, based on look-up tables, these addresses can reveal sensitive information about the keys.

"Because of this time lag, malicious code can potentially detect the memory addresses that are being accessed during encryption or decryption."

On the other hand, the AES instructions are implemented via combinatorial logic, and their latency is data-independent. Therefore, software implementations of AES that use these instructions are not susceptible to any of the known software side-channel attacks [1].

The Carry-less Multiplication Instruction

What is Carry-less Multiplication?

Carry-less multiplication, also known as binary polynomial multiplication, is the mathematical operation of computing the product of two operands without generating or propagating carries. Such multiplications are an essential step in computing multiplications in binary Galois fields.

Carry-less multiplication is defined as follows. Let A and B be two n-bit operands

$$A = [a_{n-1}\, a_{n-2} \cdots a_0] \quad (1)$$

and

$$B = [b_{n-1}\, b_{n-2} \cdots b_0] \quad (2)$$

If the carry-less product of A and B is denoted by $C = A \cdot B$, and C is the bit array $C = [c_{2n-1}\, c_{2n-2} \cdots c_0]$, then, the bits of C are defined as the following functions of the bits of the inputs A and B:

$$c_i = \bigoplus_{j=0}^{i} a_j b_{i-j} \quad (3)$$

for $0 \le i \le n-1$, and

$$c_i = \bigoplus_{j=i-n+1}^{n-1} a_j b_{i-j} \quad (4)$$

for $n-1 \le i \le 2n-1$.

See illustration in Figure 1.

As an example, if A = 0x63746f725d53475d and B = 0x5b477565726f6e5d, the carry-less product is $C = A \cdot B$ = 0x1d4d84c85c3440c0929633d5d36f0451.

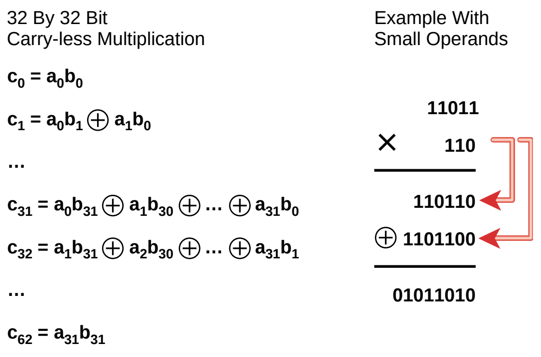

32 By 32 Bit
Carry-less Multiplication
$c_0 = a_0b_0$
$c_1 = a_0b_1 \oplus a_1b_0$
...
$c_{31} = a_0b_{31} \oplus a_1b_{30} \oplus \ldots \oplus a_{31}b_0$
$c_{32} = a_1b_{31} \oplus a_2b_{30} \oplus \ldots \oplus a_{31}b_1$
...
$c_{62} = a_{31}b_{31}$
Example With
Small Operands
11011
× 110
110110
⊕ 1101100
01011010

Figure 1: Carry-less Multiplication
Source: Intel Corporation, 2009

The PCLMULQDQ Instruction

Together with the AES instructions, Intel also introduces PCLMULQDQ, an instruction for computing the carry-less multiplication of two 64-bit halves (hereafter referred to as *quadwords*) that are selected from the instruction's two operands (two xmm registers or one xmm register and one memory location), according to an immediate byte value (imm8), defined in Table 2:

PCLMULQDQ xmm1, xmm2/m128, imm8

imm8[7:0]	Operation
0x00	xmm2/m128[63:0] · xmm1[63:0]
0x01	xmm2/m128[63:0] · xmm1[127:64]
0x10	xmm2/m128[127:64] · xmm1[63:0]
0x11	xmm2/m128[127:64] · xmm1[127:64]

Table 2: PCLMULQDQ: Instruction for Carry-less Multiplication
Source: Intel Corporation, 2009

"AES-GCM uses a single key to both encrypt and authenticate data."

Efficient Implementation of AES-GCM

What is AES-GCM

AES-GCM is an authenticated encryption algorithm, which is built upon AES encryption in counter (CTR) mode, and it is a computation of a Galois hash. AES-GCM uses a single key to both encrypt and authenticate data. An authenticated encryption algorithm is different from classical encryption and authentication schemes, where two independent keys are required to make both functions secure [2-8].

Figures 2 and 3 briefly describe the CTR mode of operation and the AES-GCM algorithm. Figure 2 shows the AES encryption of multiple blocks, by using CTR mode, and Figure 3 illustrates the AES-GCM algorithm.

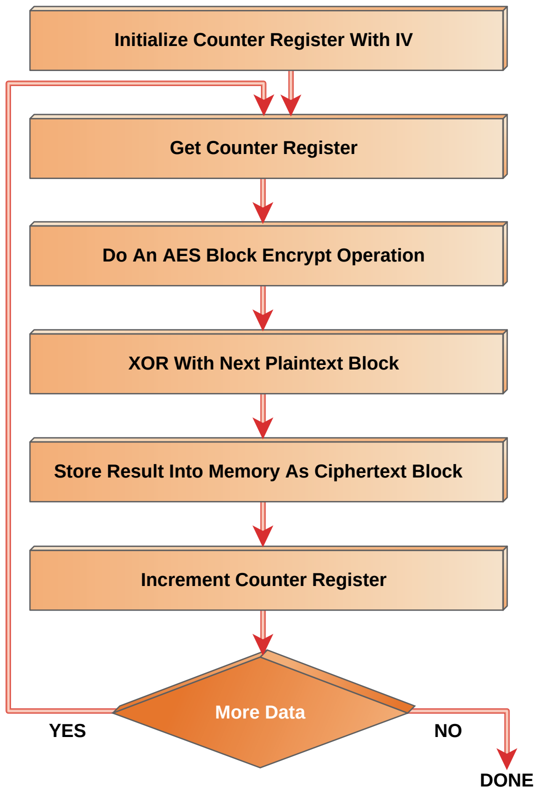

Initialize Counter Register With IV
Get Counter Register
Do An AES Block Encrypt Operation
XOR With Next Plaintext Block
Store Result Into Memory As Ciphertext Block
Increment Counter Register
More Data
YES
NO
DONE

Figure 2: AES Encryption in Counter Mode
Source: Intel Corporation, 2009

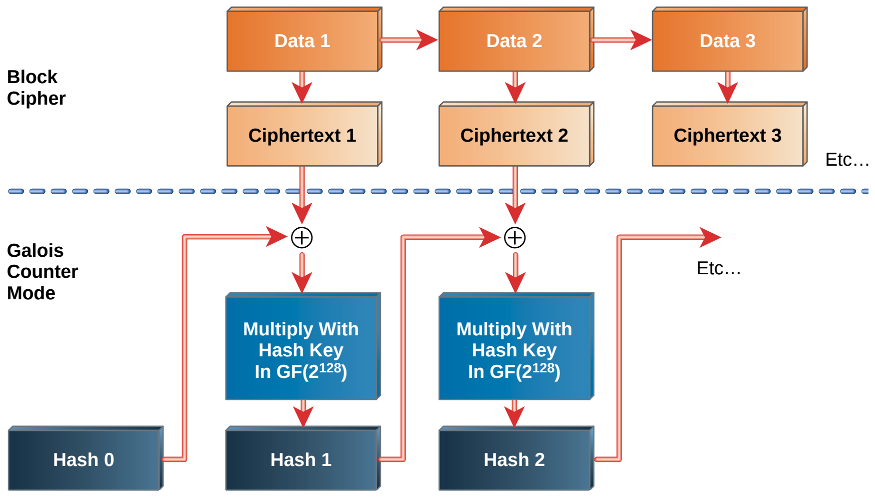


Figure 3: AES-GCM Algorithm
Source: Intel Corporation, 2009

High Performance Implementation of AES in Counter Mode

Significant performance optimization for encrypting (and decrypting) can be achieved if software using the AES instructions is designed to process multiple data blocks in parallel. This *software pipelining* technique is applicable for parallelizable modes of operation such as Electronic Code Book (ECB), CTR, and decryption with the Cipher Block Chaining (CBC-Decryption) mode.

In such modes, different data blocks can be encrypted (or decrypted) independently of each other, and the hardware that supports the AES round instructions is pipelined. This allows independent AES instructions to be dispatched, theoretically every one to two CPU clock cycles, if data can be provided sufficiently fast. As a result, the AES throughput can be significantly enhanced for parallel modes of operation, if the software implementation itself is pipelined. Instead of completing the encryption of one data block and then continuing to the subsequent block, it is preferable to write software sequences that compute one AES round on multiple blocks, using one round key, and only then continue to compute the subsequent round for multiple blocks. This technique speeds up any parallelizable mode of operation, in particular the CTR mode. Code 3 shows a code snippet encrypting eight blocks in parallel as part of the CTR mode (where the counters are encrypted).

"The AES throughput can be significantly enhanced for parallel modes of operation, if the software implementation itself is pipelined."

```
mov rdx, OFFSET keyex_addr
; load Round key
movdqu xmm0, XMMWORD PTR [rdx]
pxor xmm1, xmm0
pxor xmm2, xmm0
pxor xmm3, xmm0
pxor xmm4, xmm0
pxor xmm5, xmm0
pxor xmm6, xmm0
pxor xmm7, xmm0
pxor xmm8, xmm0

mov ecx, 9
main_loop:
; load Round key
add rdx, 0x10
movdqu xmm0, XMMWORD PTR [rdx]
aesenc xmm1, xmm0
aesenc xmm2, xmm0
aesenc xmm3, xmm0
aesenc xmm4, xmm0
aesenc xmm5, xmm0
aesenc xmm6, xmm0
aesenc xmm7, xmm0
aesenc xmm8, xmm0
;continued to the next column
```

```
loop main_loop
add rdx, 0x10
movdqu xmm0, XMMWORD PTR [rdx]
aesenclast xmm1, xmm0
aesenclast xmm2, xmm0
aesenclast xmm3, xmm0
aesenclast xmm4, xmm0
aesenclast xmm5, xmm0
aesenclast xmm6, xmm0
aesenclast xmm7, xmm0
aesenclast xmm8, xmm0
; storing the encrypted blocks
movdqu XMMWORD PTR [dest], xmm1
movdqu XMMWORD PTR [dest+0x10], xmm2
movdqu XMMWORD PTR [dest+0x20], xmm3
movdqu XMMWORD PTR [dest+0x30], xmm4
movdqu XMMWORD PTR [dest+0x40], xmm5
movdqu XMMWORD PTR [dest+0x50], xmm6
movdqu XMMWORD PTR [dest+0x60], xmm7
movdqu XMMWORD PTR [dest+0x70], xmm8
```

Code 3: AES Encryption of Eight Blocks in Parallel
Source: Intel Corporation, 2009

High Performance Implementation of Galois Counter Mode

We now examine how GCM can be efficiently computed by using the PCLMULQDQ instruction, in combination with some improved algorithms.

The most compute-intensive part of GCM is the computation of the Galois hash, which is multiplication in the finite field GF(2^{128}), defined by the reduction modulo $g = x^{128} + x^7 + x^2 + x + 1$. The multiplication in this field is carried out in two steps: the first step is the carry-less multiplication of two 128-bit elements, and the second step is the reduction of the 256-bit carry-less product modulo $g = x^{128} + x^7 + x^2 + x + 1$. We explain these steps in the rest of this section.

Computing a 256-bit Carry-less Product with the PCLMULQDQ Instruction

The following algorithm steps can be viewed as one iteration of a carry-less *schoolbook multiplication*:

1. Multiply carry-less by the following operands: A_0 with B_0, A_1 with B_1, A_0 with B_1, and A_1 with B_0. Let the results of the above four multiplications be $A_0 \bullet B_0 = [C_1 : C_0]$, $A_1 \bullet B_1 = [D_1 : D_0]$, $A_0 \bullet B_1 = [E_1 : E_0]$, $A_1 \bullet B_0 = [F_1 : F_0]$

2. Construct the 256-bit output of the multiplication $[A_1 : A_0] \bullet [B_1 : B_0]$ as follows:
$$[A_1 : A_0] \bullet [B_1 : B_0] = [D_1 : F_1 \oplus E_1 \oplus D_0 : F_0 \oplus E_0 \oplus C_1 : C_0] \quad (5)$$

One can also trade off one multiplication for additional XOR operations. This 2-step alternative approach can be viewed as a *carry-less Karatsuba* multiplication [9]:

1. Multiply carry-less by the following operands: A_1 with B_1, A_0 with B_0, and $A_0 \oplus A_1$ with $B_0 \oplus B_1$. Let the results of the above three multiplications be $[C_1 : C_0]$, $[D_1 : D_0]$, and $[E_1 : E_0]$, respectively.

2. Construct the 256-bit output of the multiplication $[A_1 : A_0] * [B_1 : B_0]$ as follows:
$$[A_1 : A_0] \bullet [B_1 : B_0] = [C_1 : C_0 \oplus C_1 \oplus D_1 \oplus E_1 : D_1 \oplus C_0 \oplus D_0 \oplus E_0 : D_0] \quad (6)$$

Both methods can be used for the first step of the computation of the Galois hash.

Efficient Reduction

To reduce a 256-bit carry-less product modulo g, we first split it into two 128-bit halves. The least-significant half is simply XOR'd with the final remainder (since the degree of g is 128). For the most-significant part, we develop an algorithm that realizes division via two multiplications. This algorithm can be seen as an extension of the Barrett reduction algorithm [10] to modulo-2 arithmetic, or as an extension of the Feldmeier CRC generation algorithm [11] to dividends and divisors of arbitrary size.

Since we do not need to take into account the least-significant half of the input (see above), we investigate the efficient generation of a remainder $p(x)$ defined as follows:

$$p(x) = c(x) \cdot x^t \bmod g(x) \quad (7)$$

Where $c(x)$ is a polynomial of degree s-1, with coefficients in GF(2), representing the most-significant bits of the carry-less product (for GCM, s =128).

t is the degree of the polynomial g. (for GCM, t = 128).

$g(x)$ is the irreducible polynomial defining the final field (for GCM, $g = g(x) = x^{128} + x^7 + x^2 + x + 1$).

For the polynomials $p(x)$, $c(x)$, and $g(x)$ we write:

$$c(x) = c_{s-1} x^{s-1} + c_{s-2} x^{s-2} + \ldots + c_1 x + c_0,\ p(x) = p_{t-1} x^{t-1} + p_{t-2} x^{t-2} + \ldots + p_1 x + p_0,\ \text{and } g(x) = g_t x^t + g_{t-1} x^{t-1} + \ldots + g_1 x + g_0 \qquad (8)$$

Hereafter, we use the notation $L^u(v)$ to denote the coefficients of the u least-significant terms of the polynomial v and $M^u(v)$ to denote the coefficients of its u most-significant terms. The polynomial $p(x)$ can be expressed as:

$$p(x) = c(x) \cdot x^t \bmod g(x) = g(x) \cdot q(x) \bmod x^t \qquad (9)$$

where $q(x)$ is a polynomial of degree $s - 1$ equal to the quotient from the division of $c(x) \cdot x^t$ with g. The intuition behind equation (9) is that the t least-significant terms of the dividend $c(x) \cdot x^t$ equal zero. Further, the dividend $c(x) \cdot x^t$ can be expressed as the sum of the polynomials $g \cdot q$ and p:

$$c(x) \cdot x^t = g(x) \cdot q(x) + p(x) \qquad (10)$$

where operator '+' means XOR ('⊕'). From equation (10) one can expect that the t least-significant terms of the polynomial $g \cdot q$ are equal to the terms of the polynomial p. Only if these terms are equal to each other, the result of the XOR operation $g \cdot q \oplus p$ is zero for its t least-significant terms. Hence:

$$p(x) = g(x) \cdot q(x) \bmod x^t = L^t(g(x) \cdot q(x)) \qquad (11)$$

Now we define:

$$g(x) = g_t x^t + g^*(x) \qquad (12)$$

The polynomial g^* represents the t least-significant terms of the polynomial g. Obviously,

$$p(x) = L^t(g(x) \cdot q(x)) = L^t(q(x) \cdot g^*(x) + q(x) \cdot g_t x^t \qquad (13)$$

However, the t least-significant terms of the polynomial $q \cdot g_t \cdot x^t$ are zero. Therefore,

$$p(x) = L^t(q(x) \cdot g^*(x)) \qquad (14)$$

From equation (14) it follows that in order to compute the remainder p we need to know the value of the quotient q. The quotient can be calculated in a similar manner as that of the Barrett reduction algorithm:

$$(9) \Leftrightarrow c(x) \cdot x^{t+s} = g(x) \cdot q(x) \cdot x^s + p(x) \cdot x^s \qquad (15)$$

Let:

$$x^{t+s} = g(x) \cdot q^{+}(x) + p^{+}(x) \tag{16}$$

where q^{+} is an s-degree polynomial equal to the quotient from the division of x^{t+s} with g, and p^{+} is the remainder from this division. The degree of the polynomial p^{+} is $t - 1$. From equations (15) and (16) we get:

$$\left.\begin{matrix}(15)\\(16)\end{matrix}\right\} \Leftrightarrow c(x) \cdot g(x) \cdot q^{+}(x) + c(x)\, p^{+}(x)$$

$$= g(x) \cdot q(x) \cdot x^{s} + p(x) \cdot x^{s} \tag{17}$$

and

$$(17) \Rightarrow M^{s}(c(x) \cdot g(x) \cdot q^{+}(x) + c(x)\, p^{+}(x))$$
$$= M^{s}(g(x) \cdot q(x) \cdot x^{s} + p(x)\, x^{s}) \tag{18}$$

One can see that the polynomials $c \cdot g \cdot q^{+}$ and $g \cdot q \cdot x^{s}$ are of degree $t + 2 \cdot s - 1$. The polynomial $c \cdot p^{+}$ is of degree $t + s - 2$, and the polynomial $p \cdot x^{s}$ is of degree $t + s - 1$. As a result, the s most-significant terms of the polynomials in the left- and right-hand side of equation (18) are not affected by the polynomials $c \cdot p^{+}$ and $p \cdot x^{s}$. Hence,

$$(18) \Rightarrow M^{s}(c(x) \cdot g(x) \cdot q^{+}(x))$$
$$= M^{s}(g(x) \cdot q(x) \cdot x^{s}) \tag{19}$$

Next, we observe that the s most-significant terms of the polynomial $c \cdot g \cdot q^{+}$ are equal to the s most-significant terms of the polynomial $g \cdot M^{s}(c \cdot q^{+}) \cdot x^{s}$. The polynomial $M^{s}(c \cdot q) \cdot x^{s}$ results from $c \cdot q^{+}$ by replacing the s least-significant terms of this polynomial with zeros. The intuition behind this observation is that the s most-significant terms of the polynomial $c \cdot g \cdot q^{+}$ are calculated by adding together the s most-significant terms of the polynomial $c \cdot q^{+}$ in as many offset positions as defined by the terms of the polynomial g. Thus, the s most-significant terms of $c \cdot g \cdot q^{+}$ do not depend on the s least-significant terms of $c \cdot q^{+}$, and consequently, this results in

$$(19) \Rightarrow M^{s}(g(x) \cdot M^{s}(c(x) \cdot q^{+}(x)) \cdot x^{s})$$
$$= M^{s}(g(x) \cdot q(x) \cdot x^{s}) \tag{20}$$

Equation (20) is satisfied for q given by:

$$q = M^{s}(c(x) \cdot q^{+}(x)) \tag{21}$$

Since there is a unique quotient q satisfying equation (10) one can show that there is a unique quotient q satisfying equation (20). As a result this quotient q must be equal to $M^{s}(c(x) \cdot q^{+}(x))$.

It follows that the polynomial p is found by

$$p(x) = L^{t}(g^{*}(x) \cdot M^{s}(c(x) \cdot q^{+}(x))) \tag{22}$$

Equation (22) can be translated to the following algorithm for computing the polynomial p.

Preprocessing: Compute the polynomials g^* and q^+ for the given irreducible polynomial g. The polynomial g^* is of degree $t - 1$, consisting of the t least-significant terms of g, and the polynomial q^+ is of degree s, and is equal to the quotient of the division of x^{t+s} with the polynomial g.

1. Multiply the input c with q^+. The result is a polynomial of degree $2s - 1$.
2. Multiply the s most-significant terms of the polynomial resulting from Step 1 with g^*. The result is a polynomial of degree $t+s - 2$.
3. Return the t least-significant terms of the polynomial resulting from Step 2. This is the desired remainder.

One can see that the quotient from the division of x^{256} with g is g itself. The polynomial $g = g(x) = x^{128} + x^7 + x^2 + x + 1$ contains only five non-zero coefficients (therefore also called *pentanomial*). This polynomial can be represented as the bit sequence [1:<120 zeros>:10000111]. Multiplying this carry-less with a 128-bit value and keeping the 128 most-significant bits can be obtained by (i) shifting the 64 most-significant bits of the input by 63, 62, and 57 bit positions to the right; (ii) XOR-ing these shifted copies with the 64 least-significant bits of the input. Next, carry-less multiply this 128-bit result with g, and keep the 128 least-significant bits. This can be done by (i) shifting the 128-bit input by 1, 2, and 7 positions to the left and (ii) XOR-ing the results.

Special attention should be paid when implementing the GCM mode, because the standard specifies that the bits inside their 128-bit double quadwords are reflected. That is, the bit corresponding to the least-significant coefficient of the polynomial representation of the entities that are multiplied is bit number 127, rather than bit number 0. This also implies that the order of bits in the reduction polynomial is [11100001:<120 zeros>:1] as opposed to [1:<120 zeros>:10000111]. Note that this property is not merely the difference between Little Endian and Big Endian notations.

To handle this peculiarity, we point out the following fundamental property of carry-less multiplication, namely

$$\mathit{reflected}(A)\ \mathit{reflected}(B) = \mathit{reflected}(A \cdot B) >> 1 \qquad (23)$$

Using the identity (23), and shifting-by-one of two registers containing the carry-less product of two inputs, the Galois Hash can be computed using the PCLMULQDQ instruction, regardless of the bit-order representation of the input and the output operands (see [2] for details and code samples).

"Significant acceleration can be achieved when the new instructions are used efficiently."

Estimated Performance Benefits

Encryption in CTR mode can be accelerated by roughly an order of magnitude, compared with some current and frequently used AES look-up tables that are based on implementations of AES (for example, OpenSSL implementation). The 64-bit carry-less multiplication helps speed up the computation of the GCM, and avoids the potential security problems that are associated with the current implementation that is based on look-up tables.

Conclusion

In this article, we describe Intel's new instructions for high-performance cryptographic processing, which also eliminate all currently known software side-channel threats. Our main focus was on the use of these instructions for obtaining a high-performing and secure implementation of AES-GCM authenticated encryption.

Significant acceleration can be achieved when the new instructions are used efficiently, by taking advantage of the parallelism in the CTR mode, and by using the new techniques for carry-less multiplication and reduction modulo sparse polynomials.

References

[1] S. Gueron. "Advanced Encryption Standard (AES) Instructions Set." At ***http://softwarecommunity.intel.com***

[2] S. Gueron and M. Kounavis. "Carry-Less Multiplication and its Usage for Computing The GCM Mode." At ***http://softwarecommunity.intel.com***

[3] M. Dworkin. "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) for Confidentiality and Authentication." *Federal Information Processing Standard Publication FIPS 800-38D*, April 20, 2006. At ***http://csrc.nist.gov***

[4] "IEEE 802.1AE - Media Access Control (MAC) Security." *IEEE 802.1 MAC Security Task Group Document*. At ***http://www.ieee802.org***

[5] J. Viega and D. McGrew. "The Use of Galois/Counter Mode (GCM)." In *IPsec Encapsulating Security Payload (ESP), IETF RFC 4106.* At ***http://www.rfc-archive.org***

[6] "IEEE Project 1619.1 Home." At ***https://siswg.net***

[7] "The Fibre Channel Security Protocols Project." *ISO-T11 Committee Archive*. At ***http://www.t11.org***

[8] J. Salowey, A. Choudhury and D. McGrew. "RSA-based AES-GCM Cipher Suites for TLS." At ***http://www1.ietf.org***

[9] A. Karatsuba and Y. Ofman. "Multiplication of Multidigit Numbers on Automata." *Soviet Physics. Doklady*, Volume 7, pages 595-596, 1963.

[10] P. Barrett. "Implementing the Rivest, Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor." *Master's Thesis*, University of Oxford, UK, 1986.

[11] D. Feldmeier. "Fast Software Implementation of Error Correcting Codes." *IEEE Transactions on Networking*, December, 1995.

Acknowledgments

We are indebted to all of our many colleagues who contributed to the design, specification, and implementation of the new AES and PCLMULQDQ instructions.

Author Biographies

Shay Gueron is an Intel Principal Engineer. He is the security architect for the CPU Architecture Department in the Mobility Group, at the Israel Development Center. His interests include applied security, cryptography, and algorithms. He is also an Associate Professor at the Department of Mathematics in the Faculty of Science at the University of Haifa in Israel. Shay is one of the co-recipients of the Intel Achievement Award (2008) for his work on the AES instructions. His e-mail is shay.gueron at intel.com.

Michael E. Kounavis is a Senior Research Scientist working at Intel Labs. Michael is responsible for conducting research on novel digital arithmetic and cryptographic algorithms with the aim of accelerating a wide range of client, server, and networking applications. Michael is a co-inventor of the CRC32 SSE4 instruction of the Intel Core i7 architecture used for iSCSI CRC generation. He is also a co-recipient of an Intel Achievement Award (2008) for his work on the AES instructions. His e-mail is michael.e.kounavis at intel.com.

Copyright

https://everywhere! ENCRYPTING THE INTERNET

Contributors

Satyajit Grover
Intel Corporation

Xiaozhu Kang
Intel Corporation

Michael Kounavis
Intel Corporation

Frank Berry
Intel Corporation

Index Words

Secure Communications
Cryptographic Algorithm Acceleration
AES-NI

Abstract

The evolution of the Internet has resulted in large quantities of information being exchanged by businesses or private individuals. The nature of this information is typically both public and private, and much of it is transmitted over the hyper text transfer protocol (HTTP) in an insecure manner. A small amount of traffic, however, is transmitted by way of the secure sockets layer (SSL) over HTTP, known as HTTPS. HTTPS is a secure cryptographic protocol that provides encryption and message authentication over HTTP. The introduction of SSL over HTTP significantly increases the cost of processing traffic for service providers, as it sometimes requires an investment in expensive end-point acceleration devices. In this article, we present new technologies and results that show the economy of using general-purpose hardware for high-volume HTTPS traffic. Our solution is three pronged. First, we discuss new CPU instructions and show how to use them to significantly accelerate basic cryptographic operations, including symmetric encryption and message authentication. Second, we present results from a novel software implementation of the RSA algorithm that accelerates another compute-intensive part of the HTTPS protocol—public key encryption. Third, we show that the efficiency of a web server can be improved by balancing the web server workload with the public key cryptographic workload on a processor that is enabled with simultaneous multi-threading (SMT) technology. In conclusion, we show that these advances provide web services the tools to greatly reduce the cost of implementing HTTPS for all their HTTP traffic.

Introduction

As of January 2009, it is estimated that the Internet connects six hundred and twenty five million hosts. Every second, vast amounts of information are exchanged amongst these millions of computers. These data contain public and private information, which is often confidential and needs to be protected. Security protocols for safeguarding information are routinely used in banking and e-commerce. Private information, however, has not been protected on the Internet in general. Examples of private information (beyond banking and e-commerce data) include personal e-mail, instant messages, presence, location, streamed video, search queries, and interactions on a wide variety of on-line social networks. The reason for this neglect is primarily economic. Security protocols rely on cryptography, and as such are compute-resource-intensive. As a result, securing private information requires that an on-line service provider invest heavily in computation resources. In this article we present new technologies that can reduce the cost of on-line secure communications, thus making it a viable option for a large number of services.

"Private information has not been protected on the Internet in general."

A lot of private information is transmitted over the HTTP in an insecure manner. HTTP exists in the application layer of the TCP/IP protocol stack. The Secure Sockets Layer (SSL) and its successor, Transport Layer Security (TLS) are security technologies applied to the same layer. In this article, we specifically refer to SSL/TLS over the HTTP application layer, known as HTTPS. The introduction of HTTPS significantly increases the cost of processing traffic for web-service providers, due to the fact that it is not possible for previous-generation, web-server hardware to process high-volume HTTPS traffic with all the added cryptographic overhead. In order to process this high-volume traffic, a web-service provider has to invest in expensive end-point SSL/TLS acceleration devices. This added cost makes HTTPS a selective or premium choice among web-service providers. Consequently, a large amount of private information is transmitted over the web in an insecure manner and can, therefore, be intercepted or modified en route. In this article we provide a solution to this problem by presenting new technologies and results that show that it is now possible to use general-purpose hardware for high-volume HTTPS traffic.

"In order to process this high-volume traffic, a web-service provider has to invest in expensive end-point SSL/TLS acceleration devices."

Organization of this Article

Our solution to mitigating the overhead of an SSL-enabled HTTP session is three pronged. First, we discuss new processor instructions and show how to use them to accelerate basic cryptographic operations by factors. This substantially reduces the server load during the bulk data transfer phase of HTTPS. Second, we present results from a novel implementation of the Rivest Shamir Adleman (RSA) asymmetric cryptographic algorithm [1] that accelerates the most compute-intensive stage of the HTTPS protocol: that is, the stage in which the server has to decrypt handshake messages coming from a large number of clients. Third, we analyze a web server and show how its efficiency can be improved by balancing a web-server workload with a cryptographic workload on a processor enabled with simultaneous multi-threading (SMT) technology. By doing this, we show that the cryptographic overhead can be hidden by performing it in parallel with memory accesses that have long stall times.

We then elaborate on our motivation and vision of deploying HTTPS everywhere. First, we present an in-depth study of an SSL session and its resource requirements. We then describe our three-pronged strategy, together with our experiments and results.

Motivation

The motivation behind our research is primarily to enable widespread use of, and access to, HTTPS. It is important for service providers and users to be able to trust each other for their mutual benefit. An important aspect of the trust comes from knowing that private communications are kept confidential and adhere to the policies established between providers and users. Users need to be educated and informed about the benefits of HTTPS for privacy in on-line communications. Providers need to adopt ubiquitous HTTPS offerings to ensure that they hold up their end of the deal. Enabling HTTPS without expensive investment is important in creating such a partnership.

"It is important for service providers and users to be able to trust each other for their mutual benefit."

"If there is not an end-to-end solution for security, these communications can be easily observed by network neighbors."

HTTPS provides an end-to-end solution to data privacy and authenticity. This end-to-end solution ensures that when users transmit information from their device to a provider, the information cannot be seen by man-in-the-middle spyware. This is important due to the fact that packets travel over untrusted networks all the time in the Internet. Although most routing devices are hidden from direct observation, they are not impervious to motivated eavesdroppers. Even more observable are the publicly accessible wireless access points that are in use all over the world. These access points broadcast information to all devices managed by them. If there is not an end-to-end solution for security, these communications can be easily observed by network neighbors. There are other solutions to the security problem, such as Layer 3 Virtual Private Networks (VPNs), but VPNs are typically limited to networks where users communicate with other users within a centrally managed network; that is, having multiple users but a single provider. In such cases, the network provider already has strict policies about data privacy and security that are communicated to users via training. For example, e-mails within an enterprise are often allowed only over the enterprise-managed VPN. For the larger Internet, users connect across the networks of multiple providers. In addition, in recent years we have seen a reduction in the use of a wide variety of communication protocols (for example, FTP) in favor of the HTTP protocol. In this environment, HTTPS is the most viable solution to enabling private and secure communications amongst the large and growing numbers of users and providers.

Future applications of HTTPS may include widespread e-mail encryption, secure video streaming, secure instant messaging and encrypted web searching. These are a few of the many applications of HTTPS that are not widely used today. Moreover, with each passing year, users are putting more of their personal and private information on-line. Cloud computing enables them to access their information across all their devices everywhere. We believe that it is inevitable that users will demand HTTPS support from their providers for all their communications. Being prepared for that day led us to research and develop the technologies described in this article. We envision that with these advancements, every HTTP-based communication made by every device today will be HTTPS-based in the near future. We refer to this as "https://everywhere!".

Anatomy of a Secure Sockets Layer Session

Secure Sockets Layer

Secure sockets layer (SSL) (later versions known as Transport Layer Security, TLS) includes a handshake phase and a cryptographic data exchange phase. The overall SSL handshake is shown in Figure 1. In our diagram, in phase 1, the handshake begins when a client sends a server a list of algorithms the client is willing to support as well as a random number used as input to the key generation process.

"HTTPS is the most viable solution to enabling private and secure communications amongst the large and growing numbers of users and providers."

In phase 2, the server chooses a cipher and sends it back, along with a certificate containing the server's public key. The certificate proves the server's identity. We note that the domain name of the server is also verified via the certificate (which helps eliminate phishing sites) and demonstrates to the user they are talking with the correct server/service. In addition, the server provides a second random number that is used as part of the key generation process. In phase 3, the client verifies the server's certificate and extracts the server's public key. The client then generates a random secret string called a pre-master secret and encrypts it by using the server's public key. The pre-master secret is sent to the server. In phase 4, the server decrypts the pre-master secret by using RSA. This is one of the most compute-intensive parts of the SSL transaction on the server. The client and server then independently compute their session keys by using the pre-master secret to apply a procedure called a key derivation function (KDF) twice. In phases 5 and 6, the SSL handshake phase ends with the communicating parties sending authentication codes to each other, computed on all original handshake messages.

"In phases 5 and 6, the SSL handshake phase ends with the communicating parties sending authentication codes to each other."

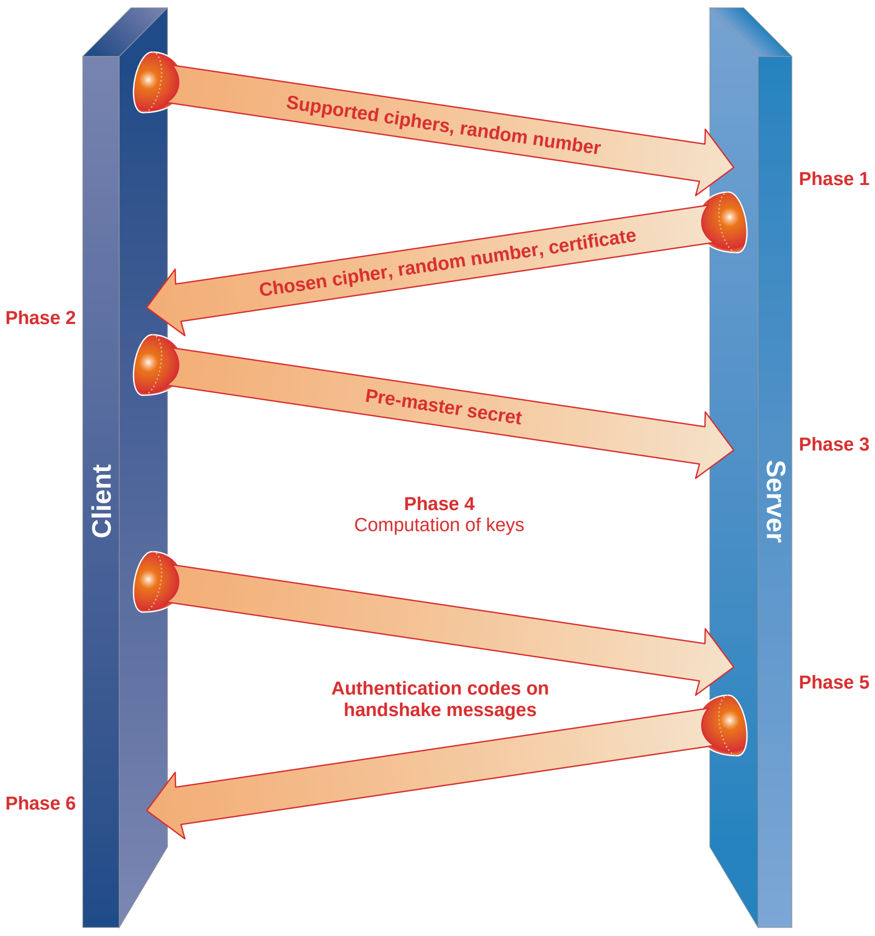

Client
Server
Supported ciphers, random number
Phase 1
Chosen cipher, random number, certificate
Phase 2
Pre-master secret
Phase 3
Phase 4
Computation of keys
Phase 5
Authentication codes on handshake messages
Phase 6

Figure 1: Secure Sockets Layer (SSL) Handshake
Source: Intel Corporation, 2009

"In IPsec, protection is supported on an IP-packet-by-IP-packet basis, whereas in SSL, protection is supported on a fragment-by-fragment basis."

In SSL, the data are transferred by using a record protocol. The record protocol breaks a data stream into a series of fragments, each of which is independently protected and transmitted. In other words, in IPsec, protection is supported on an IP-packet-by-IP-packet basis, whereas in SSL, protection is supported on a fragment-by-fragment basis. Before a fragment is transmitted, it is protected against attacks by the calculation of a message authentication code on the fragment. The fragment's authentication code is appended to the fragment, thereby forming a payload that is encrypted by using the cipher selected by the server. Finally, a record header is added to the payload. The concatenated header and encrypted payload are referred to as a record.

A secure web server is clearly a memory-intensive application. For an SSL connection, the most significant type of overhead is the one related to cryptography. This includes the operations of encrypting packets with a symmetric key, providing message authentication support, and setting up the session by using RSA, as mentioned previously. In the section that follows, we describe in more detail two encryption algorithms that we accelerate with technologies described in this article: the Advanced Encryption Standard (AES) and Rivest Shamir Adleman (RSA).

The Advanced Encryption Standard and the RSA Algorithm

Advanced Encryption Standard

"AES is a symmetric encryption algorithm, which means that the same key is used for converting a plaintext to ciphertext, and vice versa."

AES is the United States Government's standard for symmetric encryption, defined by FIPS Publication #197 (2001) [2, 3]. It is used in a large variety of applications where high throughput and security are required. In HTTPS, it can be used to provide confidentiality for the information that is transmitted over the Internet. AES is a symmetric encryption algorithm, which means that the same key is used for converting a plaintext to ciphertext, and vice versa. The structure of AES is shown in Figure 2.

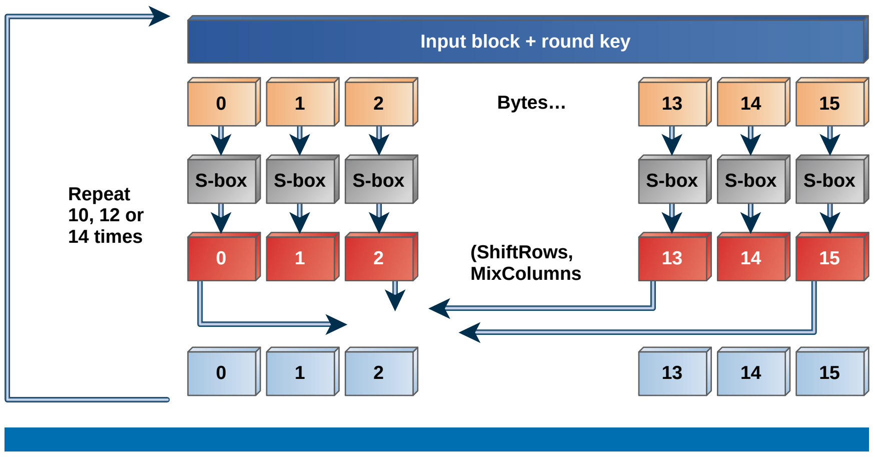


Figure 2: Structure of AES
Source: Intel Corporation, 2009

AES first expands a key (that can be 128, 192, or 256 bits long) into a *key schedule*. A key schedule is a sequence of 128-bit words, called round keys, that are used during the encryption process. The encryption process itself is a succession of a set of mathematical transformations called *AES rounds.*

During an AES round the input to the round is first XOR'd with a round key from the key schedule. The exclusive OR (XOR) logical operation can also be seen as addition without generating carries.

"The exclusive OR (XOR) logical operation can also be seen as addition without generating carries."

In the next step of a round, each of the 16 bytes of the AES state is replaced by another value by using a non-linear transformation called *S-box*. The AES S-box consists of two stages. The first stage is an inversion, not in regular integer arithmetic, but in a finite field arithmetic based on the set GF(2^8). The second stage is an affine transformation. During encryption, the input x, which is considered an element of GF(2^8); that is, an 8-bit vector, is first inverted, and then an affine map is applied to the result. During decryption, the input (y) goes through the inverse affine map and is then inverted in GF(2^8). The GF(2^8) inversions just mentioned are performed in GF(2^8), defined by the irreducible polynomial $p(x) = x^8 + x^4 + x^3 + x + 1$ or 0x11B.

Next, the replaced byte values undergo two linear transformations called *ShiftRows* and *MixColumns*. ShiftRows is just a byte permutation. The MixColumns transformation operates on the columns of a matrix representation of the AES state. Each column is replaced by another one that results from a matrix multiplication. The transformation used for encryption is shown in Equation 1. In this equation, matrix-times-vector multiplications are performed according to the rules of the arithmetic of GF(2^8) with the same irreducible polynomial that is used in the AES S-box, namely, $p(x) = x^8 + x^4 + x^3 + x + 1$.

$$output = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot input \qquad \text{Eq. 1}$$

During decryption, inverse ShiftRows is followed by inverse MixColumns. The inverse MixColumns transformation is shown in Equation 2.

$$output = \begin{bmatrix} 0xE & 0xB & 0xD & 0x9 \\ 0x9 & 0xE & 0xB & 0xD \\ 0xD & 0x9 & 0xE & 0xB \\ 0xB & 0xD & 0x9 & 0xE \end{bmatrix} \cdot input \qquad \text{Eq. 2}$$

Note that while the MixColumns transformation multiplies the bytes of each column with the factors 1, 1, 2 and 3, the inverse MixColumns transformation multiplies the bytes of each column by the factors 0x9, 0xE, 0xB, and 0xD. The same process is repeated 10, 12, or 14 times depending on the key size (128, 192, or 256 bits). The last AES round omits the MixColumns transformation.

The RSA Algorithm

RSA is a public key cryptographic scheme. The main idea behind public key cryptography is that encryption techniques can be associated with *back doors*. By back doors we mean secrets, known only to at least one of the communicating parties, which can simplify the decryption process. In public key cryptography, a message is encrypted by using a public key. A public key is associated with a secret called the *private key*. Without knowledge of the private key it is difficult to decrypt a message. Similarly, it is very difficult for an attacker to determine what the plaintext is.

"By back doors we mean secrets, known only to at least one of the communicating parties, which can simplify the decryption process."

We further explain how public key cryptography works by presenting the RSA algorithm as an example. In this algorithm, the communicating parties choose two random large prime numbers p and q. For maximum security, p and q are of equal length. The communicating parties then compute the product:

$$n = p \cdot q \qquad \text{Eq. 3}$$

Then the parties choose the public key E, such that the numbers E and $(p-1) \cdot (q-1)$ are relatively prime. The private key associated with the public key is a number D, such that:*

$$E \cdot D \bmod (p-1) \cdot (q-1) = 1 \qquad \text{Eq. 4}$$

The encryption formula is simply:

$$C = M^{E} \bmod n \qquad \text{Eq. 5}$$

where M is the plaintext and C is the ciphertext. The decryption formula is similarly:

$$M = C^{D} \bmod n \qquad \text{Eq. 6}$$

One can show that the decryption formula is correct by using elements of number theory:

$$C^{D} \bmod n = M^{ED} \bmod n = M^{k \cdot (p-1) \cdot (q-1)+1} \bmod n = M \cdot (M^{k})^{(p-1) \cdot (q-1)} \bmod n$$
$$= M \cdot (l \cdot p \cdot q + 1) \bmod p \cdot q = M \qquad \text{Eq. 7}$$

The above calculation is correct since $(p-1) \cdot (q-1)$ is the Euler function of the product $p \cdot q$, and we know from number theory (by using the Little Fermat Theorem) that:

$$(M^{k})^{(p-1) \cdot (q-1)} = (l \cdot p \cdot q + 1) \qquad \text{Eq. 8}$$

for some l. D and E can be used interchangeably, meaning that encryption can be done by using D, and decryption can be done by using E.

RSA is typically implemented using Chinese Remainder Theorem that reduces a single modular exponentiation operation into two operations of half length. Each modular exponentiation in turn is implemented, by using the square-and-multiply technique that reduces the exponentiation operation into a sequence of modular squaring and modular multiplication operations. Square-and-multiply may also be augmented with some windowing method for reducing the number of modular multiplications. Finally, modular squaring and multiplication operations can be reduced to big number multiplications by using reduction techniques such as Montgomery's or Barrett's [4, 5].

"The next micro-architecture generation adds new instructions for potentially speeding up symmetric encryption by 3-10 times."

Acceleration Technologies

We are currently researching solutions to realize the vision of encrypting the Internet so that HTTPS sessions are accelerated by factors. The next micro-architecture generation adds new instructions for potentially speeding up symmetric encryption by 3-10 times. These instructions not only provide better performance but also protect applications against an important type of threat known as side-channel attacks. Second, we have developed improved integer arithmetic software that can speed up key exchange and establishment procedures by a factor of 40 to 100 percent.

Third, the Intel® Core™ i7 micro-architecture re-introduces the SMT feature into the CPU. SMT is ideal for hiding the cycles of compute-intensive public key encryption software under the stall times of network application memory lookups.

New Processor Instructions

In the next generation of Intel processors, a new set of instructions will be introduced that enable high performance and secure round encryption and decryption. These instructions are AESENC (AES round encryption), AESENCLAST (AES last round encryption), AESDEC (AES round decryption), and AESDECLAST (AES last round decryption). Two additional instructions are also introduced for implementing the key schedule transformation, AESIMC and AESKEYGENASSIST.

The design of these new processor instructions is based on the structure of AES. Systems such as AES involve complex mathematical operations such as finite field multiplications and inversions [6], as discussed earlier. These operations are time or memory consuming when implemented in software, but they are much faster and more power efficient when implemented by using combinatorial logic. Moreover, the operands involved in finite field operations can fit into the SIMD registers of the IA architecture. In this article, we discuss the concept of implementing an entire AES round as a single IA processor instruction by using combinatorial logic. An AES round instruction is much faster than its equivalent table-lookup-based software routine and can also be pipelined, thereby allowing the computation of an independent AES round result potentially every clock cycle.

"These operations are much faster and more power efficient when implemented by using combinatorial logic."

"Our new instructions out-perform by approximately 3-10 times the best software techniques doing equivalent mathematical operations on the same platform."

The AESENC instruction implements these transformations of the AES specification in the order presented: ShiftRows, S-box, MixColumns, and AddRoundKey. The AESENCLAST implements ShiftRows, S-box, and AddRoundKey but not MixColumns, since the last round omits this transformation. The AESDEC instruction implements inverse ShiftRows, inverse S-box, inverse MixColumns, and AddRoundKey. Finally, the AESDECLAST instruction implements inverse ShiftRows, inverse S-box, and AddRoundKey, omitting the inverse MixColumns transformation. More details about these AES instructions can be found in [7].

Our AES instructions can be seen as cryptographic primitives for implementing not only AES but a wide range of cryptographic algorithms. For example, several submissions to NIST's recent SHA-3 hash function competition use the AES round or its primitives as building blocks for computing cryptographic hashes. Moreover, combinations of instruction invocations can be used for creating more generic mathematical primitives for finite field computations. Our new instructions out-perform by approximately 3-10 times the best software techniques doing equivalent mathematical operations on the same platform.

Together with the AES instructions, Intel will offer one new instruction supporting carry-less multiplication, named PCLMULQDQ. This instruction performs carry-less multiplication of two 64-bit quadwords that are selected from the first and second operands, according to the immediate byte value.

"An instruction that accelerates carry-less multiplication is important for accelerating GCM and all communication protocols that depend on it."

Carry-less multiplication, also known as Galois Field (GF) multiplication, is the operation of multiplying two numbers without generating or propagating carries. In the standard integer multiplication, the first operand is shifted as many times as the positions of bits equal to "1" in the second operand. The product of the two operands is derived by adding the shifted versions of the first operand to each other. In carry-less multiplication, the same procedure is followed, except that additions do not generate or propagate carry. In this way, bit additions are equivalent to the exclusive OR (XOR) logical operation.

Carry-less multiplication is an essential component of the computations done as part of many systems and standards, including cyclic redundancy check (CRC), Galois/counter mode (GCM), and binary elliptic curves, and it is very inefficient when implemented in software in today's processors. Thus, an instruction that accelerates carry-less multiplication is important for accelerating GCM and all communication protocols that depend on it [8].

"We are able to accelerate other popular cryptographic schemes such as RSA 2048 and Elliptic Curve Diffie-Hellman, based on the NIST B-233 curve."

Improved Key Establishment Software

We have also developed integer arithmetic software that can accelerate big number multiplication and modular reduction by at least 2X. Such routines are used not only in RSA public key encryption but also in Diffie Hellman key exchange and elliptic curve cryptography (ECC). Using our software, we are able to accelerate RSA 1024 from a performance of approximately 1500 signatures per second (OpenSSL v.0.9.8g) or 2000 signatures per second (OpenSSL v.0.9.8.h), to potentially 2900 signatures per second on a single Intel® Core i7 processor. Similarly, we are able to accelerate other popular cryptographic schemes such as RSA 2048 and Elliptic Curve Diffie-Hellman, based on the NIST B-233 curve.

The performance of RSA can be improved by accelerating the big number multiplication that is an essential and compute-intensive part of the algorithm. Our implementation uses an optimized schoolbook big number multiplication algorithm. RSA is a compute-intensive operation consuming millions of clocks on multiplying, adding, and subtracting 64-bit quantities. However, the state which RSA accesses is small, typically consisting of key information as well as 16-32 multipliers that fit into the L1 cache of Intel CPUs. With our software, an RSA 1024 decrypt operation consumes about 0.99 million clocks, whereas the corresponding RSA 2048 decrypt operation consumes about 6.73 million clocks on an Intel Core i7 processor. This is about 40 percent faster than corresponding operations that use OpenSSL (v. 0.9.8h).

The code listed in Code 1 illustrates the main idea, which is to combine multiply and add operations with a register recycling technique for intermediate values. In Code 1, 'a' and 'b' hold the two large numbers to be multiplied, and the results are stored in 'r'. These operations are repeated over the entire inputs to generate intermediate values that are then combined with addition to produce the large number multiplication result.

```
asm("mulq %3;\n"
        :"=a"(t0), "=d"(t1)
        :"a"(a[0]), "g"(b[0])
        :"cc");
    t2 = t0;
    t3 = t1;
    r[0] = t2;
    t2 = t3;
    t3 = t4;
    t4 = 0;
    asm("movq (%5), %%rax;\n\t"
            "mulq 8(%6);\n\t"
            "addq %3, %0;\n\t"
            "adcq %4, %1;\n\t"
            "adcq $0, %2;\n\t"
            "movq 8(%5), %%rax;\n\t"
            "mulq (%6);\n\t"
            "addq %3, %0;\n\t"
            "adcq %4, %1;\n\t"
            "adcq $0, %2;\n"
            :"+r"(t2), "+r"(t3), "+r"(t4), "=a"(t0), "=d"(t1)
            :"r"(a), "g"(b)
            :"cc");
r[1] = t2;
    asm("mulq %3;\n"
        :"=a"(t0), "=d"(t1)
        :"a"(a[1]), "g"(b[1])
        :"cc");
    asm("addq %2, %0;\n\t"
        "adcq %3, %1;\n"
        :"+r"(t0), "+r"(t1)
        :"r"(t3), "r"(t4)
        :"cc");
    r[2] = t0;
    r[3] = t1;
```

Code 1: RSA Implementation
Source: Intel Corporation, 2009

We also investigated other techniques for big number multiplication, including Karatsuba-like constructions, but we found this schoolbook algorithm implementation to be the fastest [9, 10].

"One compute-intensive thread performs only RSA public key encryption operations, and another thread performs memory access-intensive tasks."

Simultaneous Multi-threading

The most recent Intel i7 core micro-architecture re-introduces the feature of hyper-threading (now referred to as simultaneous multi-threading or SMT) into the CPU. SMT represents a major departure from the earlier core micro-architecture, where each core was single threaded. As part of our research, we have demonstrated that SMT can result in substantial performance improvements for a certain class of workloads. Such workloads are associated with secure web transactions. We propose a new programming model where one compute-intensive thread performs only RSA public key encryption operations, and another thread performs memory access-intensive tasks. We show that RSA is an ideal companion thread for four representative memory access-intensive workloads when SMT is used, resulting in a 10–100 percent potential efficiency increase.

"The RSA computation is hidden under the very long stall times of the memory thread."

The system benefits most when a thread performing dependent-memory lookups is paired with an RSA thread. The throughput of the memory thread almost doubles, reaching the value it would have had if it hadn't been paired with RSA. Another way to interpret the same result is that the RSA computation comes for free, because of SMT. In reality the RSA computation is hidden under the very long stall times of the memory thread. We also observe that the throughput of a single memory thread is increased by approximately 30 percent when SMT is switched on, and the memory thread is multiplexed with another memory thread. The same throughput is almost doubled when the memory thread is paired with an RSA thread. These results indicate that RSA is a much better companion thread than a second memory thread, due to the fact that one workload is memory access-intensive, and the other workload is compute-intensive. If an RSA thread is paired with a memory thread, then RSA performance also increases by 21 percent when SMT is switched ON as compared to OFF [11].

To further validate our position that SMT is beneficial especially to crypto workloads, we built a test bed running SpecWeb* 2005. The test bed consisted of a server machine using an Intel Core i7 processor connected to two client machines running a total of four client engines. We measured the server's capacity with SMT turned on and off for the banking and support (regular HTTP) workloads. Our experiments indicate that SMT improves the overall system performance by at least 10 percent—more for the banking workload than the support workload. This result is in accordance with our earlier experiments, and it indicates that crypto workloads can take advantage of SMT.

"This result is in accordance with our earlier experiments, and it indicates that crypto workloads can take advantage of SMT."

The overall impact of our cryptographic algorithm acceleration technologies is shown in Figure 3. The first bar represents the crypto overhead of a 230 Kbyte SSL transaction as it runs on an Intel Core i7 processor today. The encryption scheme used is AES-256 in the counter mode. The next bar shows the acceleration gain if AES is implemented with the new instructions. The third bar shows the incremental gain by using our RSA software and SMT. Finally, the last bar shows the gain associated with replacing SHA1 with GCM. GCM is a message authentication scheme offering the same functionality as HMAC-SHA1. As is evident from the figure, our acceleration technologies substantially reduce the crypto overheads resulting in significant performance and efficiency improvement.

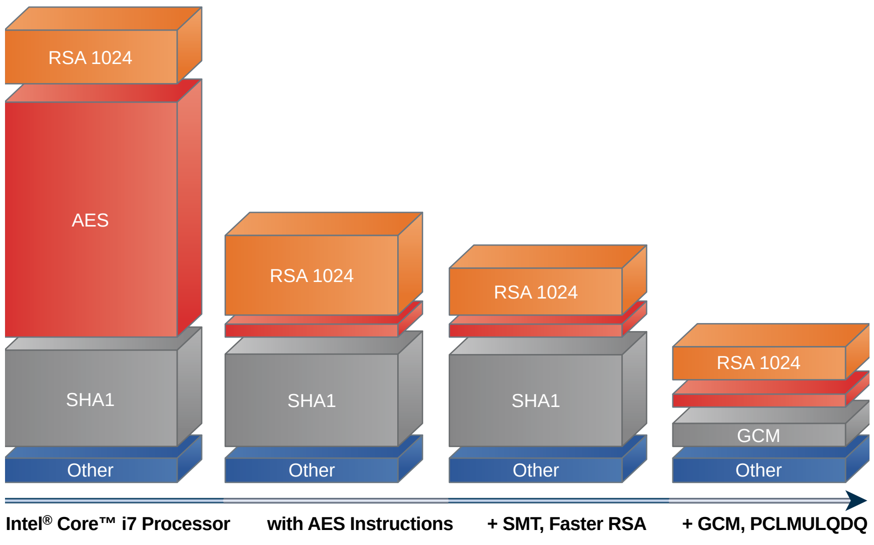


Figure 3: Impact Of Crypto Acceleration Technologies
Source: Intel Corporation, 2009

Conclusion

In summary, Intel is researching new technologies that offer cryptographic algorithm acceleration by factors. We described new processor instructions that can accelerate AES symmetric encryption. This acceleration substantially reduces the server load during the bulk data transfer phase of HTTPS. We also present results from a novel implementation of the RSA asymmetric cryptographic algorithm. This accelerates a very compute-intensive stage of the HTTPS protocol, a stage in which the server has to decrypt handshake messages coming from a large number of clients. Third, we analyze a web server and present some initial experimental results indicating that the efficiency of the server can be improved by balancing a web server workload with a cryptographic workload on an SMT-enabled processor. This shows that the cryptographic overhead can be hidden by performing it in parallel with memory accesses with long stall times. Our ultimate goal is to make general-purpose processors capable of processing and forwarding encrypted traffic at very high speeds so that the Internet can be gradually transformed into a completely secure information delivery infrastructure. We also believe that these technologies can benefit other usage models, such as disk encryption and storage.

"Our ultimate goal is to make general-purpose processors capable of processing and forwarding encrypted traffic at very high speeds so that the Internet can be gradually transformed into a completely secure information delivery infrastructure."

References

[1] R.L. Rivest, A. Shamir, and L. M. Adleman. "*A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.*" Communications of the ACM, 21,2, pages 120–126, February 1978.

[2] V. Rijmen. "Efficient Implementation of the Rijndael S-box." At *http://www.google.com*

[3] "Advanced Encryption Standard." *Federal Information Processing Standards Publication* 197. At *http://csrc.nist.gov*

[4] P. Montgomery. "Multiplication without trial division." *Math. Computation*, Volume 44, pages 519—521, 1985.

[5] P. Barrett. "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor." *Masters Thesis*, University of Oxford, UK, 1986.

[6] S. Gueron, O. Parzanchevsky and O. Zuk. "Masked Inversion in GF(2^n) Using Mixed Field Representations and its Efficient Implementation for AES." *Embedded Cryptographic Hardware: Methodologies & Architectures.* Nadia Nedjah and Luiza de Macedo Mourelle (Editors), Nova Science Publishers, Inc.(ISBN: 1-59454-012-8), 2004.

[7] S. Gueron. "Advanced Encryption Standard (AES) Instructions Set." At: *http://software.intel.com/*

[8] S. Gueron and M. Kounavis. "Carry-Less Multiplication and Its Usage for Computing the GCM Mode." At *http://software.intel.com/*

[9] A. Karatsuba and Y. Ofman. "Multiplication of Multidigit Numbers on Automata." *Soviet Physics—Doklady*, Volume 7, pages 595–596, 1963.

[10] M. E. Kounavis. "A New Method for Fast Integer Multiplication and its Application to Cryptography." In *Proceedings 2007 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*. San Diego, CA, 2007.

[11] S. Grover and M. Kounavis. "On the Impact of Simultaneous Multi-threading on the Performance of Cryptographic Workloads." *Technical Report*, available from the authors upon request.

Author Biographies

Satyajit Grover is a Software Engineer working at Intel Labs. His duties involve researching and prototyping new ideas in the area of system security and integrity. He has been working in this area at Intel for over two years. Previous to that he was a graduate student and research assistant at the Computer Science Department at Portland State University. His e-mail is satyajit.grover at intel.com.

Xiaozhu Kang is a Research Scientist working at Intel Labs. Her research interests include algorithm design and performance analysis. She obtained a Ph.D. degree in Electrical Engineering from Columbia University in 2008, and she joined Intel in January of 2009. Before that, she worked as an intern in Intel Corporation, Mathworks Corporation, and NEC Labs. Her e-mail is xiaozhu.kang at intel.com.

Michael Kounavis is a Senior Research Scientist working at Intel Labs. Michael is responsible for conducting research on novel digital arithmetic and cryptographic algorithms with the aim of accelerating a wide range of client, server, and networking applications. Michael is a co-inventor of the CRC32 SSE4 instruction of the Intel® Core i7 architecture used for iSCSI CRC generation. He is also a co-recipient of the 2008 Intel Achievement Award for his work on AES instructions. His e-mail is michael.e.kounavis at intel.com.

Frank Berry is a Principal Engineer working at Intel Labs. His area of expertise is the hardware/software interface where the hardware and software work closely together. His expertise extends to operating system internals, device drivers, and networking stacks. Frank has received two Intel Achievement Awards for his work on the InfiniBand Architecture and AES Instructions. His e-mail is frank.berry at intel.com.

Copyright

RECENT CONTRIBUTIONS TO CRYPTOGRAPHIC HASH FUNCTIONS

Contributors

Jesse Walker
Intel Corporation

Michael Kounavis
Intel Corporation

Shay Gueron
Intel Corporation

Gary Graunke
Intel Corporation

Index Words

Hash Function
Compression Function
Cascade Construction
Merkle-Damgård
Block Cipher

Abstract

Hash functions are cryptography's most widely used primitives, in that they are a fundamental building block used for a wide variety of constructions. The recent attacks of Joux, Wang, and others against the current generation of hash functions has stimulated a resurgence of research into these primitives as well as spawned an international competition, sponsored by the U.S. Government agency, the National Institute of Standards and Technology (NIST), to create a next-generation hash function design.

In this article, we provide the background to understand what a hash function is and what problems it addresses. We then describe and contrast two radically different designs entered in the NIST competition, Skein and Vortex. Skein was designed by a team assembled from across the industry and academia, while Vortex was designed exclusively by Intel employees. We next go on to describe the design rationale for each hash function, and we compare and contrast the basic design decisions.

Introduction

Hash functions are one of cryptography's most fundamental building blocks, even more so than encryption functions. For example, hash functions are used for digital fingerprinting and commitment schemes, such as message authentication and random number generation, as well as for digital signature schemes, stream ciphers, and random oracles.

"Hash functions are one of cryptography's most fundamental building blocks."

Recently, Andre Joux, one of the leading cryptographers of our time, discovered multi-collision attacks against the general framework in which hash functions are constructed [1], and Xiaoyun Wang created an attack that breaks the collision-resistance property of the most widely deployed hash functions [2], including MD5 and SHA-1. In 2005, Arjen Lenstra and Wang demonstrated how to forge two digital certificates, based on the MD5 hash function, by using different keys but the same signature—something that was hitherto thought to be impossible [3]. These attacks that were discovered, and other vulnerabilities that were exposed by researchers have stimulated a resurgence of research into hash functions. This research has also spawned an international competition, sponsored by the National Institute of Standards and Technology (NIST), an agency responsible for standards used by the U.S. Government, to create a next-generation hash function design.

In this article we highlight some recent work in hash function development. We begin by providing some background on hash functions and look at the problems that hash functions address. We then sketch how to build a hash function. Moving on, we outline recent seminal work in the field of hash functions. We describe two radically different designs entered in the NIST competition, the Skein and Vortex designs, created in part with Intel participation. This is followed by a discussion of the design rationale for each where we also compare and contrast the basic design decisions. We end with a summary of our findings.

Hash Functions

In this section, we first describe what a hash function is and then look at some typical use cases.

What is a Hash Function?

A hash functions is usually defined as a function H satisfying three properties [4]:

- *Collision resistance*. It is computationally infeasible to find two distinct bit strings $s \neq s'$ such that $H(s) = H(s')$.
- *Pre-image resistance*. Given a hash value t in the range of H, it is computationally infeasible to find a string s for which $H(s) = t$.
- *2nd Pre-image resistance*. Given a string s and hash value t such that $H(s) = t$, it is computationally infeasible to find a second string s' such that $H(s') = t$ as well.

A hash function maps the set of all bit strings into a *message digest* of defined length, called the hash function's *block size*. Since there are many more strings than message digests, at least one digest output by the hash function must be the image of more than one input string. It is therefore remarkable that it is possible to build a function h that has the three properties previously noted. These properties imply that h essentially acts like a randomly selected compression function.

"These properties imply that h essentially acts like a randomly selected compression function."

Use Cases

It is instructive to describe some typical use cases:

- *Digital fingerprinting.* Hash functions construct effective digital fingerprints. If d represents a digital data structure, such as a document, then the message digest $h(d)$ is its fingerprint; the 2nd-pre-image-resistance property says it is infeasible to find a second data structure d' with the same fingerprint $h(d') = h(d)$.
- *Digital signatures.* Digital signatures extend digital fingerprinting by encrypting the hash value $h(d)$ of a data structure d under a private key.

- *Message authentication.* A hash function h used with a secret key K can be used to authenticate a message m. The idea is to create a tag $t = h(K \| m)$, where "$\|$" denotes string concatenation, which is sent with the message and verified by the receiver. (This does not quite work in practice, because the cascade construction introduces vulnerabilities hashing the last message block. Instead the tag is essentially computed as $h(K \| h(K \| m)))$. Because of pre-image resistance, it is infeasible for an attacker to create the same tag t unless he or she knows the key K, and because of collision resistance, the tag could have been created only by concatenating the message m to the key K.
- *Pseudo-random number generation.* One standard way to build a random number generator is to take a key K, usually called a seed, and to compute $h(K \| 0)$, $h(K \| 1)$, $h(K \| 2)$, with each digest representing a different random number. If h and K are carefully selected, it is infeasible to distinguish this, by any statistical test, from a stream of genuine random numbers.
- *Stream ciphers.* A stream cipher can be built by taking a hash function $\boldsymbol{h}$, and encrypting message $\boldsymbol{m_i}$ by $\boldsymbol{m_i} \rightarrow \boldsymbol{m_i} \oplus h(K, i)$, where "$\oplus$" denotes XOR, and decryption is $m_i \oplus h(K, i) \rightarrow (m_i \oplus h(K, i)) \oplus h(K, i) = m_i$.
- *Random oracles.* Random oracles can be thought of as specialized random number generators. They are used widely in cryptography, such as for randomizing public and private key encryption, in order to make them secure from arcane attacks.

Designing Hash Functions

"To meet the indistinguishability property, block ciphers are keyed, so a block cipher represents a family of permutations."

The standard approach to building a hash function is first to construct a *compression function* that operates on the input strings of a fixed length, and then to use the *cascade construction* to extend the compression function to strings of arbitrary length [5, 6].

Compression functions are usually built out of block ciphers. Recall that a block cipher is a pair of $\boldsymbol{D}$ and $\boldsymbol{E}$ functions, for decrypting and encrypting, respectively, that operate on strings of a particular length, called the *block size*. If the block size is $\boldsymbol{n}$-bits, then the encryption of an $\boldsymbol{n}$-bit string $\boldsymbol{s}$ is $\boldsymbol{E}(\boldsymbol{s})$, and its decryption is $\boldsymbol{D}(\boldsymbol{s})$. Every string can be encrypted or decrypted, and $\boldsymbol{E}(\boldsymbol{D}(\boldsymbol{s})) = \boldsymbol{D}(\boldsymbol{E}(\boldsymbol{s})) = \boldsymbol{s}$, meaning $\boldsymbol{E}$ (and $\boldsymbol{D}$), is a *permutation* of the set of all $\boldsymbol{n}$-bit strings. Cryptographers say a block cipher is *secure* if both $\boldsymbol{s} \rightarrow \boldsymbol{E}(\boldsymbol{s})$ and $\boldsymbol{s} \rightarrow \boldsymbol{D}(\boldsymbol{s})$ are indistinguishable from a randomly selected permutation. To meet the indistinguishability property, block ciphers are keyed, so a block cipher represents a family of permutations. A particular block cipher instance is selected by choosing a key $\boldsymbol{K}$. The resulting encryption and decryption instances are denoted $\boldsymbol{E_K}$ and $\boldsymbol{D_K}$. That is, for each choice of a key $\boldsymbol{K}$, $\boldsymbol{s} \rightarrow \boldsymbol{E_K}(\boldsymbol{s})$ (and $\boldsymbol{s} \rightarrow \boldsymbol{D_K}(\boldsymbol{s})$) behave like different randomly selected permutations.

Compression Functions

The compression functions for all the hash functions commonly used today are built in the following way:

1. Select a block cipher scheme ($\boldsymbol{E, D}$).
2. Define a compression function $\boldsymbol{c}(\boldsymbol{iv}, \boldsymbol{s}) = \boldsymbol{E}_s(\boldsymbol{iv}) \oplus \boldsymbol{iv}$.

Here $\boldsymbol{s}$ denotes a message of exactly $\boldsymbol{n}$-bits, and $\boldsymbol{iv}$ denotes an *initialization vector*. This recipe for $\boldsymbol{c}$ says to use $\boldsymbol{s}$ as the encryption key and $\boldsymbol{iv}$ as the data to be encrypted, and then to use XOR $\boldsymbol{s}$ with the encrypted result $\boldsymbol{E}_s(\boldsymbol{iv})$. The mapping $(\boldsymbol{iv}, \boldsymbol{s}) \rightarrow \boldsymbol{E}_s(\boldsymbol{iv}) \oplus \boldsymbol{iv}$ is called the *Davies-Meyer construction* [7] for $\boldsymbol{E}$. It is easy to show that a block cipher used in Davies-Meyer mode is collision-resistant, pre-image resistant, and 2nd pre-image resistant. $\boldsymbol{c}$ is called a compression function because it compresses $<\boldsymbol{iv},\boldsymbol{s}>$ into a new string $\boldsymbol{iv}'$ of exactly $\boldsymbol{s}$'s length. Other compression function constructions also exist: both Vortex and Skein use the Matyas-Meyer-Oseas [8] construction, c $(\boldsymbol{iv}, \boldsymbol{s}) \rightarrow \boldsymbol{E}_{iv}(\boldsymbol{s}) \oplus \boldsymbol{s}$, which is identical to Davies-Meyer, except it reverses the role of $\boldsymbol{iv}$ and $\boldsymbol{s}$.

*"Most padding schemes pad **s** with a single **1** bit followed by as many **0** bits as are necessary to bring the length to a multiple of **n**."*

The Cascade Construction

The *cascade construction* builds a hash function $\boldsymbol{h}$ from a compression function $\boldsymbol{c}$ with block size $\boldsymbol{n}$ as follows:

```
cascade ( $s$ )
pad ( $s$ ); $s_1 s_2 \ldots s_b \leftarrow s$; $iv_1 \leftarrow iv$; do $i = 1$ to $b \Rightarrow iv_{i+1} \leftarrow c(iv_i, s_i)$ od;
output $iv_{b+1}$
```

Every hash function based on a block cipher must define a padding scheme, because compression functions only operate on strings $\boldsymbol{s}$ of length $\boldsymbol{n}$ bits exactly. Most padding schemes pad $\boldsymbol{s}$ with a single **1** bit followed by as many **0** bits as are necessary to bring the length to a multiple of $\boldsymbol{n}$. The length of the unpadded string $\boldsymbol{s}$ is then encoded as an $\boldsymbol{n}$-bit integer and appended to defend against extension attacks.

Once padded, partition $\boldsymbol{s}$ into $\boldsymbol{b} = |\boldsymbol{s}|/\boldsymbol{n}$ blocks, each consisting of $\boldsymbol{n}$ bits ($|\boldsymbol{s}|$ denotes $\boldsymbol{s}'$s length in bits): $\boldsymbol{s_1 s_2 \ldots s_b} \leftarrow \boldsymbol{s}$.

Finally, beginning with a hash-function-specific initialization vector $\boldsymbol{iv}$, serially compute $\boldsymbol{c}(\boldsymbol{iv}_i, \boldsymbol{s}_i)$ for each block $\boldsymbol{s}_i$.

The cascade construction extends the collision resistance, pre-image resistance, and pre-image resistance properties from a compression function to a function operating on strings of arbitrary length [9]. The cascade construction is sometimes called the Merkle-Damgård construction after its inventors. This construction is intrinsically serial, as it needs to be able to detect problems, such as two blocks being exchanged.

"The cascade construction is sometimes called the Merkle-Damgård construction after its inventors."

Hashing Today

We just summarized the state of the art during the early part of this decade, prior to two significant publications. The first was by Andre Joux, who introduced the multi-collision attack. The second was by Xiaoyun Wang, where she described an attack, based on differential cryptanalysis, against all of the hash algorithms broadly used today.

"Joux's result says that by itself the cascade construction is too weak to serve as an adequate building block for constructing hash functions."

Suppose a hash function is built out of a compression function by using the cascade construction. Also suppose that someone has broken the collision resistance of the hash function; that is, they have discovered two distinct strings $s \neq s'$ so that $h(s) = h(s')$. Joux observed that it is easy to find many more collisions for little additional cost [1]. The source of the problem is that the cascade construction maintains too little state as it progresses from one invocation of the compression function to the next. Joux's result says that by itself the cascade construction is too weak to serve as an adequate building block for constructing hash functions.

"Wang demonstrated that a collision can be produced in SHA-1 at a cost of about 2^{61} operations. This caused upheaval in the cryptographic community."

Wang's attack [2], based on differential cryptanalysis, has a different flavor. Differential cryptanalysis is a technique to analyze block ciphers. Essentially, differential cryptanalysis follows a bit slice through the block cipher being analyzed, to characterize how it gets diffused. The goal of differential cryptanalysis is to identify bits leading to unusually high or low levels of diffusion. When such bits are identified, they can be used to recover bits of the encryption key. This can dramatically shrink the size of the key space, making brute force search realistic. As an example, differential cryptanalysis reduced the cost of key recovery attacks against the DES cipher from 2^{56} encryptions to about 2^{41}.

Wang showed that a differential attack could produce collision in message digests, thereby breaking the collision-resistance of the hash function producing them. Wang first demonstrated her attack against MD4, MD5, RIPE-MD, and SHA-0. This was viewed as a stunning result, but then she demonstrated that a collision can be produced in SHA-1 at a cost of about 2^{61} operations. This caused upheaval in the cryptographic community, raising the question as to whether we even understand what a hash function is.

"A lesson previously learned by the community is that contests have great efficacy in galvanizing technical consensus building."

The cryptographic community has vigorously debated hash design principles in the intervening years. The only clear consensus emerging from this debate is that we need a worldwide, focused project whose goal is to create a new generation of hash functions that defend against the new attacks. A lesson previously learned by the community is that contests have great efficacy in galvanizing technical consensus building. In 2007, NIST initiated an international competition to create a new hash standard [10]. Candidate submissions were due on October 31, 2008. Fifty-five algorithms were entered. In February of this year, NIST whittled down the list of candidates to 40, and from this it plans to select ten to fifteen first-round candidates by August 2009. NIST plans to select a set of finalist algorithms in 2010, then to announce the winner(s) in 2011.

NIST is widely influential in the creation of cryptographic standards worldwide, so it is a good sponsor for the competition. One of NIST's most important contributions to cryptography standards has been the creation of requirements for algorithms submitted to the competition. The competition requires that candidate algorithms provide the collision-resistance, pre-image-resistance, and 2nd-pre-image-resistance properties—and be free of any known intellectual property. Algorithms must support output block sizes of 128, 160, 224, 256, 384, and 512 bits. The rules encourage support for features outside the core properties, especially for parallelization. Submissions must be accompanied by a security rationale, to help establish confidence in the algorithms.

"Candidate algorithms provide the collision-resistance, pre-image-resistance, and 2nd-pre-image-resistance properties—and be free of any known intellectual property."

Some New Designs

Two of the candidates submitted to the NIST hash competition, Skein and Vortex, include contributions by Intel personnel. Both are among the forty entries remaining in the competition.

Skein

Skein was designed by Mihir Bellare (U.C. San Diego), Jon Callas (PGP Software), Niels Ferguson (Microsoft), Tadayoshi Kohno (University of Washington), Stefan Lucks (Bauhaus University-Mannheim), Bruce Schneier (British Telecom), Doug Whiting (Hi-Fn), and Jesse Walker (Intel Corporation). Skein produces message digests of any length from 1 to 2^{96} bytes. Skein has three major components: a new block cipher named Threefish, a replacement for the cascade construction named Unique Block Iteration (UBI), and an argument system extending Skein's domain of use beyond hashing.

The First Skein Component: the Threefish Block Cipher

Threefish is a *tweakable* block cipher [11], which means that a randomizer called a *tweak* is passed to the cipher with the key and data to encrypt. Skein uses the block offset from the start of the message as the Threefish tweak. The tweak addresses many deficiencies in the cascade construction and represents the major innovation in Skein.

"A randomizer called a tweak is passed to the cipher with the key and data to encrypt."

Threefish has three flavors: a 256-bit, a 512-bit, and a 1024-bit block size. The Threefish encryption key is the same size as the block size. The tweak is always 128 bits.

Threefish is a *product* cipher, meaning it is composed of *rounds*. Each round is a simple but weak encryption function. Threefish obtains security by piling round upon round: 72 rounds for Threefish-256 and for Threefish-512 and 80 rounds for Threefish-1024. The number of rounds represents a tradeoff between performance and security.

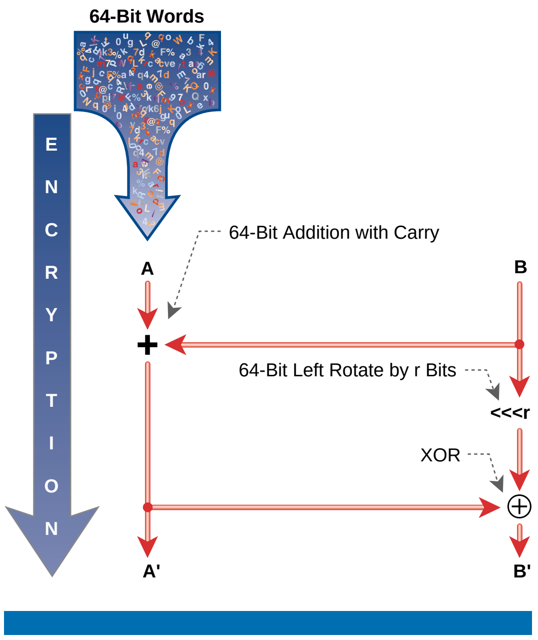


Figure 1: The Threefish MIX
Source: Intel Corporation, 2009

A Threefish round consists of a number of parallel MIX functions followed by a permutation, so that different blocks are mixed for different rounds. The MIX functions are made up of just three instructions—64 bit addition, left rotate, and XOR—to combine two 64-bit words ***A*** and ***B***, as depicted in Figure 1.

Threefish-256 splits its input into four 64-bit words, so each round consists of two parallel MIXes: Threefish-512 uses eight words with four parallel MIXes, and Threefish-1024 uses sixteen words with eight parallel MIXes. Figure 2 depicts the Threefish round structure for Threefish-512. The parallel MIXes efficiently exploit the super-scalar properties of modern processors. The rotation constants ***r*** were selected by a hill-climbing algorithm that maximized diffusion over randomly selected sets of rotation constants.

Threefish adds a *round* key every four rounds. Figure 2 depicts one of these additions. The Threefish round keys come from a *key schedule* inspired by Skipjack's key schedule [12]. Each Threefish round key is the same size as the plaintext data block, and each key depends on all the bits of both the encryption key and the tweak.

The Second Skein Component: Unique Block Iteration

Unique Block Iteration (UBI) mode replaces the cascade construction in Skein. UBI consists of four parts. First, UBI uses the Matyas-Meyer-Oseas construction, $(\boldsymbol{iv}, \boldsymbol{s}) \rightarrow \boldsymbol{E}_{\boldsymbol{iv}}(\boldsymbol{s}) \oplus \boldsymbol{s}$, to build a compression function *c* out of any block cipher. Second, UBI padding appends enough 0 bits to bring the length of the message being hashed to a multiple of the block size. Third, UBI constructs and passes the tweak to the block cipher. The UBI tweak is composed of two flags and of the message block offset from the beginning of the message in bytes. One of the flags is set on the first block, and the second flag is set for the final block. Finally, UBI computes its output just like the cascade construction, the only difference being the construction of the tweak:

```
UBI($iv, s$)
pad(s)
$t \leftarrow 0 \oplus$ start-flag; $iv_2 \leftarrow c(iv, s_1, t)$ ; $t$ = block-size;
do $i = 2$ to $b-1 \Rightarrow iv_{i+1} \leftarrow c(iv_i, s_i, t)$; $t$   $t$+block-size od;
$t \leftarrow t \oplus$ end-flag; output $c(iv_b, s_b, t)$
```

UBI uses the Matyas-Meyer-Oseas construction instead of Davies-Meyer. This converts attacks against a hash function from related key attacks to chosen plaintext attacks against the block cipher: the community understands more about defending against the latter than it does about defending against the former.

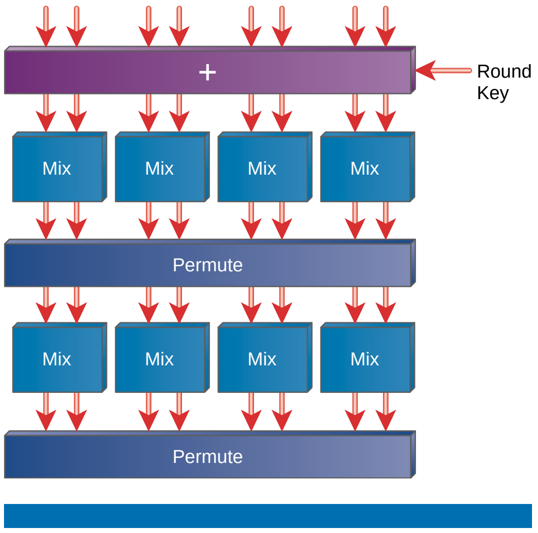


Figure 2: Two Rounds of Threefish-512
Source: Intel Corporation, 2009

The Third Skein Component: Skein Argument System

The Skein argument system extends the algorithm beyond *normal* hashing to application-specific or personalized hashing, message authentication, key derivation, pseudo-random number generation, stream ciphers, and tree (that is, parallelized) hashing.

Putting it Together in Skein

Skein instantiates UBI mode with the Threefish block cipher. The design's initialization vector is computed as the UBI-Threefish output of the configuration string "SHA-3." Skein first hashes a string ***s*** under UBI mode and ***iv*** to obtain an intermediate value. Skein uses the intermediate value as an ***iv*** to hash the integers 0, 1, 2, … under UBI, again to obtain the final output. Classical theory justifies the claim that Skein-***n*** (***n*** = 256, 512, or 1024) achieves ***n/2*** bits of security against collisions, and ***n*–1** bits of security against 1st and 2nd pre-image attacks—the best that can be achieved, theoretically. The *double hashing* under UBI mode also allows Skein to make additional, unusually strong claims, as follows:

*"The **double hashing** under UBI mode also allows Skein to make additional, unusually strong claims."*

- If Threefish is a pseudo-random permutation, then Skein can be used as a pseudo-random function, a secure key derivation function, a secure message authentication code, a secure stream cipher, and a secure pseudo-random number generator.
- If Threefish acts like an ideal cipher, then Skein cannot be differentiated from a random oracle.

The first claim says that Skein can be used naively in a broad range of applications that usually require great sophistication when constructed from classical hash functions. The second claim is a non-trivial result: it claims that Skein is structurally sound when Threefish is viewed as a black box; that is, the attacker is not allowed to utilize any knowledge about the internals of Threefish. This structural property means the security of Skein depends on the security of the underlying block cipher only. The best known attack against Threefish at this time breaks a 34-round variant (out of 72 rounds for full Threefish), which is superior to AES, whose 8-out of 10-round variant falls to attack.

In software Skein is one of the fastest unbroken algorithms ever devised: it runs at 6.1 clocks/byte on an Intel® Core™ Duo processor and requires no special hardware acceleration, such as an AES round instruction. This is twice as fast as the best software implementations of the current hashing standard. Skein also maintains a very small footprint for its in-memory state, allowing implementation in even constrained environments such as smart cards.

Vortex

Vortex is a family of hash functions developed by Michael Kounavis and Shay Gueron of Intel. A main strength of the Vortex design is that this hash function can achieve an ideal performance of 2.2-2.5 cycles per byte by using the AES round [14] and carry-less multiply instructions [15]. Such instructions have been announced for future Intel processors. Vortex is one of the fastest collision-resistant hashes known when running on future IA processors, outperforming SHA-1 (approx. 7 cycles/byte) by 3.18X, and outperforming SHA256 (approx. 19 cycles/byte) by 8.63X.

"Vortex is one of the fastest collision-resistant hashes known when running on future IA processors."

The Vortex family produces message digests of 224, 256, 384, and 512 bits, respectively. The main idea behind Vortex is to use well-known algorithms with very fast diffusion in a small number of steps. These algorithms also balance the cryptographic strength, that comes from iterating block cipher rounds with S-box substitution and diffusion, against the need to have a lightweight implementation with as small a number of rounds as possible. Vortex is built upon the following algorithms:

- The Rijndael round function, which performs very fast mixing across 32 bits, as a standalone operation, and 128 bits or 256 bits, if combined with at least one more round.
- A variant of Galois Field multiplication that mixes bits of different sets in a manner that is cryptographically stronger than many other simpler schemes.

"Vortex uses a variable number of Rijndael rounds with a stronger key schedule."

Vortex uses a variable number of Rijndael rounds with a stronger key schedule. The number of rounds is a tunable parameter. Rijndael rounds are followed by a variant of Galois Field multiplication to cross-mix between 128-bit or 256-bit sets. This transformation is not simple carry-less multiplication; rather, it combines bit reordering operations, XORs, and additions with carries. In this way, this variant of Galois Field multiplication achieves better diffusion than the straightforward carry-less multiplication between the 128-bit or 256-bit inputs; it is also a non-commutative operation, protecting against chaining variable swapping attacks.

Vortex uses the Enveloped Merkle-Damgård (EMD) construction to lift collision resistance, pre-image and 2nd pre-image resistance, pseudo-random oracle preservation, and pseudo-random function preservation from the underlying compression function to the hash function. To achieve its properties, the EMD construction first hashes the input string $\boldsymbol{s}$ under one initialization vector to get an intermediate value, and then it hashes the intermediate value under a second initialization vector to obtain a final result.

For Vortex-256, Gueron and Kounavis demonstrate that the number of queries required to find a collision with a probability greater or equal to 0.5 is at least $1.18 \cdot 2^{122.55}$ [13].

In summary, the Vortex compression function uses the Rijndael round function. Vortex-224 and Vortex-256 use Rijndael-128 rounds. Vortex-384 and Vortex-512 use Rijndael-256 rounds. AES uses the Rijndael-128 round function. For the remainder of this section, $\tilde{A}_{\boldsymbol{K}}(\boldsymbol{X})$ denotes a block cipher based on the Rijndael round function that encrypts $\boldsymbol{X}$ by using key $\boldsymbol{K}$. $\boldsymbol{V}_{\boldsymbol{M}}^{(\boldsymbol{A})}(\boldsymbol{A}, \boldsymbol{B})$ is a multiplication-based merging function.

The Vortex-block algorithm is the Vortex compression function. This algorithm incorporates two repetitions of an algorithm called *Vortex-sub-block*. The first repetition of Vortex-sub-block accepts as input the chaining variable $\boldsymbol{A_i} \,||\, \boldsymbol{B_i}$ and two least-significant input block words $\boldsymbol{W_{4i}}$, $\boldsymbol{W_{4i+1}}$ of the message being hashed. It returns an intermediate value for the chaining variable $\boldsymbol{A} \,||\, \boldsymbol{B}$. The second repetition of Vortex-sub-block accepts as input the intermediate value of the chaining variable $\boldsymbol{A} \,||\, \boldsymbol{B}$ and two most-significant input block words $\boldsymbol{W_{4i+2}}$, $\boldsymbol{W_{4i+3}}$. It returns an update on the chaining variable $\boldsymbol{A_{i+1}} \,||\, \boldsymbol{B_{i+1}}$.

With the exception of the last sub-block (discussed later), the algorithm for processing a Vortex-sub-block is as follows:

Vortex sub-block ($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{W_0}, \boldsymbol{W_1}$)

```
    // $W_0$ is the first word of the current sub-block to be processed
    $A \leftarrow \tilde{A}_A(W_0) \oplus W_0;\ B \leftarrow \tilde{A}_B(W_0) \oplus W_0;\ A \,||\, B \leftarrow V_M^{(A)}(A, B)$

    // $W_1$ is the second word of the current sub-block to be processed
    $A \leftarrow \tilde{A}_A(W_1) \oplus W_1;\ B \leftarrow \tilde{A}_B(W_1) \oplus W_1;\ A \,||\, B \leftarrow V_M^{(A)}(A, B)$
    output $A \,||\, B$
```

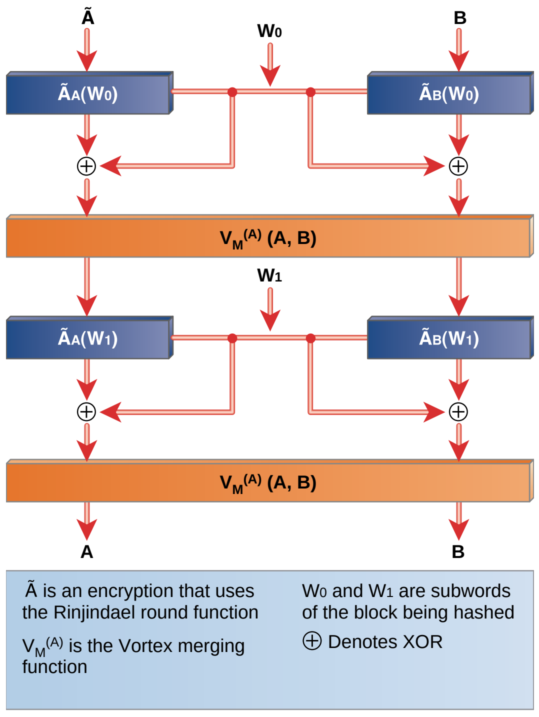

Ã
B
W0
ÃA(W0)
ÃB(W0)
VM(A) (A, B)
W1
ÃA(W1)
ÃB(W1)
VM(A) (A, B)
A
B
Ã is an encryption that uses the Rinjindael round function
VM(A) is the Vortex merging function
W0 and W1 are subwords of the block being hashed
⊕ Denotes XOR

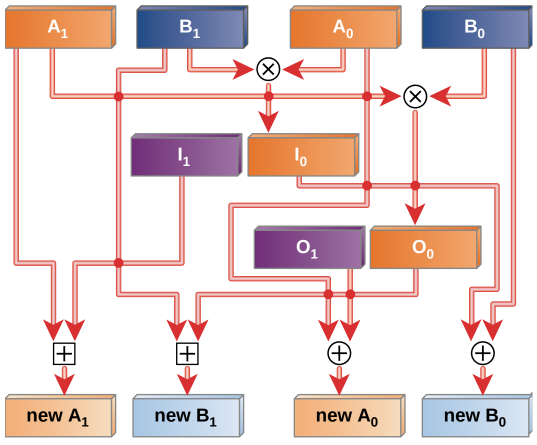
Figure 3: Vortex Sub-Block Structure
Source: Intel Corporation, 2009

The structure of the Vortex sub-block is shown in Figure 3. There are four instances of the transformation $\tilde{A}_K(x)$ in the Vortex sub-block. Each instance is wrapped by using a feed-forward provided by the Matyas-Meyer-Oseas construction to make the transformation non-reversible. The first two instances process input word W_0. The other two instances process the input word W_1. W_0 is the least-significant word of the current sub-block to be processed. Instances of $\tilde{A}_K(x)$ that accept the same input word process a different variable from among A, B. Each instance treats its input variable A or B as a key and treats its input word, which is one from W_0 or W_1 as plaintext, as that is the norm in a Matyas-Meyer-Oseas construction.

The Vortex merging function $\boldsymbol{V_M^{(A)}(A, B)}$ operates as follows:

```
$V_M^{(A)}(A, B)$
        $A_1\,A_0 \leftarrow A\,;\ B_1\,B_0 \leftarrow B$
        $O \leftarrow A_0 \otimes B_1\,;\ I \leftarrow A_1 \otimes B_0$
        $I_1\,I_0 \leftarrow I\,;\ O_1\,O_0 \leftarrow O$
        output $B_1 \boxplus I_1 \,||\, B_0 \boxplus O_0 \,||\, A_1 \oplus O_1 \,||\, A_0 \oplus I_0$
```

where $\boxplus$ is ordinary 64-bit addition, and $\otimes$ denotes carry-less multiplication.



A_0, B_0: least significant words of chaining variables A, B
A_1, B_1: most significant words of chaining variables A, B
I_0: least significant word of inner product I
I_1: most significant word of inner product I
O_0: least significant word of outer product O
O_1: most significant word of outer product O
$\boxplus$: add with carry
$\oplus$: XOR
$\otimes$: carry-less multiply

Figure 4: Vortex Merging Function
Source: Intel Corporation, 2009

"Vortex uses carry-less multiplication because it is easier to make analytical assertions about the collision resistance and pre-image resistance of the hash."

The Vortex merging function (as shown in Figure 4) ensures that the bits of ***A*** impact the bits of ***B*** and vice versa. In fact, each bit of one variable affects a significant number of the bits of the other variable in a non-linear manner. This makes the design better than a straightforward XOR or other simple mathematical operation.

Carry-less multiplication is the default configuration of Vortex. The reason why Vortex uses carry-less multiplication by default is because it is easier to make analytical assertions about the collision resistance and pre-image resistance of the hash. In another configuration, Vortex uses integer multiplication. An integer multiplier increases the performance of the hash (not all processor architectures have a carry-less multiplier) and also increases the non-linearity of merging; however, it also makes the security of the scheme more difficult to prove.

The last Vortex sub-block is different. It repeats the sequence of Matyas-Meyer-Oseas transforms and merging several times. The total number of times every bit is diffused over all bits of the hash is determined by the number of sequences of Rijndael rounds and merging found in the last Vortex sub-block; this is another tunable parameter of the hash.

Design Rationale and Comparison

In this section, we summarize and compare the design rationale for both algorithms.

Skein Design Rationale

The Skein team had a number of goals:

- *Design for simplicity.* Simple designs, rather than complex ones, are easier to optimize and analyze for security flaws.
- *Maximize security per clock.* Performance trumps security in practice, so extract as much security as possible from each cycle. In particular, it is important to use only the simplest instructions that have been highly optimized on every platform. Exploit the super-scalar behavior of modern processors and maximize the diffusion from each operation. Through a series of experiments the Skein team discovered that simpler round functions with more rounds optimize both performance and security.
- *Achieve high performance and easy implementation on all processors.* Building in a performance advantage for one processor family over another is a disadvantage in a public competition.

The Skein team made a number of critical design decisions when designing the algorithm:

- *Base the design on a block cipher.* This is the most conservative security choice. The community knows better how to analyze designs based on block ciphers. Moreover, there is a well-established theory on how to turn a block cipher into a hash function.
- *Build your own block cipher.* The Skein team obtained better performance and a simpler security analysis with a new cipher of the correct block size instead of adapting an existing narrow-block cipher.

- *Make the block cipher tweakable.* This introduces great flexibility in designing a new mode at no cost.
- *Replace the cascade construction with something better.* The cascade construction has many known defects; the tweak allows UBI to offer provable security, and it suffers from none of the cascade construction's problems.
- *Use Matyas-Meyer-Oseas construction instead of Davies-Meyer.* Attacks against Matyas-Meyer-Oseas-based compression functions are chosen plaintext attacks; attacks against Davies-Meyer-based compression functions are related key attacks. The community has more experience defeating chosen plaintext attacks than related key attacks.
- *Do not use table lookups.* Most cipher designs use a table called an S-box. Lookups in the S-box enable side-channel attacks on software implementations, where the encryption key can be read by monitoring the power, timing, or EMI of the processor. Threefish is not subject to these attacks since it incorporates no table lookups.
- *Build in three different internal state sizes and allow output of any size.* This allows flexibility in the level of security available across different use cases.
- *Change the design when necessary.* The design should be changed when doing so allows for simpler security proofs.

Vortex Design Rationale

The Vortex team had a number of goals:

- *Design for performance.* Vortex uses algorithms that are implemented by using dedicated instructions in future IA processors. These are instructions for AES round computation (AES-NI) and carry-less multiplication (GFMUL-NI). The Vortex team argues that such instructions will become a trend in the industry.
- *Maximize security per clock.* Like Skein, Vortex extracts as much security as possible from each cycle. In particular, it uses independent AES round operations in each block. Such operations can potentially be completed in a single clock in future processors. It also introduces parallelism in the design of its compression function (two AES rounds and their key schedules can be executed in parallel); and finally, it maximizes the diffusion from each operation. Diffusion is supported by the AES round algorithms (S-box substitution, ShiftRows, MixColumns) as well as by the multiplication stage that follows.
- *Achieve high performance and easy implementation on future processors.* The Vortex team believes that instructions for AES round computation and carry-less multiplication will become a trend in the industry. This is because (i) several hardware vendors including IBM and Sun are either implementing or researching them; (ii) even processors for embedded systems now include AES hardware; and (iii) there is a precedence in the industry that good instruction sets are widely adopted (for example, SSE instructions).

The Vortex team made a number of critical design decisions when designing the algorithm:

- *Base the design on a well-known and secure block cipher.* Vortex uses an AES round as a building block. AES is a well-studied block cipher, and the AES round operation offers very good mixing across 32 bits, as a standalone operation, and 128 bits if repeated several times.
- *Strengthen the AES key schedule.* Hashing is a one-way operation so additions with carries are permitted in the design of a hash function. Vortex strengthens the AES key schedule by adding round constants with carries and performing S-box substitution across all round key bytes.
- *Combine the outputs of two parallel AES transformations by using carry-less or integer multiplication.* Multiplication is a highly non-linear operation and can be used for destroying bit differentials.
- *Replace the cascade construction with something better.* Vortex uses the EMD construction and a tweak value to preserve the pseudo-random function and the pseudo-random oracle properties.
- *Use Matyas-Meyer-Oseas construction instead of Davies-Meyer.* This is similar to the rationale of the Skein team. The community has more experience defeating chosen plaintext attacks than related key attacks.
- *Use Rijndael S-boxes instead of table lookups.* Rijndael S-boxes have a special structure that allows them to be implemented by using combinatorial logic as opposed to table lookups. Thus, side-channel attacks can be averted.

Comparison of the Skein and Vortex Designs

It is now possible to highlight some similarities and differences between the Skein and Vortex designs. We begin with the similarities.

Similarities Between the Skein and Vortex Designs

Both designs are based on block ciphers, and they both use the Matyas-Meyer-Oseas construction to convert the related key attacks against the deployed hash function designs into chosen plaintext attacks. The cryptographic community has more experience defeating chosen plaintext attacks than related key attacks.

Both designs use a flavor of the cascade construction to paste together the hash of different blocks output by a compression function, and they both *double hash* the final output; that is, the output from the cascade construction is rehashed to become the final output. Both do this to address vulnerabilities that arise from processing the final block with a construction such as Davies-Meyer or Matyas-Meyer-Oseas, within the cascade construction.

Differences Between the Skein and Vortex Designs

The differing design approaches reflect the differing skill sets of the two teams. The Skein team members were skilled in designing block ciphers. In contrast, the Vortex team members did not design a new block cipher, but instead used an existing one. The Skein design allows its security claims to be derived from the security of a block cipher. The Vortex design effort emerged from the need to demonstrate a secure hash function by using the new AES round and carry-less multiplication instructions for future IA processors, which the Vortex team members designed.

Skein uses significantly more rounds (72/72/80) than Vortex, stemming from the different design decisions made by each team. The Skein team's experiments indicated diffusion per clock is maximized by numerous simple rounds. The Vortex designers instead were motivated by performance; they believed the best performance is achieved by using fewer rounds, based on very powerful diffusion primitives.

Skein's support for a hash value of any length from 1 to 2^{64} bytes allowed the team to prove the property that Skein cannot be *differentiated from a random oracle* if Threefish acts like an ideal cipher. There are two ways to think about any algorithm: (1) as a monolithic black box, where you have no knowledge of any of the algorithm's internal parts, and (2) as presented in this article, where we know the details of the algorithm's internal structure. By saying that Skein cannot be differentiated from a random oracle, therefore, we mean that it is impossible, even in principle, for Skein to construct any statistical test that exploits differences in the two views. This means that Skein is structurally sound and that its security depends only on the security of Threefish. Vortex's final output is of a fixed length, but the second hash allows it to act like a fixed-length random oracle.

"Intel has been at the forefront, with two independent and radically different submissions to the international NIST hash competition."

None of the Skein components use table lookups such as S-boxes, so it is more difficult to launch side-channel attacks on Skein than on specific implementations of Vortex. Vortex avoids these attacks by relying on a hardware logic implementation for the AES S-boxes.

Summary

In this article we reviewed the theory of hash functions, the state of knowledge about them, and some of Intel's contributions to this field of research. Hash functions are basic building blocks that are central to cryptography's mission, but recent attacks by Joux and Wang have undermined our confidence in classical constructions. Because of this insecurity, there has been a wave of new research into hash functions, and Intel has been at the forefront, with two independent and radically different submissions to the international NIST hash competition.

References

[1] A. Joux. "Iterated Collisions on Iterated Hash Functions." *Crypto 2004*, *Lecture Notes in Computer Science* (LNCS) 3621, Springer-Verlag, Berlin, 2005.

[2] X. Wang, Y. Yin, and H. Yu. "Finding Collisions in the full SHA-1." *Crypto 2005*, LNCS 2947, Springer-Verlag, Berlin, 2004.

[3] A. Lenstra, X. Wang, and B. de Berger. "Colliding X.509 Certificates." At *http://eprint.iacr.org*

[4] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[5] R. Merkle. "Secrecy, authentication, and public key systems." Stanford University Ph.D. thesis, 1979. Available as "Technical Report No. 1979-1," *Information Systems Laboratory*, Stanford University, Palo Alto, California, 1979.

[6] I. Damgård. "A Design Principle for Hash Functions." *Crypto 1989*, LNCS 435, Springer-Verlag, Berlin, 1989.

[7] R. Winternitz. "A secure one-way hash function built from DES." In *Proceedings of IEEE Symposium on Security and Privacy*, 1984.

[8] S. Matyas, C. Meyer, and J. Oseas. "Generating strong one-way functions with cryptographic algorithms." *IBM Technical Bulletin,* 27, 1985.

[9] J. Black, P. Rogaway, and T. Shrimpton. "Black box analysis of block-cipher-based hash functions from PGV." *Crypto 2002*, LNCS 2442, Springer-Verlag, Berlin, 2002.

[10] The National Institute of Standards and Technology. US Government agency sponsor for competition on next-generation hash design. At *http://csrc.nist.gov*

[11] M. Liskov, R. Rivest, and D. Wagner. "Tweakable Block Ciphers." *Crypto*, LNCS 2442, Springer-Verlag, Berlin, 2002.

[12] National Security Agency. *Skipjack and KEA Specifications*. May 29, 1998. At *http://csrc.nist.gov*

[13] M. Kounavis and S. Gueron. "Vortex: A new Family of One-Way Hash Functions Based on Rijndael Rounds and Carry-less Multiplication." At *http://eprint.iacr.org*

[14] S. Gueron. "Advanced Encryption Standard (AES) Instructions Set." At *http://software.intel.com*

[15] S. Gueron and M. Kounavis. "Carry-Less Multiplication and its Usage for Computing the GCM Mode." At *http://software.intel.com*

Author Biographies

Jesse Walker is an Applied Cryptographer in Intel's Communication Technology Laboratory. He first identified vulnerabilities in the 802.11 WEP protocol. He also served as editor for the 802.11i standard. He joined Intel in the Shiva acquisition. He has a Ph.D. degree in Mathematics from the University of Texas. Jesse received a 2004 Intel Achievement Award for influencing China's telecommunication policy. His e-mail is jesse.walker at intel.com.

Michael E Kounavis is a Senior Research Scientist working with Intel Labs. Michael is responsible for conducting research on novel digital arithmetic and cryptographic algorithms with the aim of accelerating a wide range of client, server, and networking applications. Michael is a co-inventor of the CRC32 SSE4 instruction of the Intel® Core™ i7 architecture used for iSCSI CRC generation. He is also a co-recipient of an Intel Achievement Award (2008) for his work on the AES instructions. His e-mail is michael.e.kounavis at intel.com.

Shay Gueron is an Intel Principal Engineer. He is the security architect of the CPU Architecture Department in the Mobility Group, at the Israel Development Center. His interests include applied security, cryptography, and algorithms. He is also an Associate Professor at the Department of Mathematics of the Faculty of Science at the University of Haifa in Israel. Shay was one of the co-recipients of the Intel Achievement Award (2008) for his work on the AES instructions. His e-mail is shay.gueron at intel.com.

Gary Graunke is a Senior Staff Architect for cryptography at Intel, where he has contributed to various content protection technologies for PCs and digital television. Prior to Intel, he contributed to parallel programming languages and their compilation, synchronization primitives, and scalable decision support databases and sorting algorithms at Sequent Computer. His earlier work included numerous compilers for various programming languages and one of the earlier (1970) shell/editor programs for timesharing on Univac mainframe computers. He holds 30 patents. Gary holds a B.S. degree in Computer Science with Distinction from the University of Wisconsin at Madison. His e-mail is gary.graunke at intel.com.

Copyright

ENHANCED PRIVACY ID: A REMOTE ANONYMOUS ATTESTATION SCHEME FOR HARDWARE DEVICES

Contributors

Ernie Brickell
Intel Corporation

Jiangtao Li
Intel Corporation

Index Words

Anonymity
Privacy
Cryptography
Trusted Computing
Remote Attestation

Abstract

Enhanced Privacy ID (EPID) is a cryptographic scheme that enables the remote authentication of a hardware device while preserving the privacy of the device owner. A hardware device with an EPID private key embedded can prove to a remote party that it is a valid device, certified by the hardware manufacturer, without revealing its identity and without the verifier being able to link authentication attempts. In this article, we discuss hardware authentication and present several usage examples, such as secure e-commerce and digital drivers' licenses. We then show that EPID can be used for hardware authentication securely and privately. We discuss several revocation capabilities of EPID that allow flexible revocation in different scenarios. For instance, in signature-based revocation, it is possible to revoke an EPID private key that signed a message, even though the identity of the key is not known. We show how these revocation methods can be used while protecting the rights of the user. We also compare EPID with other possible privacy techniques.

Introduction

"A hardware device wants to prove to a verifier that it is a genuine hardware device."

Consider the following problem. A hardware device (for example, a mobile device, a graphics chip, a trusted platform module, a processor package, or a smart card) wants to prove to a verifier that it is a genuine hardware device manufactured by a certified hardware manufacturer. The easiest way to prove that it is the genuine article is for the verifier to read the serial number to the hardware manufacturer to verify that it is indeed the device in question. Each hardware device is assigned a unique serial number that is inscribed on the body of the device by the manufacturer. The problem with this solution is its limited application: the verifier needs to physically have the device in order for this kind of authentication to work. In many cases, a piece of hardware needs to be authenticated remotely. Remote hardware authentication is the main focus of this article.

"Remote hardware authentication is the main focus of this article."

A possible solution to the problem of remote hardware authentication is for the hardware manufacturer to assign each device a unique device certificate. More specifically, each device could be assigned a unique public and private key pair. The hardware manufacturer certifies the device by issuing a cryptographic certificate to the device public key. When the hardware device needs to be verified, it sends its device certificate to the verifier along with a signature signed with its private key. The verifier can then use the hardware manufacturer's public key to verify the device certificate and then use the device's public key in the certificate to verify the signature. This solution is secure, as long as the device can protect its private key, since only hardware devices made by the original manufacturer have valid device certificates.

This certificate approach is also scalable, as the device manufacturers can issue as many device certificates as they want. However, issuing a device certificate raises a privacy concern, because the device certificate is used to uniquely identify the device. The verifier, therefore, can use the device certificate to trace a device and the associated authentication activities.

In this article, we introduce a new cryptographic scheme called Enhanced Privacy ID (EPID) for remote, anonymous authentication of a hardware device. Using EPID, a hardware device can prove to a verifier remotely that it is a valid device, certified by the hardware manufacturer, without revealing its identity and without the verifier being able to link multiple authentication attempts made by the device.

"We introduce a new cryptographic scheme called Enhanced Privacy ID for remote, anonymous authentication of a hardware device."

Conceptually, an EPID scheme can be viewed as a special digital signature scheme. Unlike traditional digital signature schemes, one public key in the EPID scheme corresponds to multiple private keys. There are three types of entities in an EPID scheme: issuer, members, and verifiers. In our context, the issuer is the hardware manufacturer, the member is a hardware device made by the manufacturer, and the verifier could be software on the host, a server on the Internet, or another hardware device. The issuer creates an EPID public key and issues a unique EPID private key to each member. Each member can use this private key to digitally sign a message, and the resulting signature is called an EPID signature. The verifier can use the public key to verify the correctness of a signature, that is, to verify that the EPID signature was indeed created by a member in good standing with a valid private key. The EPID signature, however, does not reveal any information about which unique private key was used to create the signature.

In the rest of this article, we first present our design requirements. We then go on to describe the application of remote hardware authentication. We continue with an overview of EPID and explain how we construct the EPID protocol and how we handle revocations. We conclude by comparing EPID with other related techniques.

"There are three types of entities in an EPID scheme: issuer, members, and verifiers."

Prerequisites and Design Requirements

We first formalize the remote hardware authentication problem, then describe the prerequisites for the problem, and end with our design requirements.

Remote Hardware Authentication Problem

To do remote authentication securely, cryptographic keys need to be used. There are three entities involved in a remote authentication scenario: the issuer, members, and verifiers. The issuer is a hardware manufacturer who creates a group. A member is a hardware device manufactured by the issuer. A member can join or leave the group.

"The EPID signature does not reveal any information about which unique private key was used to create the signature."

"The member should have secure storage to store the private key and have a trusted execution environment to use the key to perform the membership proof."

When a member joins the group as it is manufactured, the issuer issues a private key to the member. When the member leaves the group, the issuer revokes the private key of the member. Leaving the group is a rare event. It occurs only when the private key of the member (the hardware device) has been extracted from the hardware device, and the issuer has to revoke the membership of the device, or in other words, revoke the private key. A verifier is an entity that wants to know that a hardware device is a member of the group. The remote hardware authentication is an interaction between a member and a verifier. The member uses its private key to prove to the verifier that it is a valid member of the group and has not been revoked.

Prerequisite

To have a secure remote hardware authentication scheme, the member (that is, the hardware device) must have a good protection system for its private key. In other words, the member should have secure storage to store the private key and have a trusted execution environment to use the key to perform the membership proof.

"Only a member in good standing could perform the membership proof successfully."

If an attacker can easily extract the key information from a member, then there is no way to do remote hardware authentication securely, as the attacker can always use the extracted private key to perform the membership proof before the key is revoked. This means that the verifier cannot tell whether the proof comes from the attacker or from a real hardware device.

Security Requirements

The basic security requirement is straightforward; that is, only a member in good standing could perform the membership proof successfully. In other words, if the prover is not a member of the group, then its proof of membership would be rejected by the verifier. This property should hold unless a private key has been removed from a member and has not yet been revoked, or unless a problem considered computationally infeasible has been solved.

Privacy Requirements

"The membership proof must be anonymous and unlinkable."

In a remote hardware authentication scheme, the membership proof must be anonymous and unlinkable. In addition, the private key of each member should be unknown to the issuer. More specifically, these are the required privacy properties:

- Given a membership proof, the verifier or the issuer cannot identify the actual prover, that is, cannot extract any identifiable information about the member from the proof. This is known as the anonymity property.
- Given two membership proofs, the verifier or the issuer cannot tell whether the proofs are generated by one member or by two different members. This is known as the unlinkability property.
- The issuer does not know any of the private keys of the members. Therefore, the issuer does not have a database of all the members' private keys.

The unlinkability requirement is optional. In some applications, the verifier may require the membership proofs from a member to be linkable. Linkable proofs help to prevent a member from abusing the anonymity requirement. For example, suppose a verifier is a key-provisioning server that provisions a key to each member (that is, each hardware device). This verifier wants to make sure that each member is provisioned with only one key. Suppose that an adversary is able to extract a private key from a member device. If the remote hardware authentication scheme has the property of anonymity but not unlinkability, then the verifier would issue many keys to this adversary by using this one compromised member key. Then if the verifier found that one of the provisioned keys had been abused, he or she would be able to revoke it but would not be able to revoke all of the other keys that this adversary had obtained from the one compromised member key. Privacy issues with this use can be controlled, since the member needs to obtain the provisioned key from this verifier only once, and this usage can be unlinkable to any usage with any other verifier.

Revocation Requirements

A remote hardware authentication scheme must handle revocation. In general, when a hardware device is manufactured, it joins the group. Even if the ownership of the hardware device changes or the device is stolen, it is still a valid, authentic hardware device; thus, it is still in the group and does not need to be revoked.

““When a hardware device is manufactured, it joins the group.”

Only if the private key has been extracted from the secure storage of the hardware device, does the member have to be revoked. Given the prerequisites mentioned earlier, the issuer assumes that the member's (the hardware device's) private key is well protected. Thus, the revocation of a member is a rare event. However, the issuer needs to have the ability to revoke a member from the group if needed.

The first revocation requirement is that the revocation of a group member should have minimum impact on the rest of the group members.

The second revocation requirement is that if an extracted private key is known to the issuer, then the issuer should be able to revoke that private key.

The third revocation requirement is that if a private key is used in a transaction, and it is later discovered that the key used in that transaction had been extracted, then this key should be revocable, even if it is not known. An example of such a case would be if a transaction was to provision a verifier key into a hardware device, and this verifier key was later shown to be extracted, the issuer may conclude that the private key of the hardware device has been corrupted and should be revoked.

Note that if an attacker extracts a private key from a hardware device and never uses the key, the key can probably never be revoked. On the other hand, if the attacker never uses the extracted private key to forge or emulate a hardware device, there is no damage to the hardware authentication scheme.

“Revocation of a member is a rare event.”

Application of Hardware Authentication

In this article, we present a new cryptographic scheme called Enhanced Privacy ID (EPID) that satisfies the security, privacy, and revocation requirements just discussed. Before we discuss EPID, however, we first discuss two applications of EPID. We first describe remote anonymous attestation and then discuss the application in digital drivers' licenses and identity cards.

Remote Anonymous Attestation

Why are we interested in the problem of remote hardware authentication? The answer lies in the fact that in many scenarios a verifier wants to know whether a request comes from an authentic hardware device or from a software emulator.

"We first describe remote anonymous attestation and then discuss the application in digital drivers' licenses and identity cards."

Consider the following remote attestation example, depicted in Figure 1 as a conversation between a client and a service provider. A client platform is running a hardware-based trusted execution environment, based on a smartcard, or on a Trusted Platform Module (TPM). The trusted execution environment includes functionalities, such as secure code execution, secure data storage, and secure key generation. The platform requests a resource from a service provider, such as a Digital Rights Management (DRM) key. The service provider needs to determine whether the platform can protect its resource. The platform can do a remote attestation by sending the service provider a measurement of its computation environment, for example, the platform can send its hardware and software configuration. The attestation needs to be combined with a remote hardware authentication, that is, one signed by the hardware's private key. The logic of such an authentication is that an attacker can easily forge a measurement, but an attacker cannot compute a valid signature without knowing a valid hardware private key. A hardware authentication scheme satisfying the design requirements, outlined in the previous section, can provide both security and privacy for the remote attestation.

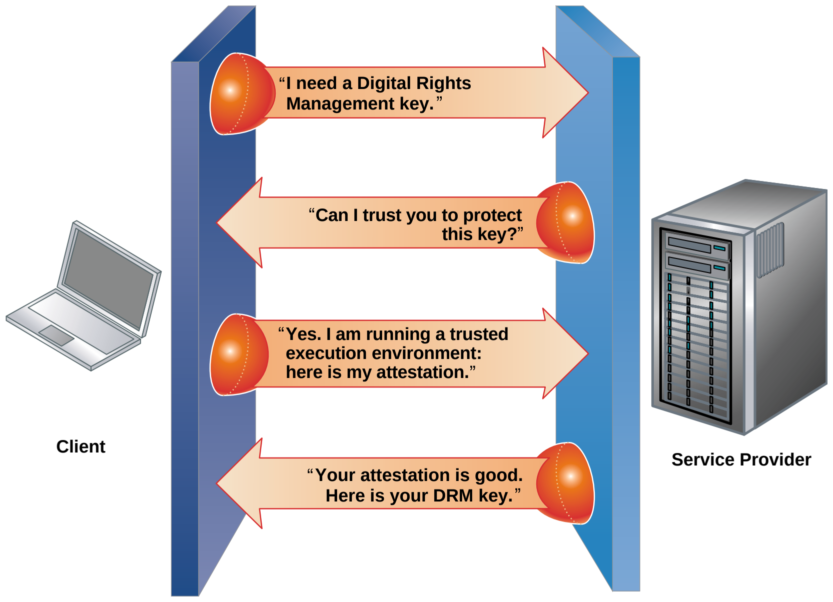

"I need a Digital Rights Management key."
"Can I trust you to protect this key?"
"Yes. I am running a trusted execution environment: here is my attestation."
"Your attestation is good. Here is your DRM key."
Client
Service Provider

Figure 1: An Example of Remote Attestation
Source: Intel Corporation, 2009

The remote attestation problem was first introduced in the domain of a TPM, a small hardware module integrated into a platform, such as a laptop or a desktop. A direct anonymous attestation (DAA) scheme was developed by Brickell, Camenisch, and Chen [4] for remote authentication of a TPM, while preserving the privacy of the TPM. The DAA scheme was adopted by the Trusted Computing Group, an industry standardization body that aims to develop and promote an open industry standard for trusted computing hardware and software building blocks, and it was included in TPM specification version 1.2 [11].

Note that the EPID scheme presented here is an extension of the DAA scheme but has more revocation capabilities. Our EPID scheme has broader applicability beyond the remote attestation of a TPM. Let us look at two concrete applications of anonymous attestation.

Secure E-Commerce and On-line Banking

We now describe how EPID can be used for secure on-line banking. On-line banking is increasingly popular and provides great convenience to end users. However, the security of on-line banking is a concern, not only to end users but also to the banks. If the end user runs a platform that has a trusted execution environment and trusted I/O, the end user can conduct business in a relatively secure environment. However, the bank does not know whether the user is running in a secure environment. An anonymous attestation from the user's platform to the bank would give the bank more confidence that the transaction is secure.

"An anonymous attestation from the user's platform to the bank would give the bank more confidence that the transaction is secure."

For example, if a bank user performs some high-volume transactions, the bank wants to make sure that the transactions are properly authorized. If the user runs a trusted execution environment, the user can use the EPID scheme to anonymously attest to the bank so that the bank can give a token to the platform for future transactions.

The bank would know that the token was being secured in a trusted execution environment. In later transactions, the user enters a password into the trusted execution environment that unlocks the token so that the bank can authenticate the user's environment. This assures the bank of the authenticity of the transaction.

Content Protection

Here we describe how EPID can be used to protect content. An on-line media server provides high-definition media content to clients. In order to download media content, each client needs to first download a Digital Rights Management (DRM) key from the server in order to decrypt the content. Before sending a DRM key to the client, the server wants to know whether the client can protect its DRM key. If the client has a hardware-based trusted execution environment and has a unique EPID private key embedded, it can use EPID to perform an anonymous attestation to the media server. After the attestation, the media server is convinced that the client is indeed running a trusted execution environment and is not in fact running a software emulator.

Observe that in this example, if an attacker corrupts one EPID private key from a hardware device, the attacker may not publish the private key publicly. Instead, the attacker may use the compromised private key to obtain a DRM key from the media server. If the DRM key is found to be compromised on the Internet (for example, in ripper software), it can be traced back to the EPID private key that links it to the transaction that was used for provisioning the DRM key. Consequently, the issuer can revoke the compromised private key, based on the transaction of the key. This is an example in which the media server may wish to be assured that it issues only one DRM key for each EPID private key. This is accomplished through making the requests for DRM keys linkable to each other. But these requests would not be linkable to any other transactions.

"If an attacker corrupts one EPID private key from a hardware device, the attacker may not publish the private key publicly."

Drivers' Licenses and Identity Cards

Various governments are considering including machine-readable information on drivers' licenses and identity cards. In the case of drivers' licenses, the machine-readable portion (for example, the bar code or magnetic strip) of the license is readable to anyone with a license reader. Unfortunately, such an approach raises serious privacy concerns, as personal information in the magnetic strip or bar code can be easily gathered and then sold without the owner's consent—potentially leading to identity theft.

Encrypting the machine-readable portion of the license has also been proposed. Such a practice poses significant key management challenges; the decryption must be available to authorized parties only.

"The media server may wish to be assured that it issues only one DRM key for each EPID private key."

We describe how EPID can be applied to drivers' licenses. Each license has an embedded smart card chip that can store and process information. A card reader is used to communicate with the license. The license is assigned a unique private key when it is issued by the government department that oversees the licensing of automobile drivers. It can be used for various purposes without violating the user's privacy. The smart card license would be able to prove to the reader that it is a valid license and that it has not been revoked, suspended, reported lost, and so forth. The smart card accomplishes this by using the proof of membership protocol; in this way the identity of the license is not revealed.

Each government agency would have multiple groups capable of issuing of licenses. During the process of proving its validity to a reader, a license reveals which license group it is in, and it reveals whether or not it is a valid license in good standing. It does not, however, reveal which license it is within that license group.

"During the process of proving its validity to a reader, a license reveals which license group it is in, and it reveals whether or not it is a valid license in good standing."

Using the EPID scheme, the membership proof is unlinkable. This means that if a license is used at a restaurant, for example, and it is valid and issued to someone of legal age, when that same license is used again at the same restaurant the next night, the restaurant owners would not be able to tell that it was the same license that was being presented. The restaurant owner is only acquainted with certain information: the validity of the license and that the patron is of legal age.

Overview of EPID

In our EPID scheme, there are three types of entities: issuer, members, and verifiers. There are two revocation lists: a list of corrupted private keys, denoted as PRIV-RL, and a list of signatures made from suspected extracted keys, denoted as SIG-RL. An EPID scheme has the following operations:

1. *Setup*. The issuer creates a public key and an issuing private key. The issuer publishes and distributes the public key to everyone (that is, to every member and every verifier).
2. *Join*. This is an interactive protocol between an issuer and a member, the result of which is that the member obtains a unique private key.
3. *Sign*. Given a message *m* and a SIG-RL, a member creates an EPID signature on *m* by using its private key.
4. *Verify*. The verifier verifies the correctness of an EPID signature by using the public key. The verifier also checks that the key used to generate the signature has not been revoked in PRIV-RL or SIG-RL.

Figure 2 depicts the interaction flows between the issuer, a member, and a verifier.

"There are two revocation lists: a list of corrupted private keys and a list of signatures made from suspected extracted keys."

Zero-knowledge Proofs

In our EPID scheme, we use zero-knowledge proofs of knowledge [10] extensively. In a zero-knowledge proof system, a prover proves the knowledge of some secret information to a verifier such that (1) the verifier is convinced of the proof and yet (2) the proof does not leak any information about the secret to the verifier. In this article, we use the following notation for proof of knowledge of discrete logarithms. For example,

"The verifier is convinced of the proof and yet the proof does not leak any information about the secret to the verifier."

$$\mathbf{PK}\,\{ (x) : y_1 = g_1^{x}\, y \wedge\ y_2 = g_2^{x} \}$$

denotes a proof of knowledge of integer x such that $y_1 = g_1^{x}$ and $y_2 = g_2^{x}$ hold, where x is known only to the prover, and g_1, y_1, g_2, y_2 are known to both the prover and verifier. In the above equation, 'PK' stands for proof of knowledge and '$\wedge$' stands for logical conjunction.

Proof of knowledge protocols can be turned into signature schemes by using the Fiat-Shamir heuristic [9]. In our EPID scheme, we develop several efficient zero-knowledge proof protocols for proving the knowledge of a valid EPID private key. In addition, we use an efficient zero-knowledge proof protocol developed by Camenisch and Shoup [8] for proving the inequality of discrete logarithms of two group elements y_1, y_2 to base z_1, and z_2, respectively, denoted as

$$\mathbf{PK}\,\{ (x) : y_1 = z_1^{x} \wedge y_2 \neq z_2^{x} \}.$$

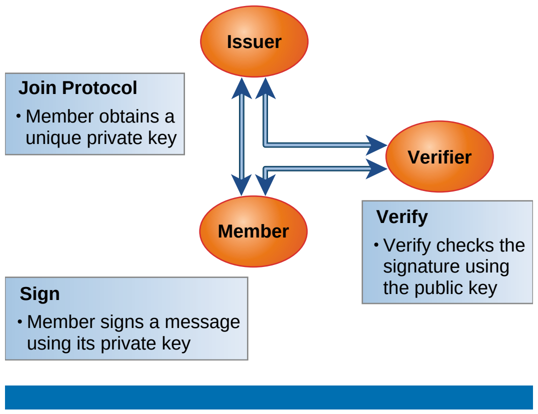

Issuer
Join Protocol
• Member obtains a unique private key
Verifier
Member
Verify
• Verify checks the signature using the public key
Sign
• Member signs a message using its private key

Figure 2: Basic EPID Scheme
Source: Intel Corporation, 2009

"The member needs not only to prove that it has a valid membership credential, but also to prove that it constructs the (B, K) pair correctly, all in zero-knowledge."

Overview of Our Construction

We begin with a high-level overview of our construction. In our scheme, each member chooses a unique membership key *f*. The issuer then issues a membership credential on *f* in a *blind* fashion such that the issuer does not acquire knowledge of the membership key *f*. The membership key and the membership credential together form the private key of the member. To sign a signature, the member proves in zero-knowledge that it has a membership credential on *f*. To verify a group signature, the verifier verifies the zero-knowledge proof.

In addition, each member chooses a base value B and computes $K = B^f$. This (B, K) pair serves the purpose of a revocation check. We call B the base and K the pseudonym. To sign a signature, the member needs not only to prove that it has a valid membership credential, but also to prove that it constructs the (B, K) pair correctly, all in zero-knowledge.

In EPID, there are two options to compute the base B: the random base option and the name base option.

- *Random base option. B* is chosen randomly each time by the member. Under the decisional Diffie-Hellman assumption, no verifier can link two EPID signatures based on the (B, K) pairs in the signatures.
- *Name base option. B* is derived from the verifier's basename; for example, *B* = Hash (verifier's basename). Note that in this option, the value *K* becomes a pseudonym of the member with regard to the verifier's basename, as the member will always use the same *K* in the EPID signature to the verifier.

We first explain how membership can be revoked based on a compromised private key. Given a private key that has been revealed to the public, the issuer extracts the membership key *f* from the private key and inserts *f* into the private-key-based revocation list PRIV-RL. The issuer then distributes PRIV-RL to all the verifiers. Given an EPID signature, any verifier can check whether it was created with the corrupted private keys in PRIV-RL as follows:

"Membership can be revoked based on a compromised private key."

Let (B, K) be the base-pseudonym pair in the EPID signature. The verifier can check that $K \neq B^{f'}$ for every f ' in PRIV-RL. If there exists an f ' in PRIV-RL, such that $K = B^{f'}$, it means that the signature was created with a revoked private key. Therefore the verifier can reject the signature.

We now explain how membership is revoked, based on a transaction that a member was involved in. We call this kind of revocation signature-based revocation. Suppose a member's private key has been compromised by an attacker and has been used in some transaction. If the issuer has collected enough evidence to show that the private key used in the transaction was corrupted, the issuer can identify the EPID signature in the transaction and revoke the key, based on the signature. To do this, the issuer extracts the (B, K) pair from the signature and inserts the pair into the signature-based revocation list SIG-RL. The issuer then distributes the SIG-RL to all the verifiers. Before a member performs the membership proof, the verifier sends the latest SIG-RL to the member, so that the member can prove that it did not perform those transactions. More specifically, the member proves that it is not revoked in SIG-RL, by proving that, in zero-knowledge,

$$\mathbf{PK}\ \{(f) : K = B^{f} \wedge K' \neq B'^{f}\}$$

for each (B', K') pair in SIG-RL. If the zero-knowledge proof holds, the verifier is convinced that the member has not conducted those transactions and that membership has not been revoked.

"We call this kind of revocation signature-based revocation."

Sketch of EPID Scheme

We have developed two EPID schemes, one from the strong RSA assumption [7] and the other from bilinear maps [6]. In this article, we briefly sketch the EPID scheme from bilinear maps. (The full scheme can be found in [6]).

Let us first review some background on bilinear maps. Let $G1$ and $G2$ be two multiplicative cyclic groups of prime order p. Let g_1 be a generator of $G1$, and g_2 be a generator of $G2$. We say e: $G1 \times G2 \rightarrow GT$ is an admissible bilinear map function, if it satisfies the following properties:

For all $u \in G1$, $v \in G2$, and for all integers a, b, equation $e(u^a, v^b) = e(u, v)^{ab}$ holds. The result of $e(g_1, g_2)$ is a generator of GT. There exists an efficient algorithm for computing $e(u, v)$ for any $u \in G1$, $v \in G2$.

Our EPID scheme is derived from Boneh, Boyen, and Shacham's group signatures scheme [2] and has the following operations:

Setup: The issuer does the following:

1. Chooses $G1$ and $G2$ of prime order p and a bilinear map function $e : G1 \times G2 \rightarrow GT$.
2. Chooses a group $G3$ of prime order p with generator g_3.
3. Chooses at random $g_1, h_1, h_2 \in G1$ and $g_2 \in G2$.
4. Chooses a random $r \in [1, p-1]$ and computes $w = g_2^r$.
   The public key is $(g_1, g_2, g_3, h_1, h_2, w)$ and the issuing private key is r.

Join: The join protocol is an interactive protocol between the issuer and a member as follows:

1. The member chooses at random f and y' from $[0, p-1]$ and computes $T = h_1^f h_2^{y'}$.
2. The member sends T to the issuer and performs the following proof of knowledge to the issuer: $PK\{ (f, y') : T = h_1^f h_2^{y'}\}$.
3. The issuer chooses at random x and y'' from $[0, p-1]$ and computes $A = (g_1 T h_2^{y''})^{1/(x+r)}$.
4. The issuer sends (A, x, y'') to the member.
5. The member computes $y = y' + y'' (mod\ p)$.
   The member's private key is (A, x, y, f).

Note that given a valid private key (A, x, y, f), the following equation satisfies:

$e\ (A, g_2^x w) = e\ (g_1 h_1^f h_2^y, g_2)$.

Sign: Let (A, x, y, f) be the member's private key. The member does the following:

1. If the random base option is used, the member chooses B at random from $G3$.
2. If the name base option is used, the member computes B = Hash (verifier's basename).

3. Computes $K = B^f$.

4. Computes the following zero-knowledge proof
$PK\{(A, x, y, f) : e(A, g_2^x w) = e(g_1 h_1^f h_2^y, g_2) \wedge K = B^f\}$
This essentially proves that the member has a valid EPID private key issued by the issuer.

5. Computes the following zero-knowledge proof

   $PK\{(f) : K = B^f \wedge K' \neq B'^f\}$

   for each (B', K') pair in SIG-RL. This step proves that the member has not been revoked in SIR-RL; that is, the member did not create those (B', K') pairs in SIG-RL.

6. Converts all the above zero-knowledge proofs into a signature by using the Fiat-Shamir heuristic [9].

Verify: Given the public key, PRIV-RL, SIG-RL, and an EPID signature, the verifier does the following:

1. If the random base option is used, the verifier verifies that B is an element in $G3$.

2. If the name base option is used, the verifier verifies that B = Hash (verifier's basename).

3. Verifies that K is an element in $G3$.

4. Verifies the following proof

   $PK\{(A, x, y, f) : e(A, g_2^x w) = e(g_1 h_1^f h_2^y, g_2) \wedge K = B^f\}$

   This step verifies that the member has a valid EPID private key.

5. Verifies that $K \neq B^{f}$'for each f' in PRIV-RL. This step verifies that the member has not been revoked in PRIV-RL.

6. Verifies the following zero-knowledge proof

   $PK\{(f) : K = B^f \wedge K' \neq B'^f\}$

   for each (B', K') pair in SIG-RL. This step verifies that the member has not been revoked in SIG-RL.

Comparison with Other Techniques

There are other techniques to remotely authenticate hardware, and in this section we review these techniques and compare them with our EPID scheme.

Public Key Infrastructure (PKI)

Each hardware device has a unique public and private key pair as well as a device certificate. To authenticate hardware by using PKI, the device simply shows its certificate to the verifier along with a signature created by using the device's private key. As mentioned previously, this PKI approach does not satisfy the privacy requirement.

"This PKI approach does not satisfy the privacy requirement."

Direct Anonymous Attestation (DAA)

DAA was designed for anonymous attestation of TPM [4, 5]. DAA satisfies all the design requirements of remote hardware authentication; however, it has limited revocation capabilities compared to those of EPID. In the DAA scheme, there are two options for a balance between linkability and revocation. If the random base option is used, that is, a different base is used every time a DAA signature is performed, then any two signatures by a device are unlinkable, but revocation only works if the corrupted device private key has been revealed to the public. If a device has been compromised, but its private key has not been distributed to the verifiers (for example, if the corrupted device's private key is still under the control of the adversary), the corrupted TPM cannot be revoked. If the name base option is used, then any two signatures produced by a device, using the same base, are linkable. Thus, if the verifier determines that a device private key, used in a signature, has been compromised, that verifier can revoke that key locally; that is, the verifier can reject all future signatures generated by that private key, without knowledge of the compromised private key. However, the verifier cannot tell if a different verifier uses a different name base to revoke that private key, because when a different name is used, the revoked key cannot be identified. Furthermore, the name-based option does not safeguard privacy, because the verifier can link the transactions.

"DAA satisfies all the design requirements of remote hardware authentication; however, it has limited revocation capabilities compared to those of EPID."

Group Signatures (GS)

A group signature scheme [1, 2] has similar properties to those of the EPID scheme. In a group signature scheme, an issuer creates a group public key and issues unique private keys to each group member. Each group member can use the private key to sign a message, and the resulting signature is called a group signature. The verifier can verify a group signature by using the group public key. Unlike EPID, group signature schemes have an additional property called traceability. This property enables the issuer to open any group signature and identify the actual group member who created the signature. In other words, a group signature is anonymous to the verifiers but not to the issuer. Again, as compared to this scheme, EPID keeps the identity of the group member from the issuer.

"In a group signature scheme, an issuer creates a group public key and issues unique private keys to each group member."

Pseudonym System (PS)

The pseudonym system [3], designed by Brands, can also be used for remote hardware authentication. In the pseudonym system, the display of a credential is anonymous by virtue of the fact that efficient zero-knowledge proof techniques are used for proving relations among committed values. To use the pseudonym system for hardware authentication, each hardware device obtains a credential from the issuer and uses the pseudonym credential for proof of membership. However, a credential in that system is linkable for multiple displays. To be unlinkable, a hardware device has to get multiple credentials from the issuer and use one credential at a time. This approach has limited application for hardware authentication, as the hardware device may never be able connect back to the issuer (the device manufacturer) once it has been produced. Thus, it cannot maintain the unlinkable property by continuing to get new credentials from the issuer.

"To use the pseudonym system for hardware authentication, each hardware device obtains a credential from the issuer and uses the pseudonym credential for proof of membership."

Summary

In Table 1, we summarize a comparison between different approaches to the remote hardware authentication problem. The EPID scheme is the only scheme that satisfies all the design requirements mentioned earlier.

Properties	**PKI**	**DAA**	**Group Signatures**	**Pseudonym System**	**EPID**
Unique Public Key	Yes	No	No	No	No
Unique Private Key	Yes	Yes	Yes	Yes	Yes
Anonymous	No	Yes	Yes	Yes	Yes
Unlinkable	No	Yes	Yes	No	Yes
Issuer Untraceable	No	Yes	No	Yes	Yes
Private-Key Revocation	Yes	Yes	Yes	Yes	Yes
Signature Revocation	Yes	No	No	Yes	Yes

Table 1: Approaches to Remote Hardware Authentication
Source: Intel Corporation, 2009

References

[1] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. "A practical and provably secure coalition-resistant group signature scheme." In *Advances in Cryptology—Crypto*, Volume 1880 of *Lecture Notes in Computer Science*, pages 255–270, 2000.

[2] D. Boneh, X. Boyen, and H. Shacham. "Short group signatures." In *Advances in Cryptology—Crypto*, Volume 3152 of *Lecture Notes in Computer Science*, pages 41–55, 2004.

[3] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA, 2000.

[4] E. Brickell, J. Camenisch, and L. Chen. "Direct Anonymous Attestation." In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 132–145, 2004.

[5] E. Brickell, L. Chen, and J. Li. "A New Direct Anonymous Attestation Scheme from Bilinear Maps." In *Proceedings of 1st International Conference on Trusted Computing, Volume 4968 of Lecture Notes in Computer Science*, pages 166–178, 2008.

[6] E. Brickell and J. Li. "Enhanced Privacy ID from Bilinear Pairing." *Cryptology ePrint Archive*, Report 2009/095, 2009.

[7] E. Brickell and J. Li. "Enhanced Privacy ID: a Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities." In *Proceedings of the 6th ACM Workshop on Privacy in the Electronic Society*, pages 21–30, 2007.

[8] J. Camenisch and V. Shoup. "Practical Verifiable Encryption and Decryption of Discrete Logarithms." In *Advances in Cryptology—Crypto*, Volume 2729 of *Lecture Notes in Computer Science*, pages 126–144, 2003.

[9] A. Fiat and A. Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems." In *Advances in Cryptology—Crypto*, Volume 263 of *Lecture Notes in Computer Science*, pages 186–194, 1987.

[10] O. Goldreich, S. Micali, and A. Wigderson. "Proofs that Yield Nothing but their Validity." *Journal of the ACM*, Volume 38(3), pages 690-728, 1991.

[11] Trusted Computing Group. "TCG TPM Specification 1.2," 2003, ***http://www.trustedcomputinggroup.org***

Author Biographies

Ernie Brickell is a Senior Principal Engineer and Chief Security Architect at Intel, responsible for the design and analysis of Intel security architectures. He has numerous designs for security and privacy features that have been incorporated into Intel products. He is a co-inventor of the EPID protocol, which provides privacy-preserving authentication. He has published many papers on cryptographic protocols and cryptanalysis. He was the founding editor-in-chief of the *Journal of Cryptology*. He has worked on privacy protocols, secret sharing, authentication theory, electronic cash, fast implementations of cryptographic algorithms, and the design and cryptanalysis of several public key cryptographic systems. He holds an M.S. degree in Computer Science and a Ph.D. degree in Mathematics from the Ohio State University. He previously worked at Sandia National Laboratories, Bellcore, and CertCo. His e-mail is ernie.brickell at intel.com.

Jiangtao Li is a Security Architect at Intel. He obtained a Ph.D. degree in Computer Science from Purdue University and a B.S. degree in Computer Science from the University of Science and Technology of China. He joined Intel in 2006 and has participated in designing several security features that have been incorporated into Intel products. He is one of the inventors of the EPID protocol, which provides privacy-preserving authentication. He has published more than twenty papers on applied cryptography, information security, and privacy-enhancing technologies. His e-mail is jiangtao.li at intel.com.

Copyright

NETWORK SECURITY: CHALLENGES AND SOLUTIONS

Contributors

Linden Cornett
Intel Corporation

Ken Grewal
Intel Corporation

Men Long
Intel Corporation

Marc Millier
Intel Corporation

Steve Williams
Intel Corporation

Index Words

Network
Protocol
Security
Offload
IPsec

Abstract

Virus and worm attacks are on the rise, along with the exploitation of equipment vulnerabilities and the use of social engineering to attack corporate networks. The resulting data breaches allow the theft and misuse of personal and corporate data at a cost of millions of dollars. In addition, network traffic is increasing due to factors such as high-speed networks, service-orientated cloud computing, remote storage, and ever-increasing numbers of client devices. This increase in traffic, combined with the increased demand for data accessibility, places a greater emphasis on the need for network and protocol security. Industry-standard network admission control (NAC) frameworks provide one-time authentication and authorization, but fail to protect the subsequent data flows.

In this article we first explore the motivation for protocol security solutions, ones that are capable of protecting data in transit. We describe several solutions, at various layers of the open systems interconnection (OSI) stack, addressing several key network and platform threats. We compare and contrast the benefits and shortcomings of each. We then describe several new network security offerings from Intel and discuss these in detail. We conclude with a summary of future requirements for protocol security.

Introduction

Network security has become more important than ever in our lives as more and more of our private and valuable information moves around on data networks. These data include not merely our financial information, as in the case of an on-line order that includes a credit card number, but they also, increasingly, include medical information that travels among doctors, hospitals, and insurance providers; and personal information that travels among friends and to or from our employers. It is clear that unprotected networks leak data both out of the network as well as into it. It is no longer enough to protect data only at the computer and while in storage. We must protect data while in transit.

"Unprotected networks leak data both out of the network as well as into it."

We are members of the community at Intel that is addressing the needs for security in the network by actively researching new solutions to network security issues, by participating in industry standardization efforts that allow network security solutions to be adopted widely, and by introducing new products that implement these solutions.

Why Network Security?

We are not saying anything new when we say that security in the information technology (IT) infrastructure is a current and growing concern. By security we mean preventing unauthorized access to data while ensuring accurate authorized access to data, without interference. By IT infrastructure we mean the IT network within the enterprise firewall, as well as the outreach from the enterprise IT network across the Internet cloud.

"What started out ten and more years ago as hobby-hacking has morphed into for-profit theft."

The situation is stark. What started out ten and more years ago as hobby-hacking into primitively protected IT assets has morphed into for-profit theft of information from corporations, governments, and private citizens that threaten the real and perceived reliability and safety of the Internet and all the networks connected to it.

It is not necessary to cite a list of network security breaches to make this point. They are familiar to most readers of this article, as is the fact that these breaches can cost victim enterprises tens of millions of dollars; moreover, they cause smaller-scale but personally damaging losses to private Internet users.

This issue of the *Intel Technology Journal* focuses on data security, both at the platform level and also at the network level. In this article we focus on network and protocol security—specifically, the protection of *bits in flight*. There are several families of network threats that should be stopped before they ever reach the data and applications they are attacking. They include the following:

*"We focus on network and protocol security—specifically, the protection of **bits in flight**."*

Eavesdropping, or unauthorized access to data as they flow through the network. Before the advent of switches, networks were breached by monitoring Ethernet traffic flowing through a hub. With the advent of switches, eavesdropping has become only slightly more difficult. Many switches allow listening to network traffic by putting one of the switch ports into *monitor* mode. Monitor mode is a legitimate means of trouble-shooting a network, but it also makes the network vulnerable to anyone with access to the switch. It is also possible to eavesdrop on fiber networks by creating a bend in a multi-mode fiber carrying network traffic. The bend allows enough light to escape the fiber, thus making it possible to eavesdrop on the network. Wireless networks have become predominant in accessing the Internet and intranet by using the wireless medium as the first hop. Historically, wireless networks afforded little security and could easily be snooped and spoofed by someone within range of the wireless access point (AP). In the last few years, a whole suite of wireless security standards have been introduced and widely deployed; they offer data authenticity as well as data confidentiality to wireless connections, from a given device to immediate Layer 2 neighbors, such as an AP. These standards are defined by IEEE. One of the most predominant wireless security standards is 802.11i, with numerous other 802.11 derivative standards already defined or in the process of being defined. These standards offer additional security services on wireless networks.

"It is also possible to eavesdrop on fiber networks by creating a bend in a multi-mode fiber carrying network traffic."

"Enterprises today employ network appliances that separate the outside, untrusted area of the network from the inside, trusted portion of the network."

Inserting malicious software into the network communications stack. The software that supports data communication in any of the several devices in any data path on the Internet is typically organized into layers. Each layer performs a different function or transforms the data as they pass through the layered software stack. If an attacker can manage to get a piece of malicious software inserted amongst these layers, the attacker may be able to eavesdrop or even inject malicious software into other systems in the communications path.

Illegitimate access to data masquerading as legitimate access. Legitimate users access data legitimately. If a hacker can trick a protected network into granting access, or piggy-back onto otherwise legitimate access, the hacker can have the same access as a legitimate user.

Network Defense

The defense against these threats is handled today by employing numerous security technologies, including physical and network access control (NAC) and anti-virus services on the platform and within the network. In the network, enterprises today employ network appliances at some boundary of the enterprise IT infrastructure to separate the outside untrusted area of the network, including the Internet, from the inside, trusted, often physically secured, portion of the network. These appliances detect and prevent intrusion from the Internet, scan incoming packets for viruses and other malware, and defend against denial of service attacks. This *defense by appliance* is working less effectively for several reasons. Firstly, many attacks come from within the defensive perimeter, from sources that should be able to be trusted, including internal employees and contractors. Secondly, it is more and more common to open ports through the perimeter defenses to provide for access to certain types of traffic streams, such as HTTP, FTP, and so on. Alas, attacks can be designed to penetrate these holes in the wall.

"The unprotected portion of the network between defense and assets is getting thinner."

As a result of these intrusions, the IT-protected perimeter is being moved closer to the assets, that is, the servers and the storage that it is meant to protect. More and more access, even from within the enterprise, is being routed through intrusion detection and prevention appliances, and the unprotected portion of the network between defense and assets is getting thinner. The logical extension of this is to define the perimeter right at the sheet metal around the servers themselves. This would mean mounting an in-depth defense of each and every server in the enterprise. Such a solution is not practical in today's computing world, given the protection burden already on enterprises. Compute resources diverted from productive work and into defensive work would place a bigger burden on each server.

One alternative to reducing the defensive work burden on each server is to build some level of protection into the communications protocols themselves, virtually allowing the data to protect themselves as they move around the network within the enterprise and across the Internet. In a sense, partial protection of this type is in wide use today. Virtual private networks (VPNs), for example, create a protective tunnel in which data are encrypted as they move between enterprise sites and between enterprise sites and remote hosts operating in the Internet. VPNs have been used effectively for years, but they still do not protect data flowing unprotected within the IT defensive perimeter, and they are often inconvenient to use for users of remote hosts.

"One alternative to reducing the defensive work burden on each server is to build some level of protection into the communications protocols themselves."

End-to-End Network Security

In this article we discuss protocol-based protection mechanisms that can provide end-to-end (E2E) protection of the data connections. Protection can be designed to terminate at various levels of the open systems interconnection (OSI) stack, depending on the requirements and scope of the protection and the value of the assets to be protected in flight. Some of these mechanisms are already in use, if not necessarily wide use, today. These include SSL/TLS protocols, commonly used to protect remote banking and commerce applications. All of these protocol security mechanisms also have their issues. These uses, benefits, and issues are the focus of this article.

Protocol Security

Protocol security is a generic term that is used to describe cryptographic services offered to network data packets. Data flow within a given network can take many forms and can be differentiated by the services being provided within the network. These can range from communications protocols that manage the network services, ad-hoc messaging between different nodes in the network, establishing distinct sessions and flows between two or more nodes within the network for the purposes of communicating pieces of data between these nodes, as well as a number of other network or end-host tasks. Many of these services can be mapped to different layers in the OSI reference model.

The OSI model abstracts out the network architecture into multiple layers where each layer performs a logical function and interacts with layers below and above it. The higher layers in this model rely on the services provided by the lower layers, with a guarantee of what these services are, without having to understand how these services may be provided. An example of this model is seen in how the transport layer TCP protocol provides connection-orientated services to a higher layer, while relying on network services from the IP layer below it. The IP layer, in turn, relies on the data link layer and physical layers below it to provide further services. These services may be dependent on the underlying communications medium, without needing any information on that underlying medium.

"Protection can be designed to terminate at various levels of the open systems interconnection (OSI) stack."

"Three examples are link layer security (Linksec) operating at Layer 2, IP layer security (IPsec) operating at Layer 3, and transport layer security (TLS) operating at Layer 5 and above."

"Security protocols are typically divided into two components: control channel and data channel."

In the context of protocol security, we describe three specific examples of the security offered at different layers of the OSI model. These are link layer security (Linksec) operating at Layer 2, IP layer security (IPsec) operating at Layer 3, and transport layer security (TLS) operating at Layer 5 and above. Although other security standards, such as 802.11i, are also available to offer protection to wireless connections, these are not described in detail in this article, as we primarily focus on higher layer E2E security offerings, such as IPsec.

What these protocol security solutions, or simply security protocols, have in common is that they are typically divided into two components: control channel and data channel. The control channel provides the negotiation and agreement on using a given security protocol, together with the associated attributes for that protocol. The data channel employs the negotiated attributes in the control channel to protect any subsequent communications, dependent on the scope of the negotiated policy and the security protocol employed, between the set of nodes that negotiated the security protocol. The control channel negotiates various attributes, including which data to protect: Layer-2 MAC addresses, Layer-3 IP addresses, or Layer-4+ sessions based on ports. The control channel also provides information on how to protect these data. Included in this information are the version of a given protocol to use that can impact packet formatting, the agreed-upon mode of operation, the different modes that may be supported by different security protocols (for example, encapsulating security protocol (ESP) [1] versus authentication header (AH) [2] in IPsec), the cryptographic algorithms to employ, and some miscellaneous attributes, such as replay protection, lifetime of the security association (SA), and identifiers that are carried in the packet to allow the recipient to map the security provided in the packet to a given negotiated policy. In contrast, the data path provides an implementation of how to provide the negotiated services, such as packet formatting and the cryptographic services.

The cryptographic services typically fall into these categories: data confidentiality and data authenticity (also known as data integrity). Optionally, replay protection may also be provided (which is protocol dependent) to ensure that previously processed packets are not processed again, due to retransmissions or resubmissions by an adversary who is trying to disrupt communications and realize malicious intent. These cryptographic services in the data path typically leverage symmetric cryptographic algorithms for efficiency. Some examples of such algorithms include the advanced encryption standard (AES), HMAC-SHA1, 3DES, and GMAC. Each specific algorithm may have different attributes for the security services provided, the performance criteria, as well as for associated cost, all of which need to be considered when using these algorithms within different security protocols.

With these commonalities in mind, we look at some examples of the different security protocols employed today. The relative positions of the various network security protocols are shown in Figure 1.

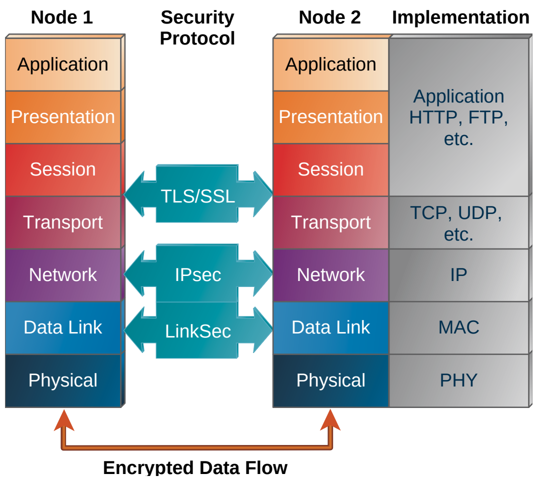


Figure 1: Security Protocols, the OSI Model, and Implementations
Source: Intel Corporation, 2009

Linksec Protocol

Linksec is employed at Layer 2 of the OSI stack and protects Layer-2 frames. Linksec is an IEEE standard, as defined by IEEE 802.1AE [3] and 802.1X-Rev [4]. 802.1AE (also known as Media Access Control Security or MACsec) provides the data path protection, while 802.1X-Rev (at least part of it) provides the control channel handshake for 802.1AE. Linksec is used to provide protection in wired networks and is logically and functionally equivalent to the IEEE wireless standard 802.11i, as well as other associated 802.11 specifications. Because this is a Layer-2 (MAC) security protocol, it provides security services on a hop-by-hop (HxH) basis, thereby protecting the Layer-2 MAC header and the associated payloads. This means that Linksec cannot provide any E2E guarantees for data traffic, but instead protects access to a given network for each hop.

"Linksec is used to provide protection in wired networks and is logically and functionally equivalent to the IEEE wireless standard 802.11i, as well as other associated 802.11 specifications."

At each hop of the data path, a secured Linksec session must terminate, at which point the packet is validated for integrity and deciphered, before it is allowed to go on to the next hop. The next hop may or may not offer Linksec protection, based on the negotiated policy for that particular hop. Because Linksec is HxH, it can be independently and incrementally deployed within the network for each hop, without it having an impact on what is happening at a previous or subsequent hop. Furthermore, because this is a Layer-2 HxH protocol, it allows intermediate network appliance devices, such as a firewall, an intrusion detection and intrusion prevention system (IDS/IPS), and any network monitoring, diagnostic and auditing tools to function unhindered, by connecting between adjacent hops of a Linksec secured connection. These devices need not be aware of the security provided on the previous or subsequent hops, and they can operate on clear text data within two adjacently secured nodes to offer valuable network service—as they do today. On any given hop, Linksec may be employed between two adjacent nodes or in a group scenario, where multicast and broadcast communication are employed. The data channel attributes of a Linksec session (MACsec) are negotiated via the control channel handshake, by using 802.1X-Rev (or alternatively, the attributes may be configured manually or communicated via a separate proprietary protocol).

IPsec

In contrast to Linksec, IPsec is an E2E security protocol. IPsec operates at Layer 3 (IP layer) of the OSI model and provides E2E data authenticity and optional data confidentiality. IPsec is an IETF-defined set of specifications and supports two protocols, ESP and AH. ESP provides data authenticity and, optionally, data confidentiality. AH provides data authenticity only. Other optimizations, such as data compression (IPCOMP), may also be negotiated and employed with IPsec to reduce the amount of data transmitted over the network. ESP and AH provide the data channel functions for IPsec, and the control channel handshake is provided by a separate protocol called the Internet Key Exchange (IKE) [5]. Today, AH is rarely used, due to the requirement of providing data integrity protection for a whole IP packet, including the IP header. Such a requirement results in the integrity value being invalidated in a network address translation (NAT) [6, 7] environment, where IP and ports (or IP or ports) may be mapped to externally visible values that differ from those employed internally within a given domain and host, or within a given domain or host.

"Because this is a Layer-2 HxH protocol, it allows intermediate network appliance devices, such as a firewall, to function unhindered."

"IPsec also supports two modes of operation: tunnel and transport."

In addition to supporting the two protocols, IPsec also supports two modes of operation: tunnel and transport. Tunnel mode is typically used in VPNs and remote access scenarios, where a given device can *tunnel* into a given network (such as the corporate network), while physically residing outside of that network. VPNs are typically constructed from a client to a VPN gateway, or by connecting one VPN gateway to another VPN gateway in order to securely connect two physically separate network domains and make them appear as a single, logical network. For network communication purposes, the VPN allows remote nodes to appear as if they are physically located on the target network.

Because IPsec operates at Layer 3 of the OSI model, it is media independent and can therefore be employed over any underlying physical media, from wireless networks to traditional Token Ring networks, to Ethernet networks, and to future optical networks. The main benefit of employing IPsec is that it provides E2E data authenticity assurances for any upper-layer protocol built on top of IP, thus making it generic in protecting all network layer traffic. The IPsec specifications support both IPv4- and IPv6-based security.

"The main benefit of employing IPsec is that it provides E2E data authenticity assurances for any upper-layer protocol built on top of IP."

The Secure Sockets Layer and Transport Layer Security Protocol

Secure sockets layer (SSL) and transport layer security (TLS) are other E2E security protocols, but they are limited to providing security services to TCP payloads only. SSL is the first generation of this protocol, and TLS is the second generation. For all intents and purposes, the two protocols are synonymous, and all future references will be to TLS only. As the majority of Internet traffic today is TCP based, TLS was architected to simply provide protection for TCP-based communications. The creation of higher-layer protocols, such as HTTP, to convey rich data between producers and consumers of these data, resulted in a desire to further protect these data from casual eavesdropping and modification; this resulted in the creation of TLS. When TLS is used in conjunction with HTTP, it is typically denoted as secure HTTP or simply HTTPS. TLS has also been natively embedded into most web servers and clients, further increasing its use and popularity, such that it is the predominant security protocol employed today.

"TLS was architected to simply provide protection for TCP-based communications."

These three security protocols are complementary in nature, and all of them can be employed at different, or at the same, network nodes to offer different security services. For example, Linksec may be employed to control network access, perhaps at a device level (although it can be also pertinent to authenticate the user) in order to ensure that an authorized device is connecting to a given network. IPsec may be employed on top of the Linksec connection to either connect back to a private domain (for example, by using a VPN) or to directly connect back to a server being accessed. This provides access control at the domain or machine (server) level to ensure that only authorized devices and users are connecting to authorized resources. Additionally, TLS may be employed on top of IPsec (and Linksec) to provide application-level guarantees for authorized access control from a given user or machine. As these protocols are at different layers of the network stack, they are complementary in nature and may be employed simultaneously (even if each service is being offered by a different service provider) to ensure that cryptographic guarantees are provided independently for each level of access within the network stack (even though these may physically terminate at different network points). Figure 1 illustrates how these protocols reside in a simplified OSI stack.

How Protocol Security is Implemented and Provisioned

It is our belief that in most production network stacks, protocol security will be provisioned independently for each layer of the OSI stack. For example, link layer (Layer-2) security will be implemented in the network controller (that is, in the Ethernet network adapter). In contrast, network layer (Layer-3) encryption and authenticity will likely be provisioned either in the hardware or in the software stack, depending on the network layer offloads provided by the network device and the end-station platform. When implemented in the network stack, the cryptography can be optimized (offloaded) by using an accelerator capability in the platform, either as an add-in device or as instruction set improvements in the processor(s). Due to the nature of current protocol standards, transport- and application-layer security is difficult to offload to a network adapter, and thus it is and is likely to be implemented in software, perhaps by using platform or device acceleration, such as the new cryptographic instruction set, which is briefly described later, and also described in another article in this issue of the *Intel Technology Journal.*

"Cryptography can be optimized by using an accelerator capability in the platform, either as an add-in device or as instruction set improvements in the processor(s)."

In the remainder of this article, we focus on network-layer security that can provide efficient E2E confidentiality and authenticity protection that is transparent to the applications. The most relevant network layer security protocols are collectively known as *IP Security* or IPsec.

Software Requirements/Stack

Internet protocol (IP) security, or IPsec, is implemented as a substitute network layer for the IP v4 or v6. This alternate layer receives encrypted packets or authenticated packets from the link layer, and it performs the appropriate cryptographic operations on those packets. If the cryptography is successful, the transport layer headers and payload are then passed *up the stack* as validated plain text data. The upper layers of the stack need not be (and are not) aware of the security provided by the IPsec layer.

Because the required cryptographic operations are computationally expensive, deployment of IPsec security in high-bandwidth-use models has been limited. These applications need hardware acceleration to be feasible.

"Because the required cryptographic operations are computationally expensive, deployment of IPsec security in high-bandwidth-use models has been limited."

Hardware Offload and Acceleration

As noted in the introduction to this section, the unloading of the cryptographic overhead from the main CPU is typically accomplished in one of two ways.

First, specialized devices can be used as cryptographic accelerators. In the *accelerator* model, the network device delivers the encrypted and authenticated packet to a software stack that has been enabled to make use of the accelerator. This can provide significant performance enhancements over software-only implementations. A similar acceleration can be realized, by using specialized instructions in the platform processor that accelerate cryptographic algorithms, without the need for specialized add-in devices. In the following section we highlight our efforts in this area.

The cryptographic performance realized from an accelerator architecture may not be complete, because it cannot provide the other services that are typically offered by a network adaptor. Modern network adapters provide several network offloads that cannot be performed on encrypted traffic. On the transmit side, these offloads include TCP segmentation offload (TSO), sometimes known as large send offload or LSO.

On the receive side, the offloads include receive side scaling (RSS) to distribute the cost of network processing across multiple cores or CPUs. Advanced offloads such as receive side coalescing for TCP streams are also disabled by encrypted and authenticated traffic.

"An accelerator architecture does not completely offset the overhead cost of cryptographically secure traffic in the end-station."

An accelerator architecture provides a large benefit in enabling secure, high-bandwidth use cases. Because of the loss of higher-level network offloads, this architecture does not completely offset the overhead cost of cryptographically secure traffic in the end-station.

An alternative to an accelerator architecture is to provide cryptographic offload in the network controller itself. In such an implementation, the network adapter has a cryptographic engine that is coordinated with the software stack to perform the required operations, as the data are received from and transmitted to the network. The received packets are handed to the software stack already validated and decrypted, along with an indication of the success or failure of the cryptographic operation. Likewise, the packets for transmission are passed down to the device *in the clear*, and the device is requested to do the cryptographic operations during transmission. By performing the security operations in the *data plane* of the network device, the valuable higher-level offloads provided by the network adapter can still be performed, and the performance impact of providing E2E security can theoretically be eliminated.

Cryptographic Algorithms and Modes

"Several common algorithms used for IPsec have come and gone over the years."

Whether accelerated or implemented purely in software, IPsec stacks implement a variety of algorithms and cryptographic modes of operation. These are negotiated between communicating hosts during the initial setup of the security association (SA).

Several common algorithms used for IPsec have come and gone over the years. Initially the encryption algorithm of choice was the data encryption standard (DES) as defined in the Federal Information Processing Standards (FIPS). It was later found that the 56-bit key used for DES was not robust enough, and the standard was modified to specify *Triple DES* as a more secure algorithm. On the authentication side, the secure hash algorithm (SHA) was promulgated as a FIPS standard, but was similarly deprecated in favor of a new SHA algorithm (SHA-2 or SHA-256/SHA-512). In 1997, the National Institute of Standards put out a request for candidate replacements for DES (called AES). The winning algorithm was one developed by Joan Daemon and Vincent Rijmen and was called Rijndael. AES is now the standard encryption algorithm in protocol security for many cryptographic protocols, including Linksec, IPsec, and TLS.

AES can be used for IPsec in multiple modes. The most common mode currently employed is called cipher block chaining (CBC) mode. In this mode, there is an initial value (IV) used as input to the encryption of each 16-byte block of data. For the first block, a random IV is used. For subsequent blocks, the IV is the cipher text of the previous block. This technique obscures the presence of long strings of identical values in the plain text.

"AES-GCM has significant performance advantages over other encryption and authentication algorithms."

Alternatively, the AES algorithm (as well as other encryption algorithms) can be used in *counter mode.* As in CBC mode, counter modes use an IV, but the IV is treated as a 128-bit counter that is encrypted and then bitwise XOR'd with the plain text to produce the cipher text. By encrypting a counter (rather than the plain text), counter mode can be implemented in hardware much more efficiently because, in hardware, the input counters to the cryptographic engine can be issued in a pipeline manner. In addition, for AES, counter mode encryption can be combined with a carry-less multiply (called a Galois Field multiply) to provide a message authentication code, along with the cipher text of the packet in a single pass over the plain text. This combined mode of AES is known as AES-GCM and has significant performance advantages over other encryption and authentication algorithms.

The proliferation of algorithms and modes for IPsec has contributed significantly to the perception that IPsec is too complex to deploy in real networks. This complexity is significantly reduced by the fact that the algorithms and modes are negotiated between the parties in the secure connection and need not be manually configured by users.

Protocol Security Initiatives at Intel

Intel is addressing the area of protocol security in several ways. In upcoming microprocessors, Intel has announced that it will provide new instructions that promise greatly accelerated cryptographic operations on the platform. The Intel cryptographic instructions provide significant performance enhancements when compared to implementations that use conventional x86 instructions. The AES instructions [8] accelerate implementations of the AES algorithm. When paired with another new instruction, PCLMULQDQ, the full AES suite of AES-CBC encryption, as well as AES-GCM and GMAC, has dramatically less overhead than leveraging two discrete cryptographic algorithms, one for data confidentiality and another for data authenticity. These instructions are useful for all cryptographic operations on the platform, including network security that uses IPsec.

"Intel has announced that it will provide new instructions that promise greatly accelerated cryptographic operations on the platform."

This AES instruction set includes six Intel SSE instructions. Four instructions (AESENC, AESENCLAST, AESDEC, and AESDELAST) support the high-performance AES encryption and decryption; the other two instructions (AESIMC and AESKEYGENASSIST) facilitate the AES key expansion procedure.

One example scenario is where multiple blocks (16 bytes per block) are encrypted by the same key in AES-128. In this scenario, we store eight data blocks in memory, and a round key is also loaded into a separate memory area. For each round, eight AES round instructions are issued, operating on the eight data blocks with the same round key. Then, the next round key is loaded. The eight blocks encryption results are finally stored into memory. In a nutshell, the code computes one AES round on all eight blocks, using one round key, and then continues to the next round (using the next round key).

"The industry is currently beginning to enable Linksec, and Intel's adapters are ready to process that traffic when it becomes available."

In the NIC offload, the difference is that the dedicated circuit does all the above operations on the key expansion and data-path encryption. The offload engine has a dedicated CAM and SRAM to store thousands of keys for different SAs. Receiving an encrypted packet, the crypto circuit extracts certain identifiers from the packet and uses them to quickly look up the corresponding key. Then the key and the cipher text blocks are fed into the dedicated AES pipeline engine for the decryption operations. The dedicated crypto circuit can flexibly handle all scenarios on the SAs: back-to-back packets from the same SAs (one key encrypting many data blocks); and a mixture of packets from different SAs (many keys encrypting many data blocks).

Intel is also developing network controllers that offload AES-GCM in the data plane. This provides the performance benefits described earlier as well as the authentication and cryptographic operations for MACsec/Linksec (802.1AE/802.1X-Rev). The industry is currently beginning to enable Linksec, and Intel's adapters are ready to process that traffic when it becomes available. IPsec, on the other hand, has been in the industry for many years, but has recently gained additional attention due to its ability to protect network traffic, independent of specific applications.

IPsec Offload

In order to address the problem of network layer cryptography, Intel's networking group has developed two products that provide data-plane offload of IPsec traffic. The first available product, the Intel® 82576EB Gigabit Ethernet controller, code name Kawela, is a 1-Gb Ethernet solution that became available in mid-2008. The second product, the Intel® 82599EB 10 Gigabit Ethernet controller, codename Niantic, is a 10-Gb Ethernet solution that was first available in 2009. There are only minor differences between the feature sets of these two products. In the remainder of this article we focus on the Intel 82599EB 10 Gigabit Ethernet controller.

"The Intel 82599EB 10 Gigabit Ethernet controller is designed to work with the Microsoft IPsec Task Offload interface."*

The Intel 82599EB 10 Gigabit Ethernet controller is designed to work with the Microsoft* IPsec Task Offload interface to offload the cryptographic operations for specific IPsec streams. All of the control plane operations remain with the operating system (OS).

Both controllers only offload the AES-GCM encryption and authentication algorithm and modes. The GCM mode of AES is optimal for implementation in silicon. Other modes of AES (that is, CBC mode) and other algorithms (3DES) are expensive to implement in hardware for high-speed network devices. As the algorithm and modes of IPsec are negotiated at session startup, the optimal algorithms for the communicating hosts can be selected.

When the driver loads for the Intel 82599EB 10 Gigabit Ethernet controller, the OS queries the IPsec offload characteristics to determine which algorithms are supported and how many flows may be offloaded. After an IPsec connection has been established and the keys negotiated, the OS may choose to offload the SA to the controller. Two sets of keys are passed down from the OS to the driver, one for outbound traffic and one for inbound traffic. The driver is responsible for transferring each set of keys to the controller in such a way that they may be efficiently accessed as needed for future data operations.

Once the driver has completed offloading the cryptographic information to hardware, the driver is available to process requests associated with the SA. For outbound traffic, the OS knows which SA needs to be used for a request, so it passes down a handle with each packet. The driver needs to translate that handle into a hardware command, but the authentication and encryption are left up to the hardware. For inbound traffic, hardware needs to do more work to figure out which SA needs to be used for a given packet. The packet parser first extracts several fields from the packet and then uses these fields to find the SA that should be used. Once a match is found, the hardware authenticates and decrypts the packet data before passing the packet to the OS for further processing. Hardware provides a status indication so that the OS can determine whether the authentication check passed or failed.

Conceptually, the Intel 82599EB 10 Gigabit Ethernet controller implements protocol security in a block right next to the physical interface of the device, as depicted in the block diagram in Figure 2.

Because security is implemented in this way, cryptographic operations are the last to be performed on outgoing traffic and the first to be performed on incoming traffic. This allows the network device to apply any of the offloads or routing algorithms that it usually applies to non-IPsec traffic. For example, after a received packet is decrypted and authenticated, the device can calculate the TCP checksum and pass the result of this test up to the OS. As another example, the Intel 82599EB 10 Gigabit Ethernet controller supports the combination of TCP segmentation offload and IPsec offload. The OS can pass down as a single request enough data for several packets on the wire. The controller will use the prototype header provided by the OS to first segment the request into packets, insert the appropriate checksums into each packet, and finally encrypt and authenticate the packet before it is sent out on the wire.

The acceleration and offloads just described are designed to overcome the performance impact of utilizing network-layer security in enterprise networks. It is useful to take a quantitative look at the performance benefit of these technologies.

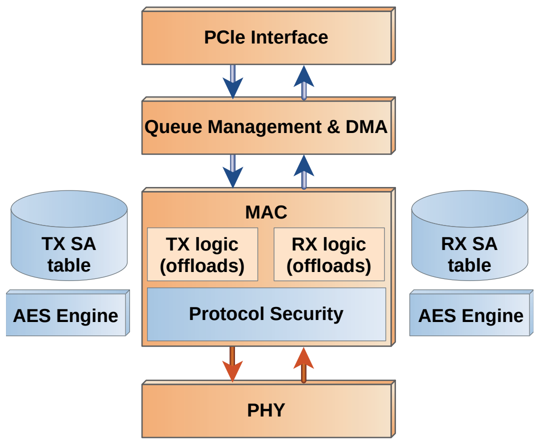


Figure 2: Network Controller Security Block Diagram
Source: Intel Corporation, 2009

Offload Performance

When offloading IPsec to the network device, the benefit is significant.

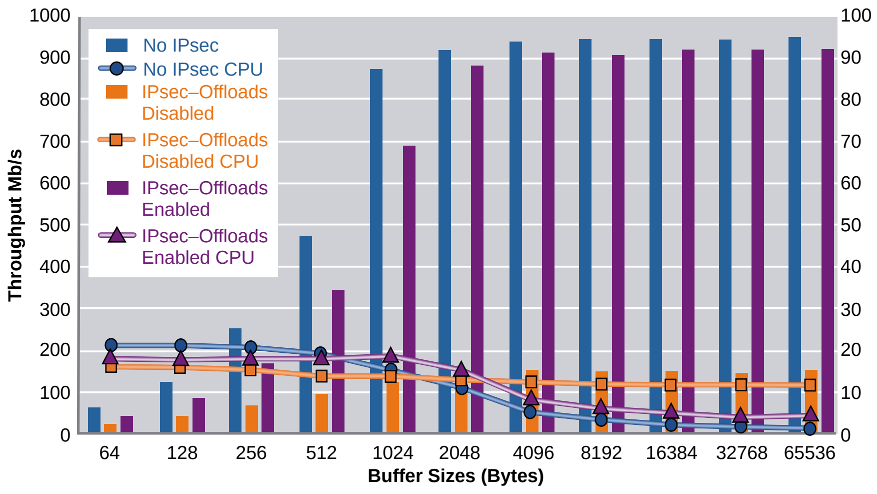


Figure 3: Network Controller Security Offload Performance
Source: Intel Corporation, 2009

The left axis and bars of the graph in Figure 3 are the maximum throughput of the network device in three circumstances. The blue bars represent the offload performance with no security enabled, at a range of I/O sizes. The orange bars represent that performance when security is enabled, but IPsec offload (or acceleration) is not available. Finally, the purple bars represent the throughput when security is enabled and the IPsec offloads are enabled in the network controller. The right axis and the three line graphs show the corresponding percentage of CPU that is consumed in each circumstance and for each I/O size. These measurements are for receive traffic only and were taken on a dual-socket server with Intel® Xeon® 5355 processors, codename Clovertown, with eight functional CPU cores running a Microsoft* Windows* 2008 server.

"These performance tests show that even today, the performance impact of IPsec protection for network flows can be mitigated."

As can be seen from the bar chart in Figure 3, the performance of an offloaded IPsec connection approaches that of an unprotected network flow. The slight decrease in throughput and increase in CPU is explained by the extra validation (padding content verification and anti-replay checking) required in the software stack for IPsec, which is not offloaded. The extremely low throughput and the constant CPU utilization (light orange line) in the non-offload case is explained by the fact that the network processing for the single stream is being done on a single core and the cryptographic operations are saturating that core.

Details on the performance characteristics of the new cryptographic instructions are provided in a separate article [9] in this issue of the *Intel Technology Journal*.

These performance tests show that even today, the performance impact of IPsec protection for network flows can be mitigated. This removes the performance barrier to widespread use of IPsec in the enterprise.

Industry Collaboration

Aside from designing and delivering products to accelerate secure networking, Intel is also working with industry players to make protocol security a more widely deployed technology. At the link layer, the same protocol must be implemented by both end nodes and by each switch or router along a given path. Thus, interoperability is critical.

One key barrier to IPsec deployment in the enterprise is the performance impact of providing cryptographic services to secure traffic. Intel is addressing this directly with its processor instructions and offload network devices. Other key barriers include the cost of designing a network device that supports tens of thousands of SAs, the difficulty in managing a network in which traffic is encrypted and therefore not visible to intermediary devices, and the fact that it is very computationally expensive to create a secure connection. These barriers cannot be cleared by any one company. Cooperation will be needed among many industry players including OS vendors, intermediary device vendors, and network device vendors in order to create a solution that will have the characteristics needed to be widely adopted.

Future Research

Network security touches the everyday lives of people. For instance, an ordinary person knows there is an encryption channel between his or her computing device and server when an on-line E-commerce or banking transaction occurs. From a layperson's vantage point, therefore, network security is thought of as the technology of one endpoint encrypting and authenticating a data packet and the other endpoint decrypting and validating that packet. As networking infrastructure devices and computers continue to evolve, it becomes clearer that this layperson's view overlooks some important subtleties. In this section, we point out a few requirements that may demand additional technology breakthroughs to tackle the deployment of real-world applications.

Traffic Visibility

Internet-based malware (viruses and worms) is getting more and more sophisticated as hackers become more sophisticated and reap increasing financial benefits from their exploits. More and more government agencies and commercial organizations also use cyber attacks as a means to gain information over their competitors. Because of this trend, we have observed a heavy investment in network appliances (such as intrusion detection systems, intrusion prevention systems, and so forth) in enterprises. Furthermore, enterprise environments employ numerous other network management tools that observe, monitor, and modify network packets to provide various network services. The network appliances usually perform deep-packet inspection on clear text traffic; however, they are not able to function properly when dealing with encrypted packets. Hence, enterprises start to face a choice between employing existing network-monitoring tools operating on clear text traffic, or succumb to the reality of securing the traffic within the network. On initial evaluation, these two goals appear to be mutually exclusive. The technical challenge is how to satisfy these two opposite goals of E2E security and traffic visibility.

One naïve solution might be to request an endpoint to relinquish the security keys (at least the data encryption keys) for each session to each intermediary device needing access to the data. However, the SAs are dynamic in terms of both spatial and temporal properties. The naïve solution might lead to the wrong key being provided for a given encrypted packet to the intermediate device. Furthermore, as well as the associated overhead in synchronizing the large number of keys with each intermediate device on a per-secure-session basis, these security sessions come and go on demand, further compounding the problem.

Scalability

Another significant development in the networking arena is cloud computing. In cloud computing, computing resources are created and maintained at powerful data centers and accessed by clients. The network serves as a vehicle to deliver the input and output between the data centers and clients. In order to protect the increasing amount of data over the network, an SA is established between a client and a server. The implications are hundreds of thousands, even millions, of SAs terminated at a server through aggregated 10-Gbps pipes. The aforementioned CPU-based and NIC offload solutions can certainly be used to tackle this problem. However, the essential question is whether or not those technologies can keep up with the scalability requirement, as load and demand for these services increase. The case of hundreds of thousands of SAs will incur megabytes of storage for keying materials that will likely contend with the CPU cache, with the packet processing (for example, TCP/IP), and with application-level logic processing. Furthermore, contention is created for data movement between the cache and accelerator at the CPU SSE domain. For NIC offload, this sheer volume of SAs will demand megabytes of SRAM and CAM, a very expensive solution. We note that the leading industry companies could provide more than 200-Gbps throughput of AES circuit at the 22-nm silicon process. However, the problems of large-volume SAs may not be easily overcome by such crypto-primitive circuits. In this case, protocol-level security is a must to address the scalability issues, such as the key context change between multiple SAs and the packet processing performance requirements.

Intel is actively working with other companies to define protocol extensions in the standards community to provide these services. One such activity is the definition of an extensible IPsec ESP packet format through the Internet Engineering Task Force (IETF) community. Within the IETF, we are introducing the concept of *Wrapped ESP*, or WESP for short, which extends the existing ESP packet format to provide additional benefits to trusted intermediary devices for packet inspection. We outline some of the problems addressed by WESP.

"In cloud computing, computing resources are created and maintained at powerful data centers and accessed by clients."

"The essential question is whether or not those technologies can keep up with the scalability requirement, as load and demand for these services increase."

*"We are introducing the concept of **Wrapped ESP**, or WESP for short, which extends the existing ESP packet format to provide additional benefits to trusted intermediary devices for packet inspection."*

WESP addresses the problem of the ability to discern between encrypted IPsec traffic and traffic carrying authentication-only tags within the IPsec ESP protocol. In the current ESP specification, there is no way to differentiate between these types of traffic, without knowing every policy associated with every single IPsec SA being monitored by the trusted intermediary device. WESP proposes to address this by introducing a new wrapper to the existing ESP protocol and leveraging a brand new protocol number to identify the WESP protocol. Devices supporting the new WESP protocol header will be able to easily identify and differentiate between encrypted data and ESP-NULL (integrity protected) data and make rapid decisions on whether the packet can be examined or if it is encrypted.

WESP also proposes some extension headers that allow the trusted intermediary devices to determine the ESP overhead in the packet header and trailer by providing these overhead field sizes directly in the WESP header. The WESP header also proposes to add a field directly in the header that readily identifies the upper-layer protocol (TCP/UDP) being carried by the IPsec packet. In the current ESP specification, the upper-layer protocol is stored at the tail end of the packet, something that requires any device to store and parse the whole packet, before determining what the upper-layer protocol within the packet is. This requirement is not amenable to building high-speed, highly-optimized, cost-effective, hardware-based solutions. These changes will allow the intermediary devices to easily identify and extract the packet payload for further analysis, instead of having to store or identify and use a specific rule for each secure IPsec connection in order to determine these overheads (as different connections may employ different security algorithms and policies), before extracting the upper-layer payload.

"Intel is working with other vendors to make IPsec ubiquitous and applicable to every Layer-3 and above data packet."

Even though IPsec has been prevalent in the industry for over a decade, it has gained little traction for applicability to E2E data protection; instead, its most popular usage has been in the areas of remote access and VPNs. Intel is working with other vendors within the industry to change this and make IPsec ubiquitous and applicable to every Layer-3 and above data packet. This will provide E2E data security, while allowing trusted intermediary devices to operate unhindered within this environment,

Summary and Conclusion

Network protocol security is increasingly important in the face of attacks that render the prevalent paradigm of perimeter security less effective. Furthermore, protocol security deals with attacks that cannot be fended off with perimeter security. As usages, such as remote storage and others, become more and more prevalent, it becomes increasingly necessary to protect the data *in flight* between platforms. Moreover, *in-flight* security as a utility is increasingly necessary for all applications.

"Protocol security deals with attacks that cannot be fended off with perimeter security."

Protocol security places new demands on the overall compute and network environment. New security solutions must be economical and scalable or they will not be widely adopted. Network security at different layers of the network stack (2, 3, and 5+) offers different services for network access control for authorized devices and users. These services are complementary in nature and may be employed simultaneously for a given connection. Without them, though, the Internet and our more local networks will become increasingly insecure and less trustworthy.

We and our colleagues at Intel are actively researching protocol security solutions, introducing new products based on our research, and working with other companies to develop a secure network ecosystem that is unobtrusive, reliable, scalable, efficient, and trustworthy. We will continue this work with new technologies on the horizon that will make secure network communications even better going forward.

References

[1] Kent, S. "IP Encapsulating Security Payload (ESP)." *RFC* 4303, December 2005.

[2] Kent, S. "IP Authentication Header." *RFC* 4302, December 2005.

[3] IEEE 802.1AE. "Media Access Control Security (MACSec)." June 2006. At *www.ieee802.org*

[4] IEEE 802.1X-Rev. "Draft standard Port-based Network Access Control." Expected end of 2009. At *www.ieee802.org*

[5] Kaufman, C. "Internet Key Exchange (IKEv2) Protocol." *RFC* 4306, December 2005.

[6] Kivinen, T., Swander, B., Huttunen, A., and V. Volve. "Negotiation of NAT-Traversal in the IKE." RFC 3947, January 2005.

[7] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg. "UDP Encapsulation of IPsec ESP Packets." *RFC* 3948, January 2005.

[8] Gueron, Shay. "Advanced Encryption Standard AES Instructions Set." April 2008. At *www.intel.com*

[9] Gueron, S., Kounavis, M. "New Processor Instructions for Accelerating Encryption and Authentication Algorithms." *Intel Technology Journal*, Volume 13, Issue 2, 2009.

Author Biographies

Linden Cornett is a Software Architect for Intel's LAN Access Division. She has worked for Intel for over ten years and holds several patents in the areas of networking and communications. Linden graduated from Williams College with a Bachelor's degree in Computer Science and Mathematics in 1998, and she received her MBA from Babson College in 2006. Her e-mail is linden.cornett at intel.com.

Ken Grewal is a Research Scientist at Intel Labs. Ken has worked for Intel for over ten years and holds several patents in the areas of network security and communications. Ken graduated from City University, UK with a Bachelor's degree in Applied Physics in 1988 and has extensive experience in the software and network security arena over the last 20 years. Ken holds two patents and has over 20 pending in the area of network and computer security. His e-mail is ken.grewal at intel.com.

Men Long is a Research Scientist at Intel Labs. Men received a B.E. degree (Honors) from Chongqing University, Chongqing, China, in 2000 and a Ph.D. degree from Auburn University, Auburn, AL, USA, in 2005, both in Electrical Engineering. Men joined Intel Corporation in 2005. He has 18 pending patents and has published 23 peer-reviewed research papers (for IEEE and ACM transactions and conferences, among others) in the areas of network and computer security, applied cryptography, wireless networking, and image processing. His e-mail is men.long at intel.com.

Marc Millier is a Software Architect and Architecture Manager in DEG Architecture and Planning. Marc is leading the LAN Access Division's protocol security architecture effort. He has worked for Intel for 15 of the last 30 years or more of a software engineering career. His e-mail is marc.millier at intel.com.

Steve Williams is a Product Planner for Intel's LAN Access Division. He has worked in various aspects of the networking industry for over 20 years, with the last 15 years being spent with Intel. He holds four patents. Steven graduated from the University of California at Santa Barbara in 1981 with a Bachelor's Degree in Computer Science. His e-mail is steven.d.williams at intel.com.

Copyright

THE DARK CLOUD: UNDERSTANDING AND DEFENDING AGAINST BOTNETS AND STEALTHY MALWARE

Contributors

Jaideep Chandrashekar
Intel Corporation

Steve Orrin
Intel Corporation

Carl Livadas
Intel Corporation

Eve M. Schooler
Intel Corporation

Index Words

Malware
Botnets
Network Defense
Cyber-Security
Stealth
Rootkits

"Botnets are the latest scourge to hit the Internet and are the latest challenge for IT personnel."

Abstract

The proliferation of botnets reveals a worrisome trend in the spread and sophistication of computer viruses and worms in the Internet today. (A *botnet* is essentially a collection of compromised distributed computers or systems, known as *bots* because of their zombie-like nature, under the control of a *bot-herder*, by virtue of the use of command and control servers.) Botnets are the latest scourge to hit the Internet, each one revealing a new level of technologic expertise and the use of quality software processes that undermine, if not downright prohibit, the ability of current anti-malware and other intrusion detection systems (IDSs) to deal with them. Most IDSs focus on detecting known threats, or on detecting the volume of traffic generated by a bot-host after it has been activated. Most bots, however, are polymorphic: they change with every instantiation so appear as something new every time. Furthermore, most bots generate only low-volume, periodic communication back to a bot-herder, and this volume is generally within the thresholds used by IDSs. In this article, we present an overview of the state of the art of botnets and stealthy malware, then develop and present several promising anti-botnet defense strategies that specifically target current and emerging trends in botnet development.

Introduction: Current and Emerging Trends in Botnets

With estimates of botnet infections continuing to gain in momentum, botnets are the latest scourge to hit the Internet and are the latest challenge for IT personnel. Each new botnet discovered reveals the use of more advanced technology and the use of quality software processes that are challenging the defense strategies of current intrusion detection systems (IDS). Thus, we begin this article with an overview of the state of the art of botnets and stealthy malware. We first describe the botnet lifecycle and highlight the advanced capabilities and stealth techniques in use today by botnets; we also examine and strategize about future advances in this area. We then go on to present several promising anti-botnet defense strategies, notably a collection of real traces to calibrate normalcy, the development of techniques that analyze communication with remote nodes with the goal of identifying botnet command-and-control (C&C) channels, and the application of various forms of correlation to amplify accuracy of detection and to root out stealthiness.

Botnets Defined

A botnet is a collection of distributed computers or systems that has been compromised, that is, taken over by rogue software. As a result, these machines are often called *zombies* or *bots*. Bots are controlled or directed by a *bot-herder* by means of one or more C&C servers. Most commonly, the bot-herder controls the botnet with C&C servers, delivered via protocols such as internet relay chat (IRC) or peer-to-peer (P2P) networking communications. Bots typically become installed on our devices via malware, worms, trojan horses, or other back-door channels. Further information on botnets can be found in [1].

"A botnet is a collection of distributed computers or systems that has been compromised."

The statistics for the size and growth of botnets differ widely, based on the reporting organization. According to Symantec's "Threat Horizon Report" [2], 55,000 new botnet nodes are detected every day, while a 2008 Report from *USA Today* states that "…on an average day, 40 per cent of the 800 million computers connected to the Internet are bots used to send out spam, viruses and to mine for sensitive personal data" [3]. *USA Today* also reports a tenfold increase in 2008 in the code threats reported over the same period in 2007, signifying the increase in threat surface area for botnet-style infections [3]. Various sources estimate that the best-known botnets—Storm, Kraken, and Conficker—have infected staggering numbers of machines. These numbers range from 85,000 machines infected by Storm, to 495,000 infected by Kraken [4], to a staggering 9 million nodes infected by Conficker [5].

The Underground Economy and Advances in Botnet Development

Like any money-driven market, botnet developers operate like a legitimate business: they take advantage of the economic benefits of cooperation, trade, and development processes, and quality. Recently, botnets have begun to use common software quality practices such as lifecycle management tools, peer reviews, object orientation, and modularity. Botnet developers are selling their software and infection vectors, providing documentation and support, as well as collecting feedback and requirements from customers.

"Bots typically become installed on our devices via malware, worms, trojan horses, or other back-door channels."

Common economic goals are driving innovation, collaboration, and risk reduction in the Botnet communities. On-line barter and marketplace sites have sprung up to service this underground community with barter and trade forums, on-line support, and rent and lease options for bot-herders. This cooperation has led to a fairly mature economy where botnet nodes or groups are bought and sold, or where several bot-herders can cooperate when targeting an entity for attack. Botnets can be rented for the distribution of spam. Stolen identities and accounts are traded and sold among the participants.

"Botnet developers are selling their software and infection vectors, providing documentation and support, as well as collecting feedback and requirements from customers."

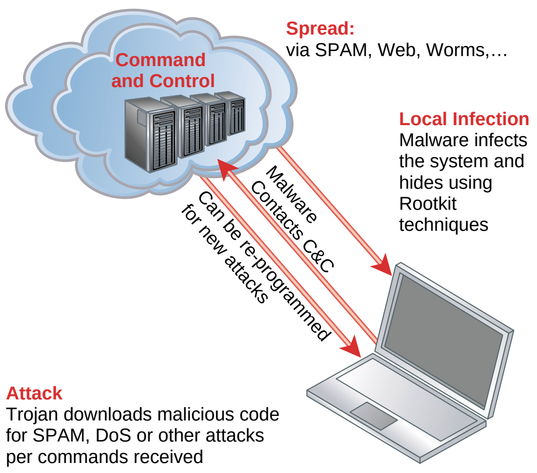

Spread:
via SPAM, Web, Worms,…
Command and Control
Local Infection
Malware infects the system and hides using Rootkit techniques
Malware Contacts C&C
Can be re-programmed for new attacks
Attack
Trojan downloads malicious code for SPAM, DoS or other attacks per commands received

Figure 1: The Botnet Lifecycle
Source: Intel Corporation, 2009

The Botnet Lifecycle

The lifecycle of a botnet typically includes four phases: spread, infection, command and control (C&C), and attack, as shown in Figure 1. We describe each phase.

Spread Phase

In the spread phase in many botnets, the bots propagate and infect systems. Bots can spread through a variety of means, including SPAM e-mails, web worms, and through web downloads of malware that occur unbeknownst to users. Since the goal of the spread phase is to infect a system for the first time, bot-herders attempt to either trick the user into installing the malware payload or exploit vulnerabilities on the user system via applications or browsers, thereby delivering the malware payload.

Infection Phase

The malware payload, once on the system, uses a variety of techniques to infect the machine and obfuscate its presence. Advances in bot infection capabilities include techniques for hiding the infection and for extending the life of the infection by targeting the anti-malware tools and services that would normally detect and remove the infection. Botnets employ many of the standard malware techniques in use by viruses today. Polymorphism and rootkitting are two of the most common techniques in use.

"By polymorphism, we mean that the malware code changes with every new infection."

- By polymorphism, we mean that the malware code changes with every new infection, thus making it harder for anti-virus products to detect the code. Further, the use of code-hardening techniques often employed by SW developers to protect from SW piracy and reverse engineering, are in turn used by botnet developers. These techniques include code obfuscation, encryption, and encoding that further hide the true nature of the malware code as well as making it harder for anti-virus vendors to analyze it. There are indications that malware and botnet developers are beginning to look into advanced rootkitting techniques to further hide the malware.
- By rootkitting, we mean the stealthy installation of malicious software–called a rootkit–that is activated each time a system boots up. Rootkits are difficult to detect because they are activated before the system's operating system (OS) has completely booted up. Advances in rootkit techniques include hyperjacking and virtualization-based rootkits as well as identifying and using new targets for code insertion such as firmware and BIOS.

"By rootkitting, we mean the stealthy installation of malicious software–called a rootkit–that is activated each time a system boots up."

A virtual machine monitor (VMM) or hypervisor runs underneath an OS, making it a particularly useful means for botnet and malware developers to gain control of computer systems. Hyperjacking involves installing a rogue hypervisor that can take complete control of a system. Regular security measures are ineffective against this hypervisor, because the OS is unaware that the machine has been compromised, and software anti-virus and local firewalls are unable to detect them.

Another technique that is currently used by botnet developers is to actively target the anti-virus, local firewall and intrusion prevention and detection software (IPS/IDS) and services. Some of the techniques employed by botnets have included attacking the anti-virus and firewall software by killing its process or blocking its ability to get updates. Two examples that we know of show how botnets blocked the security software from getting updates:

- A botnet changed the local DNS settings of the infected system to disable the anti-virus software from reaching its update site.
- A botnet was actively detecting connection attempts to the update site and blocking them.

These update-blocking techniques prevent the security software from getting potential updated signatures from the vendor that identify the newer version of the botnet or from being able to communicate with a central vendor server for anomaly correlation and update.

Timing the infection to strike between malware detection services scan times is another infection technique employed by botnet developers. The bot slowly infects a system without generating alarms in the intrusion detection software services.

"Timing the infection to strike between malware detection services scan times is another infection technique employed by botnet developers."

Other advanced bots spoof the local and remote scans performed by the IDS/IPS and anti-virus software. In this case, the botnet's malware presents a false image of memory or hard disk to the anti-virus software to scan, or the malware disrupts vulnerability scans by dropping packets, spoofing the network response, or redirecting traffic coming from vulnerability scanners.

Command and Control

Botnet C&C servers use one of several protocols to communicate, the most common of which up to this point has been IRC. Recently, however, a trend towards the use of protected or hardened protocols has begun to emerge. For example, the Storm botnet uses an encrypted P2P protocol (eDonkey/Overnet). Advances in C&C techniques are crucial for bot-herders to keep their Botnets from being detected and shut down. To this end, botnets have begun to leverage protocols such as HTTP and P2P that are common across networks, thus making the botnet harder to detect. HTTP is particularly advantageous to botnets because of the sheer volume and diversity of HTTP traffic coming from systems today. Also, botnet software can take advantage of the local browser software for much of its functionality and communications stack, leveraging HTTP's ability to transit firewalls. Other techniques on the horizon include the use of VoIP, web services, and the use of scripting within the HTTP communications stack. Another advanced technique uses a *blind drop*, a site on the Internet such as a forum, BBS, or a newsgroup, where users can leave anonymous messages. Botnet nodes can post messages to these sites, and bot-herders can anonymously check for messages from their nodes and post instructions. The botnet nodes can then poll the site for new instructions and other communications as part of a messaging-based C&C. Social networking sites are a prime target for this kind of C&C.

"Botnet software can take advantage of the local browser software for much of its functionality and communications stack, leveraging HTTP's ability to transit firewalls."

A key feature of modern botnet development is the ability to re-program or update the botnet node software after it has infected a system. The C&C directs the node either to download the update directly or to go to a specific infected site hosting the update. Botnets with this reprogrammability have a higher value in the underground economy, as they can be augmented to perform new and advanced attack and stealth missions as they are developed.

"Steganographic techniques are the next method by which botnet developers plan to evade detection."

As mentioned previously, stealth is a key feature of botnet technology. Kracken and Conficker Botnets both target and disable anti-virus software resident on the system. Other botnets deliberately try to hide from threshold-detection software by customizing the timing of infections and the frequency of communications to hide activities from both local and network security products. Steganographic techniques are the next method by which botnet developers plan to evade detection. They include the use of covert channels for communications and steganography-based messaging, such as mimicry and stegged content (i.e., embedding messages in content such as images, streaming media, VoIP, and so on).

Attack Phase

The final phase of the botnet lifecycle is the attack phase. In many cases the attack is simply the distribution of the SPAM that is carrying the infection, and when the attack is successful, the size of the botnet itself increases. Botnets also often have been used to send SPAM as part of barter and rental deals, whereby phishers, hackers, spammers, and virus writers use the botnet to sell information and services. Botnets also have been used to perform massive distributed denial-of-service (DoS) attacks against a variety of targets including government, corporate systems, and even other botnets. Some of the newer botnets can be upgraded to use various hacker tools, fault injectors (fuzzers), and so on, to further attack the networks they have infiltrated. For example, the Asprox botnet included an SQL injection attack tool, and another botnet included a Brute Force SSH attack engine. In addition to performing remote attacks, botnets can engage in persistent local attacks to phish for identities and accounts from the infected system and its users.

"Phishers, hackers, spammers, and virus writers use the botnet to sell information and services."

The Evolution of Anti-Botnet Strategies

Given the proliferation and sophistication of malware, it is not hard to see why traditional anti-malware techniques don't work against botnets. Most IDS focus on detecting known threats, or on detecting the volume of traffic generated by a bot host, after it has been activated. However, most bots are polymorphic: they change with every instantiation so always appear as new. Furthermore, most botnets generate only low-volume periodic communication back to a bot master, and this volume is generally within the thresholds used by IDS.

"Most bots are polymorphic: they change with every instantiation so always appear as new."

In the remainder of this article, we describe the Canary detector that targets early botnet detection. The Canary detector encompasses three promising anti-botnet strategies. The first strategy employed is the analysis of real enterprise network traces that reveal how the network is actually used; this analysis, in turn, reveals how certain user-driven traffic properties differ from botnet traffic. Our second strategy is an end-host detection algorithm that is able to root out the botnet C&C channel. Our approach is based on the computation of a single persistence value, a measure of how regularly remote destinations are contacted. The strength of this method is that it requires no *a priori* knowledge of the botnets that are to be detected, nor does it require inspection of traffic payloads. Although the botnet detection capability may be carried out solely at an individual end-host, we show that detection is further improved by correlating across a population of systems, either at a network operation center (NOC) or in a completely de-centralized fashion, to identify the *commonality* in persistent destinations across multiple systems. This is our third strategy.

"The Canary detector encompasses three promising anti-botnet strategies."

The Design of the Canary Detector

The Canary detector takes a novel approach to detecting stealthy, end-host malware, such as botnets. Here we use the term *stealthy* to mean not generating a noticeable level of traffic. The central idea in our detection scheme is to track the usage of *destination atoms*, the logical collections of destination addresses that describe services. Specifically, we measure the correlation of destination atoms—temporally for individual users, and spatially across sets of users–and scrutinize those destination atoms that become significant. In the case of botnets, for example, the recruited end-hosts typically call home periodically. By tracking this destination atom over time at a coarse level, we can flag it when it becomes significantly persistent.

*"Here we use the term **stealthy** to mean not generating a noticeable level of traffic."*

Preliminaries

Destination Atoms in Intel Enterprise Traces

Interested in studying correlations between user activity and network traffic patterns, we launched an enterprise data collection effort from inside Intel's corporate network. We collected traces (over a 5-week period from approximately 400 end-hosts) that we and others subsequently data-mined for interesting phenomena, statistics, and contradictions of long-held assumptions [6].

Looking at real enterprise traces, we can see that there are substantial efficiencies to be gained when correlating destination usage. Thus, our Canary algorithms rely on a level of abstraction we call *destination atoms*, that is, logical representations of network services. This level of summarization leads to a significant reduction in the number of destination *entities* that are tracked, and thus, tracking atoms requires less overhead. The base definition for a destination corresponding to a connection is the tuple (destIP, destPort, proto), which is simply the end-point for the connection consisting of the destination address, the destination port, and the transport protocol that is used. Often, in the case of well-known services, multiple physical hosts provide the same, indistinguishable application service. Thus, we can group the set into a single atom (dstService, dstPort, proto). Here, the *service* is simply the domain name to which the underlying addresses resolve. Examples of atoms include (www.google.com, 80, tcp), (akamaitech.com, 80, tcp), and (mail.cisco.com, 135, tcp).

*"Our Canary algorithms rely on a level of abstraction we call **destination atoms**, that is, logical representations of network services."*

Further summarization is also possible by applying heuristics on how ports are used by applications. Consider an FTP server, connected in PASV mode. The initial connection is over port 21, but a separate server-negotiated ephemeral port is used for data transfer. Thus, a single FTP session has two atoms, (ftp.service.com, 21, tcp) and (ftp.service.com, k, tcp), where k is a port number beyond 1024, which can be viewed as offering the same service. By considering FTP semantics, we can add the entire range of ports larger than 1024 to the associated atom (ftp.service.com, 21:>1024, tcp). This means that, when we see a connection on port 21, we can expect an ephemeral port to be used in the near future.

In the real enterprise traces, we had many occasions to perform this level of summarization, most notably on the Microsoft* RPC ports between 135 and 139. We then arrive at the full definition of destination atom, the triple (addr set, port set, proto). Here, addr set is a set of destination addresses: these addresses are identical with respect to the applications provided; port set is a set of individual ports or port ranges; and finally, proto is the transport protocol the service uses. Table 1 enumerates some atoms extracted from the enterprise traces.

Destination Atom	**Description**
(google.com, 80, tcp)	HTTP sessions to any of the Google servers
(ftp.nai.com, 21:>1024, tcp)	Updates for Norton antivirus delivered via PASV FTP from the Norton Web site
(mail.cisco.com,135:>1024,tcp)	Microsoft RPC-based services use ephemeral ports after the session is negotiated over port 135

Table 1: Atoms Extracted from Enterprise Traces
Source: Intel Corporation, 2009

Note that a single destination host can provide a number of distinct services, and in this case, the port is sufficient to disambiguate the services from each other, even though they may have similar service names, which are obtained by (reverse) DNS lookup. Finally, note that in cases where the addresses cannot be mapped to names, no summarization is possible, and the conventional destination address is the final descriptor.

Persistence

The key anti-botnet technique we propose is to identify *temporal heavy hitters* without regard to their level of traffic; that is, identify services that get used with a degree of *regularity*. Again, this strategy was validated by the analysis of real enterprise traces from a diverse group of end users in varied geographic regions with disparate usage patterns. We believe that the set of significant atoms for an end-host is small and stable, and that when a host is infected with malware, it will connect periodically to a home server, and the latter will stand out. To perform this detection, we must first assign a numeric value to the somewhat nebulous concept of *regularity*, which we refer to as the persistence of an atom. We want to track the regularity of *usage*, rather than the connections themselves. Consider the act of using your newsreader to download the news headlines. Each time the newsreader application is launched, it makes a large number of connections. To track the long(er)-term communication with the end-host, we concentrate on tracking high-level *sessions*, rather than individual connection frequencies.

"We believe that when a host is infected with malware, it will connect periodically to a home server, and the latter will stand out."

To track high-level sessions, we bin connections to the atom by using a small *tracking window*, *w*, and we assign a 1 or a 0 to that window (the atom was seen 1 or more times, or not). Clearly, the tracking window length should cover *sessions*. When we plot the inter-arrival time for individual atoms across a large number of users, we see that 59 percent of the connections to atoms are made within a minute of each other, and 87 percent of connections to the same atom are separated by at least an hour. We therefore select an hour as the tracking window length to compute persistence.

The other step needed to assign a numeric value to persistence is the construction of an observation window, W ; that is, we look at how long an atom should be regularly observed before it is classified as significant. Based on experience with the data, we defined the observation window, W = 10w, which roughly covers the average work day. Having defined w and $W = (w_1, w_2, \ldots, w_{10})$, we quantify persistence for an atom a, as observed at host h, over the observation window W, p (a, h, W), as the number of individual windows $w_1, w_2, \ldots, w_n$ where the atom was observed.

If we denote p* as a threshold for an atom to be significantly regular, then if $p(a, h, W) > p^*$, the destination a is considered persistent for host h. Note that the definition of persistence has an inherent timescale dictated by W. Suppose that w = 1hour and W = 1day. When computed at this scale, persistence captures the day-to-day behavior of the atom. However, it fails to capture longer-term trends that may exist. Consider two different atoms: a_1, seen every hour, and a_2, observed once a day. We have $p(a_1) = 24/24$ and $p(a_2) = 1/24$.

Intuitively, however, they are both quite *regular* and thus both should be termed persistent. In fact, because we are trying to detect stealthy malware about which we have no *a priori* timescale information, the one timescale we pick may be the exact one that misses the malware activity. Thus, instead of relying on a single timescale W, we consider five different timescales, $W_1, W_2, \ldots, W_5$. Therefore, for every atom, we compute $p(a, h, W_i)$ for $i = 1, 2, \ldots, 5$ and say that it is persistent if $\max_i p(a, h, W_i) > p^*$

Commonality

"*We use commonality to quantify how correlated a destination atom is across the users in a network.*"

While persistence is defined as a property of the individual end-user, we use commonality to quantify how correlated a destination atom is across the users in a network. Thus, a destination atom is significant in this dimension if a large fraction of the users are communicating with it. Since these atoms are created because of many users in a network, we expect them to be quite stable among the population. The commonality metric is defined quite simply: let N (a) be the number of users in the population that see the atom a, at least in some observation window. Thus, the commonality of atom a, $c(a) = N(a)/N$, where N is the total number of hosts in the network. Additionally, we could require a minimum persistence for the atom across the set of hosts that report connections to it; doing so would counter the effect of temporary transients such as flash crowds.

Unlike persistence, this commonality metric cannot be computed in isolation at an individual end-host. Persistence requires a means for the system to collect and correlate information across end-hosts. One solution is to assume the existence of a central IT operations center (ITOC) that can collect periodic reports of atoms observed from all the end-hosts, and that can determine the significant *common* atoms in the set. Alternatively, peer systems can share persistence information periodically with like-minded subsets of the population (e.g., proximate peers, those running a similar OS or patch level, those deemed trusted via the social network of users at the application layer, and so on).

"*Persistence requires a means for the system to collect and correlate information across end-hosts.*"

In contrast to the ITOC approach, significantly common atoms are determined and maintained at the end-hosts, as in [7]. In either scheme an important point is that a sliding window is maintained over the entire observation window (the largest among the different timescales). While computing the commonality metric, only reports within this observation window are considered. Again, the test for significance is when the value of c (a) is greater than a specific threshold $c*$. When $c(a) > c*$, we say that 'a' is common in the population.

Building Whitelists

We construct a whitelist for each user in two steps. First, the host observes its traffic for a training period, builds the set of atoms, and tracks their persistence; the length of this training period would vary with how stable the traffic patterns are, and we expect this to be defined by the network operator. We define p* to be the persistence threshold; that is, if the persistence of a particular atom is larger than p* , then the atom is added to the whitelist. In the detection phase, each end-host sends its set of observed atoms (all of them, not just the persistent ones) either to the central ITOC of the enterprise or to a subset of like-minded peers. At the ITOC, the commonality is calculated for each atom in the union. We define a threshold for commonality, c* , and collect those atoms whose commonality exceeds c* . These atoms are sent to every end-host, where they are incorporated into the whitelist. Thus, every host's whitelist has two components: an individual component capturing behaviors unique to that host, and a global component that corresponds to behavior that is common to the population. The global component can contain atoms that are not part of the individual host's regular behavior.

"Every host's whitelist has two components: an individual component capturing behaviors unique to that host, and a global component that corresponds to behavior that is common to the population."

Detection Algorithm

At a high level, our system generates alarms corresponding to two types of events. These are classified as (1) p-alarms, when a destination atom not contained in the host's whitelist becomes persistent and (2) c-alarms, when a destination atom is observed at a large number of end-hosts in the same window and is identified as common. Note that p-alarms are generated locally; the user is alerted and asked to acknowledge the alarm. In contrast, c-alarms are raised either at the central ITOC or locally, if full whitelists are distributed among peers. Note that when the alarm corresponds to an atom becoming significant, one of two things must happen: either the atom is classified as benign (by a user or operator) in which case it must be added to the appropriate whitelist, or else the alarm indicates malicious behavior, requiring remediation action. In this article, we do not address the remediation stage; we simply note that a number of possibilities have been suggested in the literature, such as throttling traffic, redirecting traffic through a scrubber, blocking traffic, and so on.

```
processPacket(pkt, t, wi)

a <-- getDestAtom(pkt)
if a in WHITELIST then
return /* ignore atoms already in the whitelist */
end if
if a is a new connection initiation then
DCT[a][currIdx] = 1      /*update persistence */
sendReport(userID, a, t)     /*report sent to central console*/
end if
```

Code listing 1: Outgoing Packet Processing
Source: Intel Corporation, 2009

In the rest of this section, we briefly review the specific actions required to process outgoing packets (summarized in Code listing 1). When the (outgoing) packet corresponds to an atom already in the individual host whitelist, nothing further is done. If the outgoing packet does not correspond to an atom already in the host whitelist, then the following steps are taken:

- If the atom was not previously seen, a new entry is created in the data structure used to track persistence (DCT); this is indexed by the atom and points to a bitmap. Each bit corresponds to a particular tracking window.
- The data structure that tracks the observations of atoms (labeled DCT) is updated for the current tracking window.
- The atom, if new, is sent to the ITOC (possibly after being filtered through a minimum persistence criterion).

"Note that our system is not tied to any particular traffic feature or threshold definition."

Note that our system is not tied to any particular traffic feature or threshold definition; for convenience, we assume connections per minute as the feature under consideration. To generate p-alarms, we track persistence at all the timescales by employing a sliding window. The data structure to do this is depicted in Figure 2. A dictionary (or hash table) is maintained, in which an atom is indexed, and this dictionary entry reveals the particular bitmap associated with the atom. When the atom is observed in a tracking window w_i, the i_{th} bit is set to 1 as described in Figure 2. As the sliding window is advanced, at the end of the last window, the persistence is computed for each atom observed in the last tracking window. It would seem that doing this for multiple timescales would be expensive. However, an interesting observation is that we do not need to replicate the structure at different timescales. Instead, we can exploit the overlapping nature of the timescales ($W_3 < W_4$); we can get away with this by using a single long bitmap that has enough bits to cover the longest observation window.

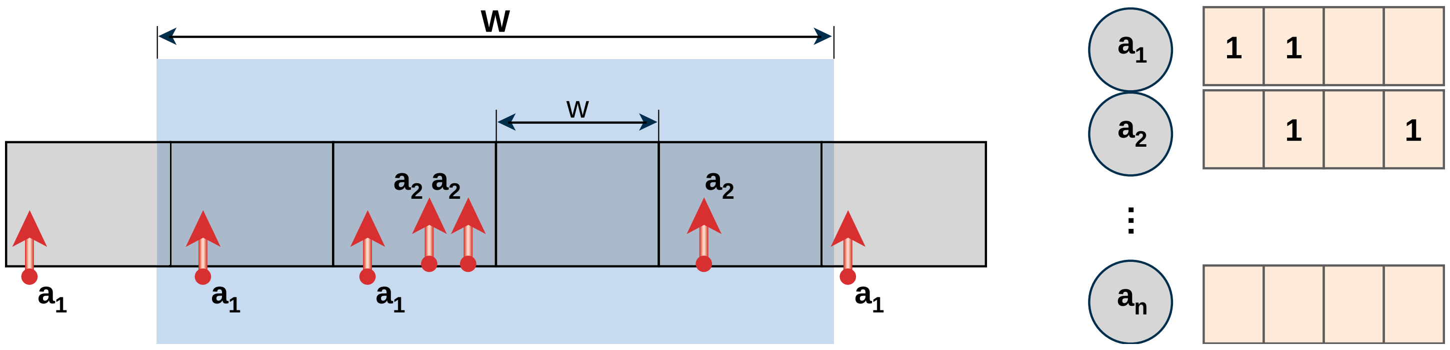


Figure 2: Data Structure Used to Track Atom Persistence
Source: Intel Corporation, 2009

If at any time, the persistence value of the atom exceeds the threshold p*, an alarm is raised for the atom; at this time, the user is asked to attest whether the atom is valid and should be added to the whitelist. If the value is not significant even after sufficient tracking windows, the bitmap is cleared out and the atom is no longer tracked (a new bitmap is instantiated if it ever appears again).

"If an outgoing packet is already in the whitelist, then no new dictionary entry is required."

To understand the overhead imposed by this procedure, we note that the length of the dictionary need not be large. If an outgoing packet is already in the whitelist (specifically, if its atom is in the whitelist), then no new dictionary entry is required. For everything else, we only need one entry per atom (even if the same atom has many connections or packets associated with it). With atoms that actually need to be tracked, the computation involved is simply the time it takes to index the dictionary and update the bitmap. However, we see in the traffic that most atoms that we track occur very infrequently (and that the most obviously persistent atoms are already in the whitelist and do not need to be tracked). Therefore, most entries in the bitmap are empty; an easy optimization would be to use sparse vectors in lieu of bitmaps. In our analysis, we found that the worst-case scenario over all users, and all observation windows W_{max} had 1435 atoms requiring tracking. The average case was 485 atoms. This is almost negligible if one considers the computational power and memory associated with modern-day mobile systems.

"Batching updates reduces the communication cost, but increases the time to detection. This is an interesting tradeoff."

We conclude this discussion by briefly discussing how the c-alarms are generated through tracking commonality—a very straightforward operation. The central console at the ITOC keeps track of atoms seen by different users over the largest observation window. When a report arrives from a host, the corresponding atom is updated. At the same time, old information is expunged (that is, sightings of an atom older than the observation window are discarded). When an atom's entry is updated, and the number of associated users (who have seen this atom recently) crosses the threshold c*, a c-alarm is generated. The frequency with which a host sends reports to the central console determines how soon an anomaly will be detected. Dispatching the report immediately (as soon as the atom is first seen) helps with catching the anomaly early, but at the cost of communication. Batching updates reduces the communication cost, but increases the time to detection. While this is an interesting tradeoff to study, we do not explore it in this article.

"*We detect three different types of anomalies: burst, persistent, and commonality.*"

Testing with Malware Traces

We present the results from running our detection algorithm with traces collected from real botnets. Recall that we detect three different types of anomalies: burst anomalies, triggered by large changes in traffic distribution; persistence anomalies triggered when destinations are communicated with regularly, even with very little traffic (such as botnet C&C channels); and commonality anomalies, triggered when a number of network users begin to exhibit correlated behavior. These anomalies correspond to the three types of alarms output by our system. Table 2 lists some well-known malware types, indicating what types of alarms are likely to result from each.

	Burst alarm	p-alarm	c-alarm
(long) DDoS attack	♦	♦	
DDoS attack	♦		♦
Scanning worm	♦		
IRC botnet	♦	♦	♦
Stealthy botnet		♦	

Table 2: Well-known Malware Types and Their Alarms
Source: Intel Corporation, 2009

"*One of our goals is to understand the detection of the* ***attack*** *behavior and the* ***channel*** *behavior.*"

Botnet Traces

We collected traffic traces from three distinct botnet families. We executed bot code on a host and logged packet traces for a week, by using the same host over multiple weeks to run the three different bots. The host was wiped clean in between collections, and a pristine copy of Windows* XP* was installed. Also, we turned off the auto-update functionality and configured the firewall to drop all incoming connections. From each trace, we discarded all packets that did not have a source or destination address corresponding to the host. The packet traces were converted to flows by using Bro [8], and the rest of the analysis uses flows. One of our goals in this section is to understand the detection of the different behaviors; that is, the *attack* behavior and the *channel* behavior (when the malware *calls home*). In the traces we collected, we saw both. Because many bots in the wild do not generate much volume (and try to remain undetected), detecting the control channel is of critical importance. We briefly describe the three Botnets and how the flows were classified:

SDBot. An SDBot is a well-studied botnet that uses IRC as the channel but on a non-standard port. However, the IRC servers are easy to pick out from the domain names, for example irc.undernet.org. The traces revealed two distinct atoms in the control flows. The remaining flows consist of scans being run on a neighboring network prefix. We noticed a large number of scans on ports 135, 139, 445, and 2097 (a well-known commercial anti-virus product). In the traces, we see connections on the well-known IRC ports and use this knowledge to identify control traffic (the IRC traffic) and attack flows.

Zapchast. This botnet also uses IRC as the channel and uses the well-known IRC ports (6666 and 6667). We saw a total of five *IRC service atoms* (about 13 distinct IP addresses) in the traces. The attack traffic was predominantly netbios traffic.

Storm. This botnet is P2P-based and very different from the others. The traces are two orders of magnitude larger than the other botnets. Lacking a single destination server or a well-defined port, it was quite hard to identify the control channels and we had to rely on some heuristics to do this: the fact that Storm uses UDP to connect to the P2P is documented.

We looked at distributions of the UDP flows (flows with two-way traffic) and noticed a very large number of packets that were of a small, fixed size (the flows were on non-standard ports and unlikely to be attacks). We took these flows to be an indicator of maintenance traffic and isolated all the ports involved. UDP flows to this set of ports are assumed to be part of the control channel. We did see a much smaller number of HTTP and SSH flows that may also be control related; the volume of these flows is such that it does not affect our results. The attack traffic for Storm is overwhelmingly on TCP port 25 (SMTP).

"Lacking a single destination server or a well-defined port, it was quite hard to identify the control channels."

Evaluation

In the rest of this section, we discuss the detection of persistence anomalies, and we defer the analysis of commonality anomalies due to space limitations.

Detecting stealthy behavior with p-alarms. To validate the detection of the control channel in each of the Botnets, we first identify the distinct atoms that can be extracted from the control traffic. For each of these atoms, we compute persistence over the lifetime of the (malware) trace. Recall that we compute this at five different timescales. For the purposes of detection, we consider the atom to be flagged as a p-alarm, if the value at any timescale exceeds the threshold $p* = 0.6$. We found that this threshold is associated with the fewest false alarms per day and the best detection rate, where the rates were averaged over all the destination atoms for all the malware traces.

In Figure 3, we plot the maximum persistence value for each of the atoms. The Y axis indicates the value used for $p*$. The scatter plot contains three distinct markers for each of the botnets, and each mark plots the persistence value for the corresponding atom. We plot a vertical line at $p=0.6$, which is the persistence threshold used by our detection system. Atoms that occur to the right of the vertical line are flagged by our system as possible C&C destinations. The particular threshold, i.e., $p=0.6$ was selected so as to achieve the best tradeoff between minimizing the number of false positives (i.e., normal, benign destinations flagged by our method as C&C destinations), and maximizing the detection rate (i.e., the fraction of C&C destinations that we correctly flag).

"For each of these atoms, we compute persistence over the lifetime of the (malware) trace."

The SDBot traces revealed exactly one atom, and this atom appears toward the top right of the plot. It is the largest marker and is shown as a triangle. The Zapchast traces contained exactly nine atoms, all but one of which appear to the right of the vertical line. Finally, the Storm traces contain approximately 82,000 atoms with persistence levels evenly distributed (for convenience, we only plot a sample of 100 atoms). While persistence is reflected on the x-axis, the vertical bands indicate different timescales. Thus, a point in the bottom band indicates the persistence value is associated with the 1-hr timescale.

We plot the maximum persistence for each destination atom, so the band indicates the timescale at which the persistence value maxed. Looking over the points, we see that the SDBot atom and eight of the nine Zapchast atoms are easily detected, appearing to the right of the threshold. For the single Zapchast atom to the left of the threshold, we noticed exactly two connections, close to each other, over the entire trace. We conclude that these connections do not really count as regular. We point out that these particular botnet instances are stealthy and generate very few connections. One of the atoms (to the right of the line) was associated with 30 connections over a whole week, with at most one connection in a window. This behavior qualifies as being close to indistinguishable. However, the persistence value for this atom is 0.7 and is above the threshold. This particular example drives home why a system such as ours is required to detect stealthy malware. With malware becoming more stealthy and with developers building in extraordinary measures to keep it from being detected, looking for volume-based anomalies is unlikely to have much success.

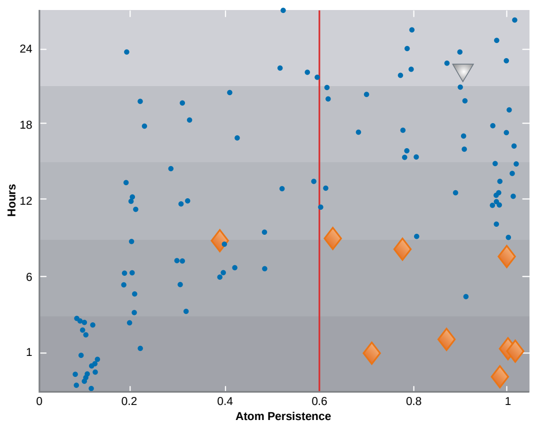


Figure 3: Detection by Persistence of Three Botnets
Source: Intel Corporation, 2009

Conclusions

With the rapid evolution of botnets toward increasingly stealthy behavior and the staggering numbers of end-hosts already infected by such malware, there is a dire need to develop and deploy techniques to counteract these problems. In this article, we reviewed the latest in botnet behavior and trends to elucidate the shortcomings of traditional approaches that depend on rule-based and/or volume-based detection. Bots and botnets are able to evade anomaly detection in part because they are polymorphic in nature and thus are considered a new vulnerability with every new sighting; their communication behaviors deliberately mimic that of normal end-hosts, and thus they stay below detector threshold settings.

"We are able to develop and present the Canary end-host detector, designed to root out the botnet command and control channel."

As a result, we analyze the behavior of real Intel enterprise end-host background traffic and contrast it to real botnet C&C channel activity. Consequently, we are able to develop and present the Canary end-host detector, designed to root out the botnet command and control channel by tracking the persistence of a node's relationships with destination hosts, and the commonality of persistence across multiple peers—both fairly stable properties of non-botnet traffic. The strength of these methods requires no *a priori* knowledge of the botnets that are to be detected, nor do they require traffic payload inspection.

References

[1] "An Inside Look at Botnets." Paul Barford and Vinod Yegneswaran. In *Series: Advances in Information Security, Springer, 2006.*

[2] Symantec. "2H 07 Threat Horizon Report."

[3] *USA Today*. "Botnet scams are exploding." March 17, 2008. At ***http://www.usatoday.com/money***

[4] Damballa. "Damballa announces discovery of Kraken BotArmy," April 7, 2008. At ***http://www.damballa.com***

[5] F-Secure. "Calculating the Size of the Downadup Outbreak." January 16, 2009. At ***http://www.f-secure.com***

[6] F. Giroire, J. Chandrashekar, G. Iannaccone, D. Papagiannaki, E. Schooler, and N. Taft. "The cubicle vs. the coffee shop: Behavioral modes in enterprise end-users." In *Proceedings Passive and Active Measurement Conference* (PAM'08), *Springer Verlag Lecture Notes in Computer Science*, pages 202-211, Volume 2979, April 2008.

[7] D. Dash, B. Kveton, J. M. Agosta, E. Schooler, J. Chandrashekar, A. Bachrach, and A. Newman. "When gossip is good: distributed probabilistic inference for detection of slow network intrusions." In *Proceedings of the 21st National Conference on Artificial Intelligence*, (AAAI'06), pages 1115-1122, July 2006.

[8] Bro. At ***http://www.bro-ids.org***

Acknowledgments

The development of the Canary detector was a collaborative research effort with Frederic Giroire, Nina Taft, and Dina Papagiannaki.

Author Biographies

Jaideep Chandrashekar is a Research Scientist at Intel in Santa Clara, CA. His general area of interest is communication networks and distributed systems. In particular, he has worked on Internet and end-host security, traffic measurements and analysis, and Internet routing. His recent work has focused on building security solutions that adapt to individual traffic patterns and distributed anomaly detection mechanisms; and he has investigated the energy footprint associated with network traffic. He joined Intel research in 2006 after receiving a Ph.D. from the University of Minnesota. His e-mail is jaideep.chandrashekar at intel.com.

Carl Livadas is a Research Scientist at Intel Labs. He is currently working on the Distributed Detection and Inference (DDI) project; a cyber-security project focusing on collaborative techniques among overlay peers to promptly and accurately detect malicious behavior. His current research interests include peer-to-peer systems, content-based networking, and cyber security. Prior to joining Intel, Carl worked at BBN Technologies on several cyber-security projects, such as Zombiestones, IPSPOOR, Stingray, and STARLITE. Zombiestones involved the network-based detection and identification of IRC-based Botnets. IPSPOOR involved a simple, light-weight, and effective router-based solution to the problem of IP packet traceback. Stingray involved the design and implementation of a network-based insider threat detection and investigation system. Finally, STARLITE involved the development of novel stepping-stone detection techniques. Carl received his Ph.D. degree in Electrical Engineering and Computer Science from the Theory of Distributed Systems (TDS) group at the Laboratory for Computer Science at MIT. His Ph.D. work involved applying formal techniques to model, analyze, and design retransmission-based reliable multicast protocols. Prior to this work, Carl worked on formally modeling and verifying the correctness and safety of hybrid systems, such as collision avoidance systems for commercial aircraft and autonomous vehicles. His e-mail is clivadas at alum.mit.edu.

Steve Orrin is Director of Security Solutions for Software Pathfinding and Innovation, a part of the Software and Services Group at Intel Corporation, and is responsible for security platforms architecture and security strategy and product direction. Steve joined Intel as part of the acquisition of Sarvega, Inc. where he was their CSO. Steve was previously CTO of Sanctum, a pioneer in Web application security. Prior to joining Sanctum, Steve was CTO and co-founder of LockStar, Inc. and SynData Technologies, Inc. Steve was named one of InfoWorld's Top 25 CTOs of 2004 and is a recognized expert and frequent lecturer on enterprise security. He has also developed several patent-pending technologies covering user authentication, secure data access, and steganography, and he has one issued patent in steganography. Steve is a member of the Information Systems Audit and Control Association (ISACA), the Computer Security Institute (CSI), the International Association of Cryptographic Research (IACR), and he is also a co-founder of WASC (Web Application Security Consortium) and a co-founder of the SafeSOA task force. He participates in several OASIS, and AFEI working groups. His e-mail is steve.orrin at intel.com.

Eve Schooler joined Intel in 2005. She is a Principal Engineer at Intel Labs. Presently she leads the Distributed Detection and Inference (DDI) project, an effort that focuses on collaborative anomaly detection in large-scale networks and that, more broadly, promotes the adoption of an end-host correlation framework that leverages the idea of measurement everywhere. Eve obtained a B.S. degree from Yale University, an MS degree from UCLA, and a Ph.D. from Caltech, all in Computer Science. Her broad interests lie at the intersection of distributed systems, networking, and scalable group algorithm design. Interested in protocol standards, Eve served on the Transport Directorate of the IETF, co-founded and co-chaired the IETF MMUSIC working group for many years, and is a co-author of the SIP protocol that is widely used for Internet telephony. Prior to Intel, she held positions at Apollo Computer, Information Sciences Institute, AT&T Labs-Research, and Pollere LLC. Her e-mail is eve.m.schooler at intel.com.

Copyright

DECENTRALIZED TRUST MANAGEMENT FOR SECURING COMMUNITY NETWORKS

Contributors

Meiyuan Zhao
Intel Corporation

Hong Li
Intel Corporation

Rita Wouhaybi
Intel Corporation

Jesse Walker
Intel Corporation

Vic Lortz
Intel Corporation

Michael J. Covington
Independent Researcher

Index Words

Certificate
Public Key
Social Networks
Identity
Mobility
Distributed Algorithms

Abstract

Emerging networking technologies, such as community networks, mobility, ad hoc connectivity, mesh, ubiquitous computing, and other infrastructure-less networks, have an urgent need for a first-class set of features that address the problems of direct device-to-device introduction, authentication, and trust management. Traditional approaches have proven to be inadequate to address these needs, because these approaches rely on centralized or managed infrastructures, which are brittle or cannot scale. In this article, we discuss the fundamental difficulties and system requirements for decentralized trust management, and we propose new technologies that enable the bootstrapping of trust, and, subsequently, the calculation of trust metrics that are better suited to these domains.

Introduction

The development of wireless technologies, mesh networks, ubiquitous computing, and ad hoc networking enables new applications to enhance the user experience, provide more flexibility, and give the user more choice. Network technologies, recently deployed, have increased user connectivity options, allowing users to connect from virtually anywhere. These changes create challenges for network security. Traditional centralized authority-based mechanisms do not work, or work at less than optimum, for emerging networking patterns. Novel network topologies, such as peer-to-peer (P2P) networking, void many of the assumptions made by established approaches to security, and these new topologies, therefore, require us to rethink the entire security architecture.

One example of this kind of network is mobile, ad hoc networks (MANETs) that provide wireless network services without relying on any centralized infrastructure. MANETs treat each node in the network equally, and each node acts as both a client and a server node. The network topology is emergent, based on collaborative routing. Because of the emergent connectivity, a single centralized authentication server may not be reachable at all times.

"Traditional centralized authority-based mechanisms do not work, or work at less than optimum, for emerging networking patterns."

Another example is P2P networks, widely used to share data and resources. As of 2006, over 80 percent of all Internet traffic consisted of P2P network traffic, and this percentage appears to be steadily growing. The network structure of a P2P network is also self-organizing, is typically unmanaged and unplanned, is unsupported by any dedicated support staff or servers, and is utilized by potentially very large numbers of users. The operation of these networks is distributed and autonomous. Interjecting a traditional centralized authentication scheme, therefore, into these sorts of networks would impose a centralized control structure and require permanent on-line servers and support staff to manage them, thereby undermining the emergent, unmanaged character that makes these networks so attractive to their users. Instead, what is needed is a trust management system that matches the emergent nature of these networks, and one that is based on collaborative individual decisions.

"What is needed is a trust management system that matches the emergent nature of these networks."

In this article, we examine centralized authentication systems and analyze the reasons why these systems fall short for new classes of networks. We argue that a centralized authority that creates and manages all of the identities for a domain is too inflexible to support self-organizing networks, where relationships are emerging through individual interactions. We call for a new approach wherein identities are created to signify the relationships, and entities collaboratively manage and evolve trust, based on these relationships.

We propose a decentralized trust management framework that manages identities to support authentication in self-organizing networks. This framework contains several key functions: evidence collection and distribution, identity generation and auditing, and trust calculation. Within this framework, every node collects trust evidence locally and shares information with peers. Trust decisions are made locally, based on collected information. The global consensus of trusting identities is reached by peer interactions and trust calculation. We emphasize the need to bootstrap trust relationships in order to build practical trust-management systems. For managing and propagating trust, we propose a novel trust model to calculate trust, based on both first-hand observations and on second-hand opinions from peers. This trust calculus model has two unique features: (1) support for both positive and negative trust values; and (2) whenever possible, pre-existing relationships, such as those between devices and their users from other contexts and communities, are transformed into new relationships between devices in new communities.

"We call for a new approach wherein identities are created to signify the relationships, and entities collaboratively manage and evolve trust."

We further analyze some threats, such as identity attacks, to the decentralized trust model, and we propose using device profiling to build consensus on binding a device identifier with its profiling attributes. We further discuss candidate attributes that can be used to thwart certain identity attacks.

Organization of this Article

Firstly, we analyze the trust-management challenges in current centralized models. We propose a decentralized trust-management framework for identity management and authentication and describe several of its main functions. Then, we describe applications of the basic trust model functions to bootstrap trust relationships, identities, and trusted communities. Next, we illustrate the design of community vouchers as a means to propagate trust, by using trust-calculation algorithms. We then analyze the identity attacks on our decentralized trust-management system and suggest some attack-detection methods. We finally present a discussion of related works, and we conclude the article with a look at our future research directions.

Decentralized Trust-Management Problems

Traditional Trust Models and Problems

"It has become evident that the dominant centralized security models fail to meet the challenges presented by the P2P communication patterns."

The current practice of managing trust and authentication is designed to efficiently address the needs of enterprise access control. A centralized server is deployed to perform all authentication procedures, such as X.509v3 [1] certification authority, revocation servers, on-line certificate status protocol (OCSP) servers, or RADIUS servers [2]. Having a single central server simplifies credential management and makes it easier for the organization to enforce its access-control policies. However, it has become evident that the dominant centralized security models fail to meet the following challenges presented by the P2P communication patterns in self-organizing networks:

A centralized authority has to be available all the time. Given the dynamic nature of new networking forms, it is impossible to guarantee that this centralized authority can be reached from everywhere in the network.

A single point of control makes it harder for users to communicate with the domain. The centralized authority generates and manages all the identities and credentials for the domain. This design forces users to contact the centralized authority for every enrollment and authentication activity. In self-organizing networks, communications may happen only in a local context, where contacting the centralized authority is impossible or, at best, very inconvenient.

Centralized trust models demand long-lived trust evidence. IT administrators often hold the view that computing devices belong to a single administrative domain, so that credentialing happens only once during the lifetime of a device, at most. The resulting identity credentials have to fit all usage cases. This increases the cost of gathering and maintaining evidence, increases the possible damage if credentials become compromised, and makes it difficult to re-evaluate trust evidence. In self-organizing networks, an entity's relationship with a particular domain may be dynamic and transient. Such relationships require frequent and on-line trust evidence re-evaluation.

A centralized authority imposes a single trust metric for the entire domain. This means that a name is bound to a key. However, in self-organizing networks, the trust evidence is not uniform. Evidence may be in the form of keys, names, hardware attributes, and even social relationships. Hence, evidence evaluation cannot be uniform either.

Traditional centralized security models require the domain to be established at a central place by some authority. In unmanaged networks, however, the trust relationships are formed at the grassroots level and from P2P interactions. A corresponding trust model needs to be built to match this pattern.

The centralized model enforces a uniform relationship between the individual named entities and the organization running the central server, not between different devices within the organization. This is at variance with the needs of devices in new usage models such as P2P networks, where each device needs some means to directly manage its relationships with other devices in the community.

Prior research on security models has proposed decentralized trust models that remove the dependency on the centralized authority and servers [3, 4, 5, 6, 7, 8, and 9]. However, most of the existing literature only focuses on the trust calculation models, which evolve and propagate trust on entities, based on a transitive property of trust. Decentralized trust calculations only address part of the problem. A complete and practical decentralized trust management system demands solutions to the following three additional problems: 1) trust evidence gathering; 2) trust evidence evaluation for initial trust computation; and 3) the creation of trusted communities.

"Most of the existing literature only focuses on the trust calculation models, which evolve and propagate trust on entities, based on a transitive property of trust."

Decentralized Trust-Management Framework

Requirements

We propose a decentralized trust-management framework for managing trusted member identity in self-organizing networks. The trust-management system creates and manages a trusted domain, called a trusted community. This framework satisfies the following requirements:

Removes dependency on a centralized authority. We envision a completely decentralized system in which every node in the domain has the potential to be a naming authority. Many authorities in a domain allow users to join and use the system from anywhere in the network.

Makes on-line evidence distribution a first-class ingredient. All members in the system provide a variety of evidence to help calculate the initial and ongoing trust and reputation of other members. It is important to provide on-line mechanisms so that members can distribute trust evidence, in order to build a practical trust-management system.

Provides accountability. The trusted community uses the authentication procedure to enforce accountability that cannot be repudiated, for actions performed by a member of the domain. To achieve accountability, identities must be individual, unique, and undeniable, within the administrative domain.

Treats relationships as a central component of the network. The usage of identities in P2P communications is often relevant only to the communicating parties within the domain. Identities created and managed in the trusted community should signify P2P member relationships and members' relationships with the trusted community, via the relationship the credential issuer maintains with the community.

Decentralized Trust-Management Paradigm

At a high level, a trust-management framework should provide the following functions to be able to secure communities:

- Bootstrap, or create, the community.
- Enroll members in the community.
- Authenticate members to one another.

The framework mechanisms to generate and manage identities for each member should be decentralized. They must permit members to authenticate and securely communicate with each other through mediation of any other members in the trusted community. Members self-organize the trusted community by using trust calculations to evaluate their evolving relationships within the community.

"Members self-organize the trusted community by using trust calculations to evaluate their evolving relationships."

Relationships exist between roles within a community. Each community contains *members* who are entities that can participate in community activities with full member privileges. *Issuers* are special members. In addition to ordinary member privileges, issuers have the responsibility of generating *identities* for new members. We represent identities with identity certificates that can be used as authentication credentials within the community.

Authentication within the community is achieved by the holder of an identity credential issued by a recognized issuer proving possession of a key that is bound to the credential. Authentication fails if the authenticated party does not possess a certificate from an issuer who is recognized by the authenticator as such.

A member may recognize all or some subset of the other members as issuers in the community. When a member is first enrolled in the community, this member's enroller is, by default, the first issuer recognized by the member. A member may gain trust and recognize new issuers by collecting trust evidence from the community and applying the evidence to a *trust calculation*. A member can become an issuer as a result of a consensus reached by a group of issuers and members; this group is called the issuer's *election committee*. The election committee can consist of a single member, but an issuer has little impact unless it is recognized broadly across the community. Members of the election committee execute a joint trust-calculation algorithm to elect an issuer. Once elected, an *issuer certificate* is issued to the new issuer, signed by each of the electors. The purpose of this signed certificate is to demonstrate that the issuer is recognized by more than a single member. However, having the certificate signed by multiple parties raises security issues about collusion and about members with multiple identities, which are discussed later in this article.

"Members of the election committee execute a joint trust-calculation algorithm to elect an issuer."

Founders are special issuers who initially establish the trusted community. Founders recognize each other as founders, issuers, and members.

Lastly, some entities are treated as *guests* in the community. Any community member can introduce new entities to other community members as guests. An issuer can enroll a guest as a member by following certain procedures. Creating the guest role allows a newcomer to participate in a trusted community without having to first contact an *issuer*.

It is important to stress that community credentials provide more than just identities for community members.

Credentials are relationship signifiers. Each one signifies a relationship between the member and its issuer. In this model identity is not necessarily who you say you are, but rather who your issuer says you are. Other members of the community believe this identity because it is asserted by an issuer they trust. The community collectively holds its issuers accountable for issuing identities that do not conform to the rules of the community.

In this trust-model, framework, each entity implements the same set of trust-management functions: 1) trust evidence collection and distribution; 2) identity generation and auditing; and 3) trust calculation and propagation of identities. In addition, each entity maintains a community database locally that stores the entity's knowledge about the trusted communities it belongs to. Some examples of the type of data stored in the community database are community name, policy, recognized identity certificates, recognized issuer certificates, and relevant trust evidence.

Figure 1 illustrates the three major trust-management functions and their relationships. Each entity collects evidence through its own measurements, and also through communication with other entities within the community. Entities use on-line, evidence-distribution mechanisms to share evidence gathered from either first-hand observations or from recommendations by other community members they trust.

Once sufficient evidence is collected, an entity can compute an initial trust value for the target entity. For initial enrollment, an issuer generates a name label that is based on collected evidence. The issuer binds the name label to the relevant evidence in a credential for the target entity.

For authentication, the identity auditing mechanisms ensure that the entity uses its identity properly, the one that is laid out in the authentication and authorization procedures. In particular, the entities need to pay close attention to identity-related attacks (discussed later in this article), and they need to utilize certain approaches to detect or mitigate identity attacks. The auditing function requires knowledge of both the name label and trust evidence that is bound to the entity's identity in the community.

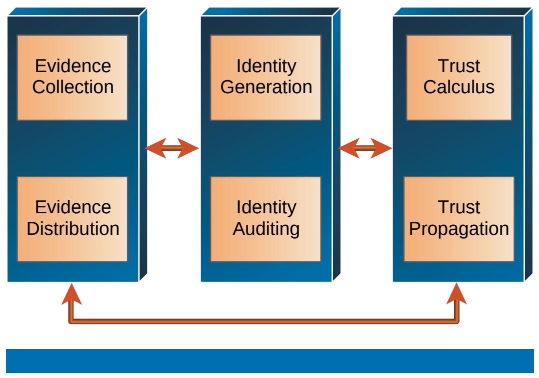


Figure 1: Distributed Trust Framework for Identity Management
Source: Intel Corporation, 2009

"Credentials are relationship signifiers."

"For authentication, the identity auditing mechanisms ensure that the entity uses its identity properly."

Lastly, each entity builds its trust value or reputation upon joining the community. To establish and update trust on an existing member, the entity utilizes trust calculus to incorporate opinions from its peers, as well as first-hand observations. More specifically, the main goal of trust calculus is to compute the trust value on a member without communicating directly with the member. Often, this is done by utilizing the transitive property of trust. Later on in this article, we introduce a novel trust-calculation model that has two unique features: it supports both positive and negative trust values, and it utilizes relationships among devices and their users inherited from other contexts to root trust whenever possible. The outcome of the trust calculation is fed back to trust-evidence collection and auditing components to further improve the knowledge of trust evidence and the binding of entity attributes with the identity of the entity.

"The main goal of trust calculus is to compute the trust value on a member without communicating directly with the member."

In the remaining sections of this article, we discuss in detail our vision for these three components with our focus on bootstrapping trust and propagating trust in the community.

Bootstrapping Trust in the Community

Trust Evidence and Evidence Collection

Bootstrapping trust starts from collected trust evidence. In this scenario, any information useful for computing trust value on an entity is referred to as trust evidence. In general, there are three types of evidence:

- *First-hand observation or measurement*. The observer can collect such evidence by direct interaction with the target entity. For instance, MAC address, hardware identifier, and past network activities associated with these hardware identifiers are observable directly.
- *Second-hand observation*. Members can share their first-hand observations with each other to help trust calculation.
- *Recommendations*. Members can also share their opinions on other entities with each other. Recommendations result from the distillation of evidence by the trust calculations of other members. For instance, one member may recognize another as an issuer. By doing so, the first member has decided that certificates issued by the issuer are an acceptable form of identity within the community. Responding to a query from another member for the issuer's certificate is its way of *recommending* the issuer.

"A general requirement is that trust evidence be distributed to wherever it is needed as quickly as possible."

The mechanisms to share trust evidence are critical to enable decentralized trust calculation. A general requirement for trust evidence sharing is that trust evidence be distributed to wherever it is needed as quickly as possible. Several P2P communication systems have the potential to satisfy this requirement, such as ant-based routing, content-based information routing, publish and subscribe systems, delay tolerant transport protocols, P2P file sharing, P2P gossip protocols, and so on. Part of our future work is to choose appropriate protocol and communication primitives to support distributed trust evidence sharing.

Bootstrapping Trust from Evidence

Device credentialing requires the creation of a relationship with a new community. This problem may be broken down into two separate problems. Firstly, the device must somehow be able to unambiguously recognize the community, and the community must somehow be able to recognize the device. Secondly, both the owner of the device and the community have to agree to form the relationship signified by the credential being established. Solutions to the first problem, such as Wi-Fi Protected Setup [10], utilize an out-of-band channel to exchange some sort of setup key, which cannot be forged, between the device and the administrative domain. The solution to the second problem requires human choice: both the device owner and the new community must somehow indicate their desire for this new relationship. (Solutions such as factory-installed identities remove human choice, thereby defeating the essential purpose of authentication credentials.)

"The device owner and the new community must somehow indicate their desire for this new relationship."

All electronic identity establishment mechanisms require an out-of-band channel to be secure. The purpose of the out-of-band channel is *demonstrative identification*; that is, the out-of-band channel establishes beyond a doubt that the identity is being assigned to the intended party. We believe pre-existing relationships can serve as the out-of-band channel in many circumstances. However, this is not always possible. A newly purchased device, for instance, has no prior relationship with any organization deploying it; therefore, it lacks any useful credential to bootstrap a new relationship. Consequently, mechanisms, such as Wi-Fi Protected Setup, will always be needed. However, this lack of credentials is no longer true after a device has been enrolled in even a single administrative domain. A device with a credential has a relationship with one administrative domain, and this relationship can be used as a basis for forming relationships with other members of other administrative domains who possess relationships with the first.

"A device with a credential has a relationship with one administrative domain."

To illustrate this point, suppose $\boldsymbol{A}$ and $\boldsymbol{B}$ both belong to community $\boldsymbol{C_1}$, while $\boldsymbol{A}$ also belongs to community $\boldsymbol{C_2}$. If both $\boldsymbol{B}$ and $\boldsymbol{C_2}$ desire that $\boldsymbol{B}$ become a member of $\boldsymbol{C_2}$, then $\boldsymbol{A}$ can use its relationship with $\boldsymbol{B}$ in $\boldsymbol{C_1}$ as evidence in $\boldsymbol{C_2}$. That is, $\boldsymbol{A}$ can use this relationship to assert that it has demonstrably identified $\boldsymbol{B}$ as evidence supporting $\boldsymbol{B}$'s request to join $\boldsymbol{C_2}$. Similarly, $\boldsymbol{B}$ can use $\boldsymbol{A}$'s recommendation of an issuer $\boldsymbol{I}$ in $\boldsymbol{C_2}$ as evidence that it is indeed joining $\boldsymbol{C_2}$, by accepting an identity certificate from $\boldsymbol{I}$. In order to protect the confidentiality of the relationship in $\boldsymbol{C_1}$ between $\boldsymbol{A}$ and $\boldsymbol{B}$, the new credential should not indicate which external relationship was used to identify $\boldsymbol{B}$ to the $\boldsymbol{C_2}$ relationship. On the other hand, $\boldsymbol{A}$ is revealing something about its own relationships within $\boldsymbol{C_2}$ by recommending an issuer to $\boldsymbol{B}$, but our assumption is that relationships within a community are open to all members for inspection. This is in keeping with current practice, as the authentication credentials provisioned by existing manual solutions to the problem do not inherently reveal the human relationships used to provision the credentials.

"Our assumption is that relationships within a community are open to all members for inspection."

Community Formation

A practical trust-management system should address the problem of bootstrapping the trusted domain, a central concern in any decentralized trust models. Here we introduce flexible approaches to establish a new trusted community. The key consideration is to reduce the cost of forming a trusted community. More precisely, we are looking into approaches that require only a limited amount of information and human intervention.

"The key consideration is to reduce the cost of forming a trusted community."

In some cases, a trusted community can be constructed with only one party, but one-party communities are usually not very interesting. Instead we illustrate the more general case where two or more parties create a new community through the following steps:

- Establish an initial trust relationship, as described earlier, based on some form of demonstrative identification, including the use of credentials from some other community.
- Run a commitment protocol to agree on the following parameters:
  - Community name, as an arbitrary name string
  - Community policy, stating membership and naming rules
  - Founders' identity information
  - Lifetime of the community

The string that identifies the community is the concatenation of community name, founders' public keys, and the community policy description. The public keys are included to uniquely identify the community.

- Create credentials to represent the community. The founders create and sign a special community certificate that contains the information of the newly established community. The founders also recognize each other as issuers and members of this new community.
- Exchange credentials and update the community database. Each founder inserts the newly-generated community certificate, issuer certificates, and member certificates into its own community database.

"At this point, the founders successfully establish a community by themselves and are collectively responsible for the management of the community."

At this point, the founders successfully establish a community by themselves and are collectively responsible for the management of the community. Issuers propagate this community information and certificates to new members as part of any successful enrollment ceremony.

By using this procedure, parties can self-organize trusted communities. This procedure is flexible and allows for parties to establish long-lived or transient communities to suit different kinds of secure applications.

Managing a Trusted Community

Human societies have been generally successful in managing trust and reputation. People have always self-organized into resilient communities for information exchange, and these self-organizing models have won the test of time [11]. In this section we describe an approach that models trust by using the exchange of reputation and trust information based on social networks [12].

Utilizing Social Networks: Community Vouchers

Earlier we described a web-of-trust model. Several attempts have been made to create viable web-of-trust models, such as *Pretty Good Privacy* (PGP) [9]. There are several problems with the models proposed to date, such as the collusion and identity problems already noted. Problems also occur because in every community, members depend on other members for trust. The recommendations are not weighted, and newly enrolled members may have difficulty finding any existing members whose recommendations are believable. Furthermore, the notion of *negative* recommendations, which is essential in defending against malicious members, does not exist in the existing web-of-trust models. We propose calling all forms of recommendations *vouchers*, because we see a recommendation as a way for one member to *vouch* or not for another member.

"The notion of ***negative*** *recommendations, which is essential in defending against malicious members, does not exist in the existing web-of-trust models."*

In Figures 2a and 2b, each solid line between two members represents an existing trust relationship. Figure 2a represents a community that has accreted naturally into three clusters: a cluster between two members labeled Cluster_i, a random network cluster labeled Cluster_r, and a mesh network labeled Cluster_m.

If member ***F*** intends to establish a trust relationship with member ***A***, any combination of the following existing trust relationships can be utilized by ***A*** to provide evidence for trusting ***F***: (1) the trust relationship between ***F*** and ***I***, because ***A*** accepts ***I***'s vouchers, (2) the trust relationship between ***F*** and ***D***, because ***A*** accepts ***D***'s vouchers, or (3) the trust relationship between ***F*** and ***S***, because ***A*** accepts ***S***'s *vouchers*.

"If member ***F*** *intends to establish a trust relationship with member* ***A****, any combination of existing trust relationships can be utilized."*

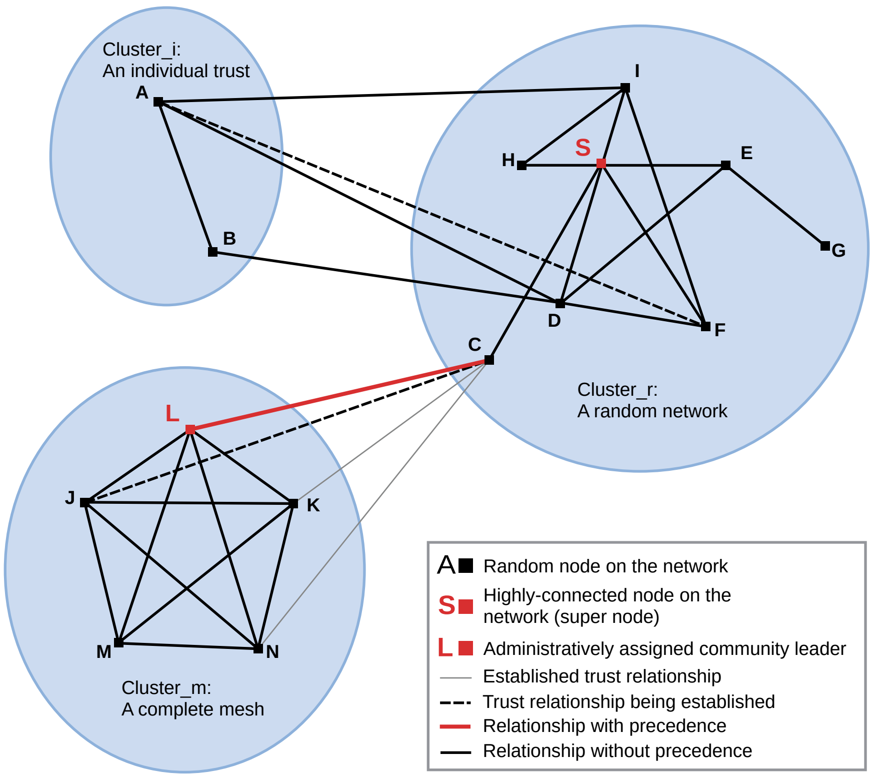

Cluster_i:
An individual trust
A
B
I
S
H
E
G
D
F
C
Cluster_r:
A random network
L
J
K
M
N
Cluster_m:
A complete mesh
A ■ Random node on the network
S ■ Highly-connected node on the network (super node)
L ■ Administratively assigned community leader
Established trust relationship
Trust relationship being established
Relationship with precedence
Relationship without precedence

Figure 2a: Community-based Trust – Formation of Trust Communities
Source: Intel Corporation, 2009

Of course, in general ***A*** does not have *a priori* knowledge about any of these relationships, and it faces the problem of how to expeditiously obtain the evidence it needs about ***F***'s trustworthiness. The solution? ***A*** should query all of the other community members it considers trustworthy that are presently on-line. Other community members can send their vouchers. Whether another member has the evidence ***A*** needs depends on that member's connectivity and experience within the community. Figure 2 depicts ***S*** as more highly connected than other members, so it is likely ***S*** can respond to more requests.

By responding to more requests, ***S*** can build a *reputation* for fast or more complete or authoritative responses, because it has access to more sources of evidence (***S*** may instead build a negative reputation by passing information that later proves to be inaccurate). A member such as ***S*** builds its reputation based on the natural evolution of relationships within *social* communities. These reputations can be used to weight trust decisions.

*"A member such as **S** builds its reputation based on the natural evolution of relationships within **social communities.**"*

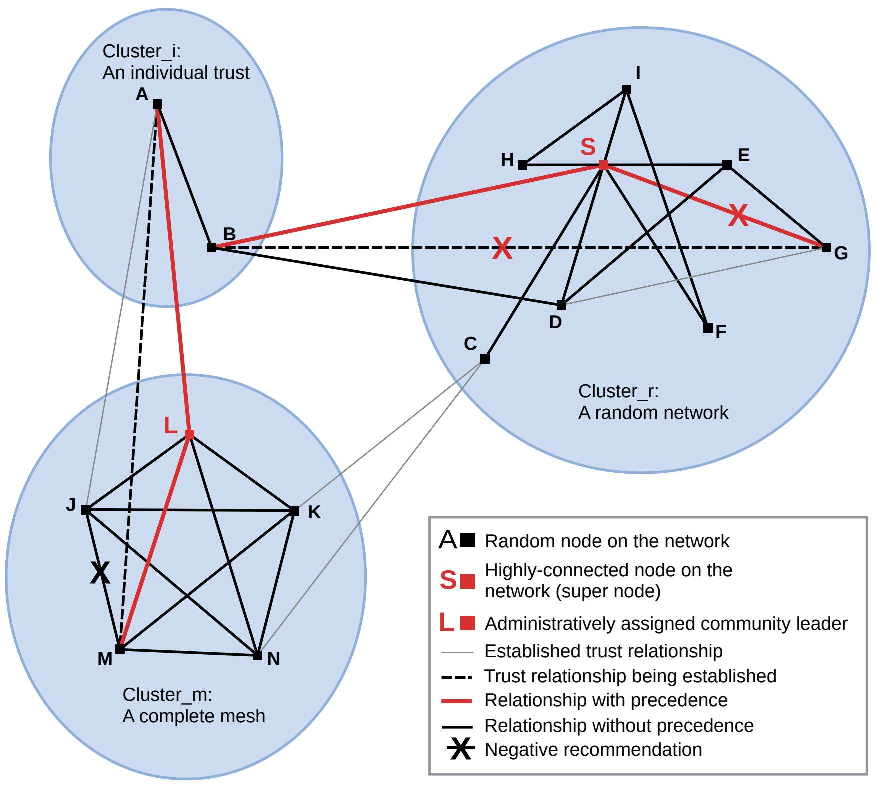

Cluster_i:
An individual trust
A
B
Cluster_r:
A random network
I
S
H
E
G
D
F
C
X
X
Cluster_m:
A complete mesh
L
J
K
M
N
X
A ■ Random node on the network
S ■ Highly-connected node on the network (super node)
L ■ Administratively assigned community leader
Established trust relationship
Trust relationship being established
Relationship with precedence
Relationship without precedence
Negative recommendation

Figure 2b: Community-based Trust – Negative Endorsement
Source: Intel Corporation, 2009

Trust Calculation

In addition to making decisions on whether or not to trust another member, the weight factor can also be used to determine trust levels. Trust levels fluctuate dynamically. For example, a voucher from a highly connected member such as ***S*** may make ***F*** more trustworthy to ***A***. A higher trust value is therefore given to ***F*** in ***A***'s trust list, because ***S*** is likely to have more evidence about ***F***'s past behavior. By reporting helpful vouchers more frequently than other members, ***S***'s trust level will increase.

"Trust levels fluctuate dynamically."

There are many ways to represent levels of trust. One representation can be trust rings, where each node creates several rings around it that represent levels of trust, and the device sorts other members into the most appropriate ring. A member places the peer members it trusts the most (that is, those that it maintains the closest relationships with) in the inner-most ring, followed by its *friends* in the next ring, and so forth. A neutral ring denotes a neutral trust state, while the outer rings represent mis-trust. A member can move between rings depending on the change in its relationship with other members. Trust, from a member's point of view, can have the following form:

$$\tau_{ax} = \sum_{\forall pos\_rings} \alpha_r \sum \tau_{ay} \tau_{yx} + \sum_{\forall neg\_rings} \beta_r \sum_{y} \tau_{ay} \tau_{y}$$

"Trust rings provide a simplified version of assigning trust for a node."

where τ_{ay} denotes how much member ***A*** trusts ***B***, $\tau_{ay} < 0$ denotes mis-trust, and α_r denotes the weight associated with a trust ring. Using this equation, if member ***A*** receives several vouchers for member ***B***, ***A*** will use the trust ring of the members vouching for ***B*** to resolve conflicting vouchers. ***A*** then uses its trust calculation to decide whether to transfer ***B*** between rings. Highly-connected members tend to migrate toward the innermost ring (at least if the vouchers they provide report good information over time), resulting in their *opinions* weighing more. Trust rings provide a simplified version of assigning trust for a node, especially in situations where the node does not have the computing power to assign and maintain individual values for all nodes it encounters. It provides the flexibility of grouping nodes according to trust levels.

As an example, in Figure 2b, node ***L*** trusts member ***M*** with a value of *3*, while ***J*** mistrusts ***M*** with a value of *-2*. Member ***A*** places ***L*** and ***J*** at level *5* and *2.5*, respectively. When ***A*** computes the trust level for ***M***, the negative endorsement of member ***J*** will be subtracted from that of member ***L***. This will result in a positive trust value τ_{AM} , which will probably not be high enough to place it in the inner circle due to the negative endorsement of ***J***. In addition, the trust levels (τ_{ab}) are dynamic variables, which makes the equation adaptive to changes in the communities (for example, nodes moving out of current communities or forming new communities).

"Trust rings provide a simplified version of assigning trust for a node."

A member's cumulative rating across all other members' rings represents its reputation within the community. Reputation requires maintenance of a relationship history, so is not free. We believe reputation makes for a good default trust value when no other information is available.

Micro and Macro Trust System

In addition to trust between members, we extend the trust model to a second dimension, where trust is calculated in a vertical (layered) manner, and reconciled to horizontal trust (for example, nodes virtual networks). Figure 3 illustrates the concept with a system comprised of three entities, and each entity is comprised of multiple components. For example, entity ***A*** consists of a user, a device, a virtual machine, a host operating system, and an application running on the device. A vertical trust relationship exists between the respective components of entity ***A***; in the meantime trust also exists horizontally, such as between entities.

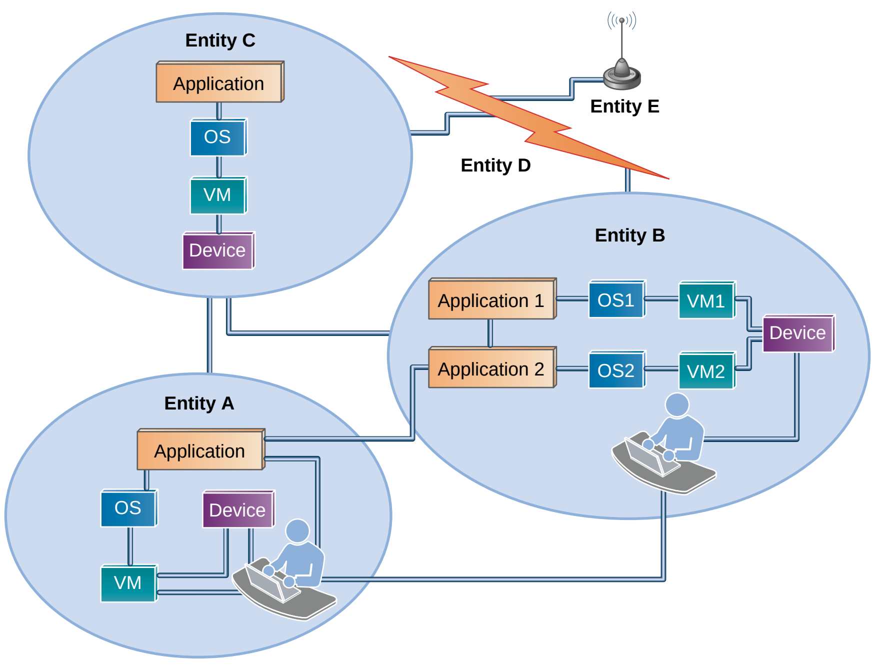


Figure 3: Micro and MacroTrust
Source: Intel Corporation, 2009

In existing technologies, individual components completely trust all the components of other trusted entities. This increases the risk when a component on a trusted entity is compromised. Our approach, on the other hand, applies trust propagation among individual components inside entities as well as among entities as a whole, and trust levels among different components within an entity can be independent. For example, assume that entity ***A*** is running a web browser, while entity ***B*** is running a web server, with a database backend. As a result, the trust of the browser in the database server on entity ***A*** is a function of its own trust level for the web server and a trust voucher on the database, while Entity ***A*** and Entity ***B*** as a whole may have different levels of trust between themselves. We assume no explicitly defined trust level between the application and the device. However, trust can propagate from the application to the device through the user, the operating system (OS), or a virtual machine (VM). In order for such a system to be usable, it is essential to have a network that is manageable to the degree that you can continue to model the interconnections based on social networks, where each entity can be treated as a community.

"In existing technologies, individual components completely trust all the components of other trusted entities."

Identity Attacks on Decentralized Trust Models

"A party might acquire multiple identities from different issuers."

Traditionally, it is easy to assign identities to devices and users within a community: choose one member to issue an electronic identity to each new member. With centralized credentialing, the major worry is impersonation attacks, where one party steals and uses the identity of another. Completely decentralized domains introduce new challenges, as a party might acquire multiple identities from different issuers. The exchange of trust evidence and trust values in our model is based on the assumption that distinct members have unique identities, and their opinions are independent; this assumption is critical to our model. Without this assumption, entities cannot be held responsible for either their actions or for the recommendations they render. In this section we discuss several common identity attacks and potential countermeasures.

Identity Attacks

Distributed trust calculation is particularly sensitive to three kinds of identity attacks:

- *Masquerade*. This kind of attack allows an attacker to use the identity of a legitimate community member. Masquerading attacks violate the requirement in trust calculus that recommendations can be attributed to the member allegedly sharing an opinion.
- *Sybil*. This kind of attack occurs when the attacker uses multiple identities simultaneously in the same community to take advantage of distributed trust calculations. Sybil attacks violate the requirement in the trust model that every member should contribute only one vote to each trust decision. Arbitrary Sybil attacks allow the attacker to disproportionally weight its own contributions to a trust calculation, thereby increasing its own influence beyond what it is entitled to, based on its reputation.
- *White-washing*. This kind of attack is similar to that of a Sybil attack, but it differs in that the attacker quickly changes its identity to avoid the consequences of its own actions. White-washing attacks violate the same fairness requirement as the Sybil attacks. In addition, they also violate the accountability requirements in the trust model, that state that each member should commit to the consequences of its own actions, including contributing opinions to the trust evaluation. This commitment is required so that community members can build stable trust relationships within the community. If the entity changes its identity quickly, its actions are not accountable, and the consequences of its actions cannot be bound effectively to the perpetrator.

"The accountability requirements in the trust model state that each member should commit to the consequences of its own actions, including contributing opinions to the trust evaluation."

Countermeasures for Identity Attacks

Potential countermeasures to identity attacks belong to two general categories: prevention or detection.

In centralized trust management systems, it is relatively easy to deploy a prevention mechanism to stop attackers from creating multiple identities illegally, because all the identity credentials are generated by the centralized authority. In the decentralized case, however, detection is a more effective countermeasure. One of our research hypotheses has been that the consensus-building nature of our trust model makes it suitable to detect Sybil and white-washing attacks. The goal is to minimize the risk that a member can illegitimately deny an identity previously acquired from the community.

"Community members build the initial binding of the device profile and device name, propagate this knowledge, and eventually build the communal consensus on the bindings."

In our trust model, trust requires consistent attribute usage, so a communal consensus about each member's attributes becomes feasible, and access to the community's resources can be regulated by the relationships maintained through time. The trust on an identity is established by building a device profile of attribute usage and verifying that the device profile is consistently mapped to the acquired device name. In other words, the community members build the initial binding of the device profile and device name, propagate this knowledge, and eventually build the communal consensus on the bindings. If the attacking device uses a different identity by modifying any of the bound attributes, this will be detected by other community members, who can then deny the attacker resources afforded through existing relationships within the community.

In order to detect the mis-bindings, the device profile contains a set of measurable device attributes. It is not required that a single attribute be able to uniquely identify the device. Together, the combined probability of forming a unique device identifier should be reasonably high. In particular, it is preferable that the attributes are tied to device hardware or the surrounding physical environment. In effect, the cost of creating a new identity for the device is close to the cost of changing all the hardware attributes in the device profile. Consequently, buying a new device becomes probably the only viable option for the attackers to create a new identity. Next, we suggest three types of attributes that can help with device profiling.

"It is preferable that the attributes are tied to device hardware or the surrounding physical environment."

Attributes for Identifying Other Devices

We examine three types of attributes for machine identification: radio attributes, hardware platform attributes, and behavioral attributes, such as network activity:

Radio attributes. Wi-Fi, Bluetooth, and other radio-based communications devices are now ubiquitous in mobile devices. Radios may have a number of attributes that can be measured and shared:

- Received signal strength indication (RSSI). RSSI is a transient attribute that could be used to detect some types of Sybil attacks. Cheriton and Faria report [13] that the signal strength measurements of a target by different receivers consistently correlate; [14] and [15] suggest a similar technique. This means it should be feasible to detect whether a device is changing low-level identifiers such as a MAC address. A community member utilizing shared RSSI values measured from the target devices can decide whether the frames sent, using different identities, render the same RSSI profile and therefore match the same physical device. We plan to design a distributed solution that utilizes a subset of real-time RSSI data.

- A second line of investigation shows that every radio has unique fingerprints. A fingerprint is a measurable characteristic, such as the rise time of the first symbol beginning a radio transmission. In [16], Xiao et al. propose using radio fingerprints as a way to recognize devices. If radio fingerprinting proves to be practical, it could be used to detect identity attacks. In particular, a radio fingerprint could be bound to an identity certificate and the radio fingerprint database that was searched, prior to issuing a credential to a new party joining a community.

Hardware measurements. A typical personal computer contains a list of hardware identifiers or serial numbers to identify each piece of hardware inside the computer. There are two challenges inherent in using such information. First, these identifiers should be externally measurable; that is, there should be ways that allow the measuring entity to retrieve such information on the target entity over the network. The second challenge is the non-repudiation of measured data. Trusted hardware from a device, such as the trust platform module (TPM), may be used to store and communicate the measurement data in order to avoid malicious change of information when it flows through potentially malicious OSs.

One particular TPM-related mechanism is to have the hardware record the community ID for every community joined, and to maintain this as a list in sealed storage. An issuer can then query the TPM of the enrollee about whether it is already a member of the community into which the enrollee wants to enter, and the TPM will provide a zero-knowledge proof that the new community is not already on its list. The zero knowledge proof will fail if the enrollee has already joined the community, thus making Sybil and white-washing attacks more difficult.

Network activity correlation. This kind of attribute is transient; yet, it is useful for building correlations between entities in the network. For instance, tables used by the address resolution protocol (ARO) on hosts reveal recent IP and MAC address bindings in the network. Information from multiple nodes may be useful to build consensus on the correct usage of MAC addresses by members. Another way to get information would be to use routing tables. Routing table entries from multiple nodes in the network help to build topological relationships between devices in the network, that can sometimes be used, together with other localization techniques, to help distinguish unique devices.

Related Work

"Our work is inspired by Gligor's analysis."

Our work is inspired by Gligor's analysis [4]. He advocated that trust establishment is an emergent property in ad hoc networks, and trust relationships may need to be established among nodes after network emergence. Hence, trust establishment has to be based on dynamic evaluation of evidence about a node and not just on a statically defined relationship with a single third party. He also urged the design of evidence-evaluation metrics to assign low certainty to evidence from questionable sources while still achieving an acceptable number of false positives. We take this a step further and use identities to signify relationships and verify entity uniqueness to examine the evidence in question.

Previous work has also proposed several variants of the distributed trust model. Eschenauer et al. introduce the general principles of trust establishment in mobile ad hoc networks [3]. Many researchers assume the transitivity of trust to establish a relationship between two entities without the necessary prior interactions. The trust evaluation is modeled as a path problem in a directed trust graph. Theodorakopoulos and Baras [8] extend the PGP model to use second-hand evidence. However, their trust evaluation assumes independent opinion sources. Reiter and Subblebine advocate that trust calculation has to be based on multiple non-intersecting paths [6]. They propose algorithms to identify trust paths in the trust graph. Unlike our model, their work still assumes every entity offering opinions is distinct.

"Many researchers assume the transitivity of trust to establish a relationship between two entities without the necessary prior interactions."

Several papers [7, 17, 18, 19, 20, and 21] adopt the idea that trust can be established through direct observations or through third-party recommendations. Sun et al. represent trust as uncertainty, computed by using entropy [7]. Zouridaki et al. use modified Bayesian approaches to build trust and reputation systems by using second-hand information [21]. Jiang and Baras use weighted voting algorithms to deal with conflicting opinions [5, 17]. The model favors local interactions over second-hand opinions. Several works use Dempster-Shafer Theory (DST) for trust evaluation [19, 20] to take into account the uncertainty of evidence that cannot be evaluated by using Bayesian methods. Raya et al. [20] propose evaluating data-centric trust in vehicular ad hoc networks (VANETs). They use simulations to evaluate algorithms by using weighted voting, Bayesian methods, and DST, and they conclude that each method has its own strength in different networks; however, they hold that DST is best suited to the decision logic requirements in a time-critical vehicular network.

"Several researchers adopt the idea that trust can be established through direct observations or through third-party recommendations."

In addition to trust evaluation, there are a few works on trust evidence generation and distribution. Eschenauer et al. describe examples of generic evidence generation and distribution in a node-centric authentication process [3]. Hubaux et al. propose a model to build partial local certificate repositories for PGP [22]. Jiang and Baras propose an ant-based routing algorithm to search for trust evidence in ad hoc networks [23].

Conclusions and Future Research Directions

In this article, we present a paradigm of a distributed trust model generalizing beyond the enterprise model to ad hoc, mesh, and self-organizing networks, where every member can serve as an authority to enroll and authenticate devices for the community. Our model elevates the problems of on-line evidence evaluation and bootstrapping trust to first-class concerns and proposes solutions to address these problems. We focus on designing credentials to signify the trust relationships that emerge within a community and suggest a novel identity-laundering concept to establish new relationships from pre-existing trust relationships rooted in different administrative domains. We also extend the existing trust propagation models to incorporate both negative opinions and social relationships.

This work opens a new research area for trust management. A number of open problems remain. We plan to design a self-organizing information distribution system suitable for trust evidence dissemination in various network sizes and topologies. Another area for future work is identifying appropriate trust calculus and trust metrics for evaluating various first-hand trust evidence and computing initial device reputation.

References

[1] R. Housley, W. Polk, W. Ford, and D. Solo. "Internet X.509 Public Key Infrastructure Certificate and CRL Profile." *RFC 3280*, 2002. At *http://www.ietf.org*

[2] C. Rigney, W. Willens, A. Rubens, and W. Simpson. "Remote Authentication Dial In User Service (RADIUS)." *RFC 2865*, 2000. At *http://www.ietf.org*

[3] L. Eschenauer, V.D. Gligor, and J. Baras. "On Trust Establishment in Mobile Ad Hoc Networks." In *Proceedings of 10th International Security Protocols Workshop*, 2002.

[4] V. Gligor. "Security of Emergent Properties in Ad-Hoc Networks." In *Proceedings of the 4th ACM Workshop on Wireless Security*, 2005.

[5] T. Jiang and J. Baras. "Trust Evaluation in Anarchy: A Case Study on Autonomous Networks." *INFOCOM 2006*, 2006.

[6] M. Reiter and S. Stubblebine. "Resilient Authentication Using Path Independence." *IEEE Transactions on Computers*, Volume 47, No. 12, December 1998.

[7] Y. Sun, W. Yu, Z. Han, and K.J. Ray Liu. "Information Theoretic Framework of Trust Modeling and Evaluation for Ad Hoc Networks." *IEEE Journal on Selected Areas in Communications*, 24(2):305—317, February 2006.

[8] G. Theodorakopoulos and J. Bara. "On Trust Models and Trust Evaluation Metrics for Ad Hoc Networks." *IEEE Journal on Selected Areas in Communications*, 24(2);318—328, February 2006.

[9] P. R. Zimmermann. *The Official PGP User's Guide*. Cambridge, MA, MIT Press, 1995.

[10] Wi-Fi Protected Setup. At *http://www.wi-fi.org*.

[11] A.-L. Barabási. *Linked: How Everything Is Connected to Everything Else and What It Means*. New York, New York: Penguin Books. 2003.

[12] A.-L. Barabási and R. Albert. "Emergence of scaling in random networks." *Science 286*, 509-512, 1999.

[13] D. Faria and D. Cheriton. "Detecting Identity-Based Attacks in Wireless Networks Using Signalprints." *ACM Workshop on Wireless Security* (WiSe'06), September 2006.

[14] Y. Chen, W. Trappe, and R. Martin. "Detecting and Localizing Wireless Spoofing Attacks." *IEEE SECON 2007*, September 2007.

[15] Y. Sheng, K. Tan, G. Chen, D. Kotz, and A. Campbell. "Detecting 802.11 MAC Layer Spoofing Using Received Signal Strength." *INFOCOM 2008*, April 2008.

[16] L. Xiao, L. Greenstein, N. Mandayam, and W. Trappe. "Fingerprints in Ether: Using the Physical Layer for Wireless Authentication." In *ICC Proceedings*, 2006.

[17] T. Jiang and J. Baras. "Autonomous Trust Establishment." In Proceedings of the 2nd International Network Optimization Conference, 2005.

[18] T. Jiang and J. Baras. "Trust Evaluation in Anarchy: A Case Study on Autonomous Networks." *INFOCOM 2006*, 2006.

[19] A. Josang. "An Algebra for Assessing Trust in Certification Chains." In *Proceedings of NDSS'99*, 1999.

[20] M. Raya, P. Papadimitratos, V. Gligor, J. Hubaux. "On Data-Centric Trust Establishment in Ephemeral Ad Hoc Networks." *INFOCOM 2008*, April 2008.

[21] C. Zouridaki, B. Mark, M. Hejmo, and R. Thomas. "Robust Cooperative Trust Establishment for MANETs." In *Proceedings of SASN'06*, 2006.

[22] J.-P. Hubaux, L. Buttyan, and S. Capkun. "The quest for security in mobile ad hoc networks." In *Proceedings of MobilHoc'01*, 2001.

[23] T. Jiang and J. Baras. "Ant-based Adaptive Trust Evidence Distribution in MANET." In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops* (ICDCSW'04), 2004.

Author Biographies

Meiyuan Zhao is a Research Scientist at Intel Labs working on improving the security and usability of the Intel next-generation platforms. Meiyuan received her Ph.D. degree from Dartmouth College. Her research interests include network security, trust and reputation systems, swarm intelligence, peer-to-peer networks, routing protocols, and distributed systems. Her e-mail is meiyuan.zhao at intel.com.

Hong Li is a Principal Engineer at Intel Labs. Hong joined Intel in 1999 as a Security Architect and led the development of IT security strategies and architectures. She was also a lead researcher on several IT security research initiatives including policy-enabled network security, trusted autonomics, and community-based trust. Hong is active in the industry and academia with many external publications and pending patents in the area of security and manageability. Hong holds a Ph.D. degree from Penn State University and a B.S. degree from Xi'an Jiaotong University, China, both in Electrical Engineering. Her e-mail is hong.c.li at intel.com.

Rita Wouhaybi is a Research Scientist with the Emerging Platforms Lab (in STL/CTG) at Intel Corporation. Rita received her Ph.D. degree in Electrical Engineering from Columbia University in 2006. She also holds BE and ME degrees in Computer and Communications Engineering from the American University of Beirut. She interned with HP Labs working on network measurements and parameters estimation. She also had an internship with Intel IT Research working on intelligent overlays and their applicability to the enterprise. Her research interests include peer-to-peer networks, game theory, the use of artificial intelligence in networking, and social networks. Her e-mail is rita.h.wouhaybi at intel.com.

Jesse Walker is an Applied Cryptographer at Intel Labs. He was the person to first identify vulnerabilities in the 802.11 WEP protocol. He also served as editor for the 802.11i standard. He joined Intel in the Shiva acquisition. He has a Ph.D. degree in Mathematics from the University of Texas. His e-mail is jesse.walker at intel.com.

Vic Lortz is a Senior Architect at Intel Labs. Since joining Intel in 1994, Vic has worked on several projects related to home networking, wireless networking, and network security. Vic served as chair of the security working committee in the UPnP Forum. He was also a Lead Architect and Editor of the Wi-Fi Protected Setup specification. His recent work involves peer-to-peer wireless discovery, setup, and optimizations to enable new mobile device usages. Vic holds M.S. and Ph.D. degrees in Computer Science from the University of Michigan. His e-mail is victor.lotz at intel.com.

Michael J. Covington was, until recently a Senior Research Scientist working at Intel Labs. His research focused on improving security and reliability for Intel's next-generation platforms. With more than six patents pending and as the author of numerous papers that have been published in leading academic conferences and journals, Dr. Covington's research has explored formal access control modeling, cutting-edge authentication techniques, and security approaches for pervasive computing environments. Dr. Covington received his Ph.D. and MSCS degrees from the Georgia Institute of Technology's College of Computing in Atlanta, Georgia. He also holds a B.S. degree from Mount Saint Mary's College in Emmitsburg, Maryland. His e-mail is research at MichaelCovington.com.

Copyright

Intel Technology Journal

Peer Reviewers
Ernie Brickell
Howard Herbert
Shay Gueron
Eric Mann