Hardware-rooted Trust for Secure Key Management and Transient Trust

Jeffrey S. Dwoskin and Ruby B. Lee
Department of Electrical Engineering
Princeton University
Princeton, NJ 08544, USA
jdwoskin@princeton.edu, rblee@princeton.edu *

ABSTRACT

We propose minimalist new hardware additions to a microprocessor chip that protect cryptographic keys in portable computing devices which are used in the field but owned by a central authority. Our *authority-mode architecture* has trust rooted in two critical secrets: a Device Root Key and a Storage Root Hash, initialized in the device by the trusted authority. Our architecture protects trusted software, bound to the device, which can use the root secrets to protect other sensitive information for many different usage scenarios. We describe a detailed usage scenario for crisis response, where first responders are given transient access to third-party sensitive information which can be securely accessed during a crisis and reliably revoked after the crisis is over.

We leverage the Concealed Execution Mode of our earlier *user-mode SP* (Secret-Protecting) architecture to protect trusted code and its execution [1]. We call our new architecture *authority-mode SP* since it shares the same architectural lineage and the goal of minimalist hardware roots of trust. However, we completely change the key management hardware and software to enable new remote trust mechanisms that user-mode SP cannot support. In our new architecture, trust is built on top of the shared root key which binds together the secrets, policy and trusted software on the device. As a result, the authority-mode SP architecture can be used to provide significant new functionality including *transient access* to secrets with reliable revocation mechanisms, controlled transitive support for *policy-controlled secrets* belonging to different organizations, and *remote attestation and secure communications* with the authority.

Categories and Subject Descriptors: K.6.5 [Management of Computing and Information Systems]: Security and Protection; C.1.0 [Processor Architectures]: General;

General Terms: Security

*This work was funded by NSF Cybertrust and DARPA under grants CNS-0430487 and CNS-0636808 for the SecureCore collaborative research project.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CCS'07, October 29–November 2, 2007, Alexandria, Virginia, USA.
Copyright 2007 ACM 978-1-59593-703-2/07/0010 ...$5.00.

Keywords: Secure Processors, Key Management, Hardware Policy Enforcement, Secret Protection (SP), Transient Trust, Emergency Response.

1. INTRODUCTION

We present a model for trust in portable computing devices, where a central authority owns many devices used remotely. It wants to share secrets and sensitive data with users who are given the devices, but must maintain control over how and when these secrets and data are used. We define *transient trust* as the ability to access protected information for a limited time under certain conditions. We protect information by encryption and hashing, and hence reduce the problem of transient trust to that of secure key management. In addition to secrets it owns, the authority also wants to provide access to third party secrets on its devices. We define new *authority-mode architecture* to support such trust in remote devices by adding a few fundamental security features to commodity processors at very low cost.

While this trust model applies to many usage scenarios, we use crisis response as a concrete motivating example. First responders are provided transient access to sensitive information while in the field. Whether it is a natural disaster, a terrorist attack, a building fire, or a medical emergency, first responders need immediate access to sensitive information stored in electronic databases maintained by a central trusted authority or by other third-party data providers. This might include data about building occupants, medical records, floor-plans, building or city evacuation plans, satellite maps, and other types of information. This sensitive information must be protected at all times, and access should be limited in scope and duration.

The fundamental hardware features we propose are used to protect critical secrets (e.g., a master cryptographic key and a root hash of a secure storage structure) and to provide a secure execution environment for the software that operates on those critical secrets. These hardware features can be used to support software architecture, protocols and storage that provide important new security functionality. Hence, the processor itself provides hardware-rooted trust for flexible software architecture and usage models. We leverage the secure execution environment from our previous work on the SP (Secret-Protecting) architecture [1] to protect the intermediate data generated during the execution of trusted software. But the similarity with the original SP ends there. Our authority-mode architecture supports remote trust usage models with secrets and devices owned by the authority; this cannot be done by the original SP ar-

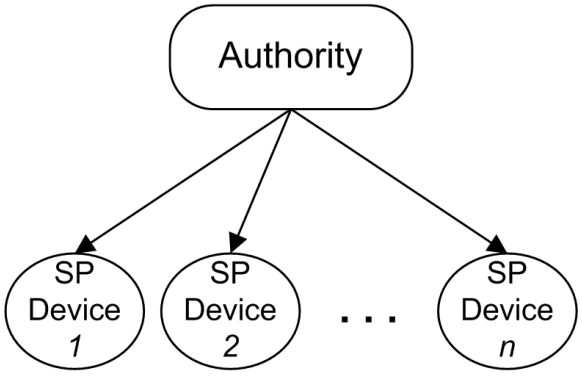

Figure 1: Trust Model of Authority and Many SP Devices

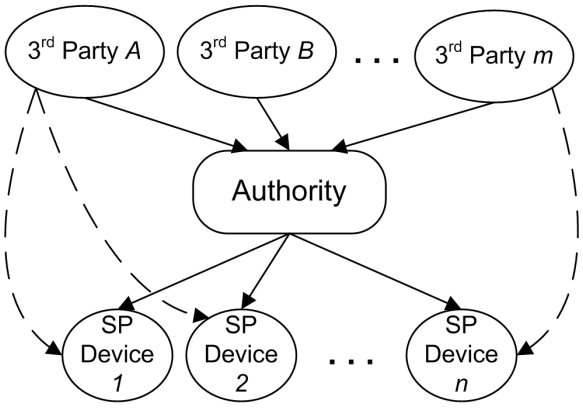

Figure 2: Trust Model Expanded to Include Third Parties

chitecture which only provides local trust of a user's secrets on his own device. We therefore refer to the original work as *user-mode SP* as compared to *authority-mode SP* for the new architecture described in this paper.

A primary contribution of this paper is the demonstration of the minimal hardware roots of trust needed for important and complex usage models involving cryptographic access control to protected information. We show that only two hardware registers are needed: a Device Root Key and a Storage Root Hash for the key chain and associated policies. We show how this can be used by a trusted authority: to establish a remote trust relationship with a device it owns that is used in the field *(remote trust)*; to support owner-controlled policy enforcement for the use of secrets on the device *(policy-controlled secrets)*; to provide transient access to secrets *(transient trust)*; and to securely delegate third party secrets to the device user *(controlled transitive trust)*.

The rest of the paper is organized as follows: Section 2 describes our new authority-mode SP trust model, our assumptions and threat model. Section 3 describes the new hardware features and software mechanisms in our architecture. Section 4 presents a detailed crisis response scenario that demonstrates how our architecture can be used. Section 5 provides a security analysis of authority-mode features and architecture. Section 6 discusses the performance and cost of the architecture. Section 7 discusses related past work and future work. Section 8 presents our conclusions.

2. TRUST MODEL AND THREAT MODEL

2.1 Trust Models

Our basic trust model is shown in Figure 1, where an authority is an entity that owns many SP devices used in the field. It establishes a trust relationship with each device and can delegate trust to the device itself, depicted as arrows in the figure. The authority can distribute secrets (i.e., keys in this paper) to the devices for remote use. It specifies access control by sending policies associated with the keys, which the devices will enforce. Through this model, we explore three concepts for authority-owned devices.

First *remote trust*, where authority-owned secrets are provided to a remote device with assurance they will be protected using the authority's own trusted software which it had earlier installed in the device. The trust relationship is based on a shared secret, binding together the authority's secrets with its trusted software and allowing the device to remotely attest to the authority that it is operating correctly. Second, *policy-controlled secrets* are the binding of secrets with access control policy for remote enforcement, and their protection from illegitimate modification or separation from each other when stored on the device. We ensure that secrets are only used in authorized ways and only through controlled interfaces. Third is *transient trust*, allowing the authority to provide secrets to remote users on a temporary basis, with reliable revocation mechanisms. The authority might set limits on use, change policies, or remove secrets entirely. Such policy enforcement and changes must be guaranteed.

We will also explore a fourth concept, *controlled transitive trust*, expanding our trust model to include third party secrets and sensitive information. In Figure 2, we show a number of third parties who each have a trust relationship established with the authority. The authority and each third party first agree on security policies to be used by the authority's devices to access third-party confidential data. The authority installs these policies in the devices, which can be trusted to enforce them. The authority thus enables transitive trust from the third parties to individual devices, by establishing policy-controlled temporary relationships between them, depicted as arrows with dashed lines. During operation, the devices can communicate directly with the third parties and access secrets and data for which they will also enforce the associated policies.

2.2 Assumptions and Threat Model

Our threat model covers operational threats and not developmental threats. Hence, we assume that the devices are manufactured correctly in a trusted factory and that the hardware is free from defects. The authority will receive the devices from the factory with all processor and system features intact and unmodified. Similarly, while we do not assume the regular system software is correct, we assume that the authority's trusted software is carefully designed and well-tested to ensure that it is correct and has no software security vulnerabilities. Furthermore, we assume strong cryptography for our encryption and hashing, which is computationally infeasible to break.

We assume that the processor chip is the physical security boundary for the hardware. If the adversary has physical

possession of the device, he can probe buses on the board but cannot probe inside the processor chip without rendering it unusable. A microprocessor chip is fabricated with many physical layers and processing steps. Because of its complexity, probing of this chip is extremely likely to destroy circuitry unless very expensive equipment is used. Hence, any registers or cache memory on-chip are assumed to be safe from physical threats of observation or modification.

We only consider the design of the authority-mode SP client device in this paper; we assume the trusted authority has secure systems where it can store its secrets and run its own software with perfect secrecy and access control. Similarly, we assume the authority has a secure depot at which it can initialize new SP devices. Also, while we protect the confidentiality and integrity of the secrets and sensitive data, we do not defend against Denial of Service attacks.

Users are trusted to protect their authentication tokens (e.g. passwords) when logging in to an SP device. They are expected to log out promptly when done with the session. The adversary has no authorized access and will not be able to login to the device.

The threats for authority-mode remote trust models are quite different than those for a local trust model like in user-mode SP [1], where the user is also the owner of the device and has physical possession of the device. The devices and secrets are authority-owned, so while the local users are intentionally given access to secrets, the users can also potentially be adversaries. Even legitimate users must not be allowed to exceed their authorization and access to secrets, or to use secrets in unauthorized ways.

The main threats we consider are violations of the confidentiality and integrity of keys that the authority supplies to the user of the authority-mode SP device. We consider both software and physical attacks. Attackers can launch software attacks on code and data, and network attacks by observing or modifying traffic on the public networks. Code and data on disk and in memory, in transit across the network or buses, as well as the processor state at interrupts, are all vulnerable to spoofing, splicing, and replay attacks. Spoofing introduces false data and modifications; splicing rearranges real data; and replay reintroduces previously used data that had been modified, deleted or revoked.

We also consider physical threats since an SP device can be lost or stolen. An attacker may have temporary physical access to the device, returning it without evidence of tampering, or "permanent" access on obtaining a lost or stolen device. Thus, this adversary can mount physical attacks on both the hardware and software. Any code or data stored on the hard disk or in main memory, belonging to the operating system or an application, is fully accessible to the adversary. By modifying the operating system, he can access and manipulate the full state of an application, including the processor registers during interrupt handling. Main memory can also be accessed by physically probing the memory bus, or by performing rogue DMA operations. We note that the Trusted Platform Module (TPM) [4] does not protect against physical attacks.

3. ARCHITECTURE

3.1 Hardware-Rooted Trust

A major contribution of this paper is to show that very little hardware support is needed to enhance secure key management and enable transient trust. Only two new processor registers containing hardware roots of trust, and a secure BIOS for bootup are needed to support this trust model. Figure 3 shows the architecture of a processor chip with these components added in bold and darkly shaded. We use a 128-bit non-volatile register to store the Device Root Key (DRK), and a 256-bit non-volatile register to store the Storage Root Hash (SRH). There is also a 1-bit DRK Lock flag which prevents software from writing to the DRK. We have a secure BIOS to allow initialization of the DRK and to lock it before loading the regular BIOS or other software.

Table 1: New Instructions

Instruction	Description
New Authority Mode SP Instructions	
GR_TO_DRK	Sets Device Root Key register from GRs.
DRK_LOCK	Sets DRK_Lock register to 1, disabling GR_TO_DRK instruction.
GR_TO_SRH	Sets Storage Root Hash register from GRs.
SRH_TO_GR	Reads the Storage Root Hash register into GRs.
DRK_DERIVE	Derives a key from DRK by computing a keyed hash over a nonce and constants.
Instructions for providing a Concealed Execution Mode for TSM code	
BEGIN_CEM	Enter CEM for next instruction.
END_CEM	End CEM for next instruction.
SECURE_STORE	Secure store from GR to memory.
SECURE_LOAD	Secure load to GR from memory.

These two registers may only be accessed by a Trusted Software Module (TSM), whose execution is protected by the hardware by a Concealed Execution Mode (CEM). CEM prevents leaking of intermediate values during TSM execution as described below.

In Figure 3, the white components represent a typical unmodified processor, and the lightly shaded components show the features added to provide the concealed execution environment for trusted software modules. New instructions to access the new registers, derive keys from the DRK, and provide the Concealed Execution Mode are listed in Table 1.

3.2 Trusted Software Modules

In our architecture, a *Trusted Software Module (TSM)* is the only software able to directly access the DRK and SRH root secrets stored on the device. This high assurance, trusted software module is provided by the trusted authority who initializes the DRK and SRH in the device at its depot. The authority trusts its TSM to use the secrets correctly and to maintain confidentiality.

The DRK is used to *sign the TSM* by inserting a keyed-hash into each cache line of code, upon installation of this trusted code on the device. Later, the code is dynamically verified for integrity during TSM execution (*Code Integrity Checking*), which we describe in detail below.

Both the trusted software and the authority's secrets will be bound to the DRK, and consequently to each other. Changing the TSM on the device, by anyone other than the authority, requires replacing the DRK, since knowledge of the DRK is necessary to sign new code. This will simultaneously cut off all access to the secrets bound to the previous DRK. Therefore the secrets and the TSM that operates on them are bound together and to the device itself.

3.3 Secure Execution Environment

To provide protection of the TSM, we leverage the secure execution environment introduced with the user-mode SP

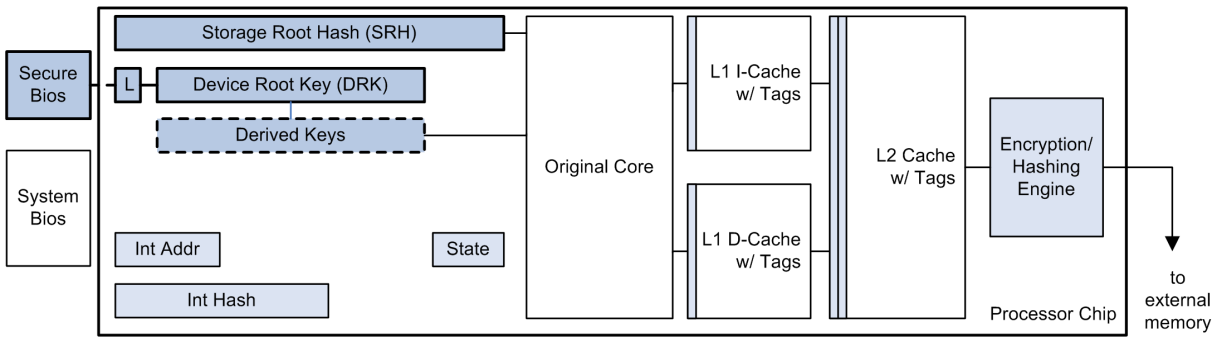

Figure 3: Hardware Features for Authority Mode SP Architecture (shaded)

architecture [1]. We summarize this as two parts: *Code Integrity Checking (CIC)* and *Concealed Execution Mode (CEM)*. While the hardware support provided for key management in the original user-mode SP architecture does not meet our needs, the hardware support it provides for a secure execution environment for trusted software is entirely appropriate, and we describe this briefly below. This section describes the lightly shaded areas in Figure 3, and the last four instructions in Table 1.

The combination of dynamic Code Integrity Checking of the TSM with the protection of intermediate data in registers, caches and memory, provides a secure execution environment for executing trusted software that does not leak the values of the DRK and SRH registers, nor the keys they protect.

The first component, Code Integrity Checking, ensures that the TSM code cannot be modified during storage, transmission and execution. It also ensures that no other code can be used as a TSM with access to authority secrets. TSM code is signed by computing a keyed cryptographic hash over each cache-line of code, keyed with the DRK, and embedding the hash into the code itself. For example, a 64-byte cache line would contain 48-bytes of TSM code, followed by a 16-byte hash (e.g., using AES-CBC-MAC as the hash algorithm [5]). When the trusted authority installs the TSM, it adds these hashes using the DRK. Without access to the DRK, no other party can sign TSM code.

During execution, the TSM code is verified as it is loaded into the on-chip Level-2 (L2) cache in the microprocessor chip. The keyed hash is recomputed over the cache line of instructions and compared to the stored value, causing any modifications to be detected dynamically during program execution — and not just upon program launch as in other schemes [4] [6]. If the hash check passes, the embedded hashes are replaced with no-op instructions so as not to affect execution. Within the processor chip, the hardware keeps track of TSM code using cache-line tags added to the L2 and L1 caches. Verified instruction cache-lines are tagged as "Secure Instructions" and are read-only. The tag is carried over from L2 cache to the L1 instruction-cache, and is checked when fetching TSM code for execution.

The second component is Concealed Execution Mode (CEM), which protects the confidentiality and integrity of data used by the TSM during execution. Intermediate data is stored in the general registers of the processor, in the caches and in main memory.

Intermediate data that could leak secret information is actively protected by the TSM by using special *Secure_Load* and *Secure_Store* instructions to read and write such data. In general, within the security perimeter of the microprocessor chip, secure data is tagged in on-chip data cache lines with a "Secure Data" tag. When such a data cache line has to be evicted from the on-chip L2 cache, it is first encrypted and hashed with the DRK. Similarly, the contents of a data cache line are only hash-verified and decrypted when it has to be brought on-chip into the L2 cache. This happens on only a few *Secure_Load* instructions which miss in both the L1 and L2 on-chip caches. Secure data in cache can only be read or written to by TSM code in CEM. Any accesses to "Secure Data" tagged cache lines with normal load and store instructions will cause them to be evicted (encrypted and hashed), and then reloaded in encrypted form.

The contents of general registers must also be protected during a processor interrupt. The executing CEM thread can be interrupted at any time for a hardware interrupt or software exception. The OS interrupt handler then saves the process's register state to memory before executing other code. When an interrupt occurs during CEM, the hardware protects the registers before turning control over to the OS interrupt handler. It encrypts the registers as a single plaintext chunk using the DRK, splits up the resulting ciphertext, and places it back in the registers. It computes a hash over the resulting ciphertext which is stored in the processor (in the *Int Hash* register shown in Figure 3), along with the memory address of the next CEM instruction (saved in the *Int Addr* register). These are checked automatically to resume CEM when the interrupt is completed. With the contents of the general registers thus protected, the hardware then allows the OS interrupt handler to assume control to save and restore the register state without being able to read the (plaintext) contents.

The state of the processor, whether executing in CEM or not, and whether a thread in CEM has been suspended, is indicated by the *State* bits in Figure 3. More details on CEM and the hardware support for a secure execution environment can be found in [1].

3.4 Secure Storage for Keys

Building off of the secure execution environment provided by CEM, the TSM in turn is responsible for protecting persistent secrets (e.g. the key-chain) from unauthorized access. For this protection, we can create *secure local storage*, a data structure that provides hierarchically encrypted and hashed storage of keys.

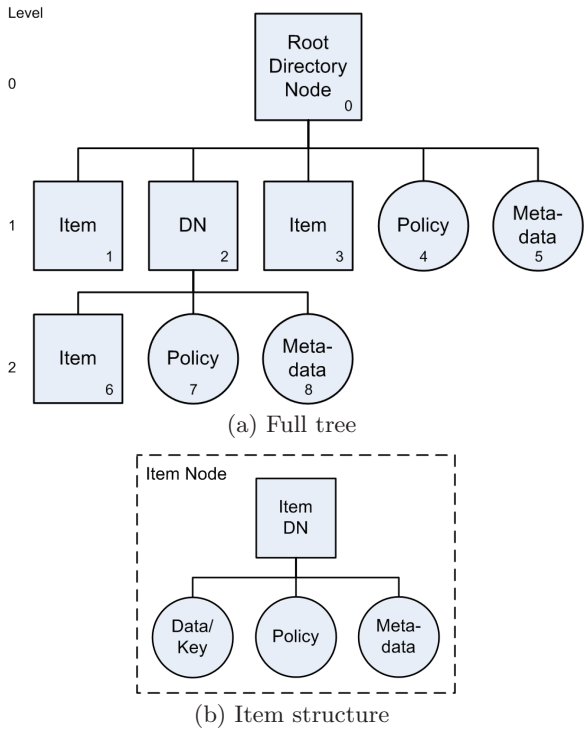

Figure 4: Secure local storage hierarchy

This secure storage structure incorporates a Merkle hash-tree mechanism [3], storing the root hash in the Storage Root Hash (SRH) register on-chip. The root hash, updated only by the TSM, ensures integrity of the keys against malicious modifications by untrusted software. Since the root hash is stored on-chip, the secure storage is also protected against replay attacks — changes to the secure storage structure are made permanent, including deletions. Stale (deleted) data cannot be replayed since the root hash will no longer match.

Replay-resistent secure storage can provide *transient trust* for the authority's keychain. At the authority's request, the TSM can permanently revoke access to certain keys by deleting them from the secure storage. The secure storage also contains nodes for access control policies associated with keys, to be enforced by the TSM. This provides *policy-controlled secrets*, which the authority can use to cut off access when a predetermined condition is met. This provides another means of revocation that is effective even when the authority cannot communicate with the remote device.

The basic tree structure of the secure local storage is shown in Figure 4(a). All non-leaf nodes are *directory nodes (DNs)* and store special meta-data about their child nodes. Leaf nodes can store not only keys, but also data, additional meta-data, and access control policies. A typical item is shown in Figure 4(b) and is actually a collection of nodes: a directory node and nodes which then contain keys or data along with their respective policies and meta-data.

The secure storage structure is built on top of insecure OS storage facilities and must be protected from attacks while on disk or in main memory. Each node in the structure is encrypted with a derived key, generated from the DRK using a nonce stored in its parent DN. A different derived key, using the same nonce, is used to generate a keyed-hash (MAC) of the contents of the node. The resulting Merkle hash-tree is incorporated into the secure storage structure by storing the MAC in the DN. When data is first added, a random nonce is chosen; it is then saved to regenerate the same derived key for later decryption and integrity verification.

ID	Type	Nonce	MAC	Size	Location
001	1	813547	0110...1101	6144	nodes/item1.dat
002	1	718232	0001...0101	9216	nodes/dn2.dat
003	1	128203	0100...1100	6144	nodes/item3.dat
004	2	519025	1010...0011	27648	nodes/policy4.dat
005	3	251092	0101...0001	13824	nodes/meta5.dat

Figure 5: Directory Node Structure

The DN structure, in Figure 5, also contains meta-data identifying the nodes, their type, size and location on disk. A chain of IDs can trace a path through the tree to reach a particular node, such as "0:2:7" for node 7 in Figure 4(a). When the TSM reads a DN, it can use the type field to quickly identify any policy nodes that apply to a key or data node without decrypting each node. Once found, policies are enforced by the TSM and are inherited hierarchically from parent to child; conflicting policies are enforced by allowing policy nodes lower in the tree to override general policy set by a parent or ancestral node. Any descriptive or application-specific meta-data (e.g. cipher type, creation date) can be stored in a separate meta-data node.

Data nodes and key nodes are distinguished with different types (not shown in Figure 5) so that specific policy can be set to control keys, preventing them from being leaked as data. The policy node can contain an Access Control List (ACL), indicating which users are allowed access to the data or key, and under what conditions. The policy may include generic read/write permissions, limits on the number of accesses, expiration dates, or type-specific access control, such as which queries can be made to a database. Policy nodes can also contain general configuration for the authority's software, such as user customizable settings. The format for policy files is determined by the authority's software and can be implementation specific. Secrets used for user authentication are similarly stored in the tree.

Figure 6 shows a simple storage tree with only one level of items. At the root of the tree is a special DN, the Root Directory Node (RDN). Unlike all other DNs, the RDN does not have a parent to store its directory entry. Instead, the TSM is hard-coded with a location to find its nonce and size (both unencrypted), and the DRK-encrypted contents of the node. The root MAC is stored in the on-chip SRH register, where it is protected from attacks.

3.5 Derived Keys

Secrecy of the secure storage is rooted in the DRK. Since the DRK is a critical secret used to maintain the trust relationship with the authority, it must never be revealed to any software on the device. Instead the TSM derives new keys from the DRK, using a new processor instruction, *DRK_derive* (see Table 1). This instruction performs a cryptographic hash function to combine the DRK with a value or nonce provided by the TSM. These derived keys are used to encrypt the nodes of the secure storage structure. As a result, the keys in secure storage are bound to the device and cannot be read (decrypted) or modified (without detection) by non-TSM software.

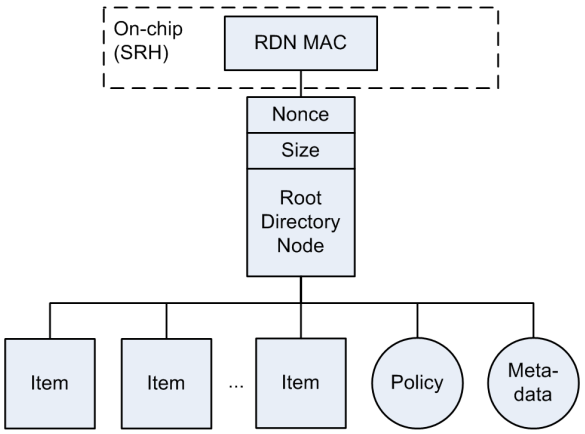

Figure 6: Simple storage hierarchy with Root Directory Node

For more general purpose use of derived keys, the value provided to the hardware can include a constant that will distinguish keys used for different purposes, e.g., communications versus storage. It also includes one or more nonces to make each key unique. When the software needs to regenerate a particular key, such as an encryption key for secure storage, the nonce can be saved with the data and reused by the TSM to obtain the derived key. Similarly, if the authority and the device need to produce the same key, they can exchange nonces and generate the same derived key — the authority deriving the key from its copy of the DRK in a secure environment. On the device, the TSM is responsible for protecting the secrecy of the derived keys that it produces, however the constants are not secret.

As an example, Table 2 shows that derived keys for storage will be derived using a constant, $C_{Storage}$, and a different nonce for each storage node. An additional set of constants, C_{Enc} and C_{MAC}, distinguish keys used for encryption and hashing of storage nodes. Derived keys that will be used as session keys for secure communications and remote attestation can be derived using a different initial constant, C_{Comm}, along with nonces that ensure freshness from the Authority, N_A, and the device, N_D. Additional constants, C_A and C_D, provide different keys for each direction of communication.

Table 2: Derived Keys

Storage key for encryption of node i: $K_{Si_{Enc}} = MAC_{DRK}[C_{Storage}, C_{Enc}, N_{Si}]$
Storage key for hash of node i: $K_{Si_{MAC}} = MAC_{DRK}[C_{Storage}, C_{MAC}, N_{Si}]$
Communication key for Authority to Device: $K_{A\rightarrow D} = MAC_{DRK}[C_{Comm}, C_A, N_A, N_D]$
Communication key for Device to Authority: $K_{D\rightarrow A} = MAC_{DRK}[C_{Comm}, C_D, N_D, N_A]$

3.6 Remote Attestation and Secure Communications

For an authority-owned device, remote attestation is the process of proving to the authority that the device still has possession of the correct DRK and is still running the correct TSM software, which properly uses and protects its keys. Remote attestation is necessary when establishing communication channels between the authority and its device in the field. Note that this is a different and more lightweight "attestation" than that used in TPM-based systems, where remote attestation means verifying the integrity (cumulative hashes) of *every* level of software running on the machine, up through the desired application level.

A (Authority), D (Device)
Initiate communications, generate nonces & send challenges:
D → A: ID_D ; send device's ID to initiate communications
A → D: N_A ; random 128 bit nonce
D → A: N_D ; random 128 bit nonce
Generate derived keys for communication:
; A,D: $K_{A\rightarrow D} = MAC_{DRK}[C_{Comm}, C_A, N_A, N_D]$
; A,D: $K_{D\rightarrow A} = MAC_{DRK}[C_{Comm}, C_D, N_D, N_A]$
Send responses:
A → D: $Response_A = MAC_{K_{A\rightarrow D}}[N_A, N_D]$
D → A: $Response_D = MAC_{K_{D\rightarrow A}}[N_D, N_A]$
Verify responses:
; A: $MAC_{K_{D\rightarrow A}}[N_D, N_A] =?\ Response_D$
; D: $MAC_{K_{A\rightarrow D}}[N_A, N_D] =?\ Response_A$

Figure 7: Challenge-response protocol for generating session keys and attestation

To establish communications, the parties first need a secure channel over the public untrusted network. This is done by generating a session key which can encrypt and hash all messages. The session key will be a derived key based on the DRK, which is a secret shared only by the authority and this particular SP device. No other parties, except the authority and the device's TSM, can produce the correct session key. Consequently, using a correct derived key proves knowledge of the DRK, so setting up a working secure channel implicitly serves as remote attestation.

Since no other party knows the shared secret, attestation also serves as a form of mutual authentication, in the sense that the device and authority both know that the other has the same DRK. The authority knows it is communicating with a good TSM on a device that it can trust. It can send new secrets and policy, knowing they will be properly protected by its own signed TSM. It is also assured that any data or messages sent back from the device, through this secure communication channel, must have been generated by its TSM. Similarly, the device has authenticated the authority, so it can incorporate new keys or policies sent over this secure channel.

Secure communications can be established between the device and the authority using the challenge-response protocol given in Figure 7. Either side can initiate the protocol, during which both sides select a nonce and transmit it in plaintext. The nonces are combined with the constant prefixes to generate the derived session keys, $K_{A\rightarrow D}$ and $K_{D\rightarrow A}$. Once these keys are established, the parties each send a Message Authentication Code (MAC)—a keyed hash—of the combined nonces, using their session key. Verifying these MACs confirms that no man-in-the-middle or replay attack has occurred, and that the authority and device share the same keys. This verification requires each side to generate both of the uni-directional keys (in the middle section of Figure 7), using one key to send messages, and another to decrypt and check received messages. These session keys can be used for secure communications, using a standard protocol such as TLS-PSK [2], which initiates the TLS protocol using preshared keys.

3.7 Device Initialization

The authority initially establishes trust in a device at its trusted depot, using physical access to install the DRK. This new secret is generated randomly by the authority for each device and is therefore independent of the manufacturer, any other device, and any past use of the device. The authority saves a copy of the DRK in its own database along with the device's ID. It then boots the device into the secure BIOS by setting a hardware jumper. The secure BIOS bypasses normal bootup and executes a verified initialization routine from its ROM, which executes entirely within the security perimeter of the processor chip, using only on-chip caches. The routine saves the new secret in the on-chip DRK register. The device is rebooted, removing the jumper, and from then on skips the initialization routine. Instead the secure BIOS executes only the *DRK_Lock* instruction before passing control to the regular system bios for a normal bootup.

Then the TSM must be signed with the new DRK. This can be done on the authority's secure computers, where the verified TSM code resides. The signing process computes hashes over each cache line using the DRK, and stores them in empty spaces left by the compiler. The signed TSM, and other system code, can be copied to the disk or flash storage on the device.

Finally, the authority initializes the secure storage structures; for this paper, this stores the keys and their associated usage policies and meta-data. The authority must also set up user authentication and authorization data, storing user passwords, biometric data, or users' cryptographic keys and specifying what privileges each user has. All of these policies and secrets can be updated in the field via secure communications.

4. USAGE SCENARIOS

Our new remote trust model is applicable to many different scenarios, where an authority entity wants to extend trust to remote devices it owns which are used in the field. The authority could be a crisis response entity with devices used by first responders, firemen, policeman, etc. The authority could also be a military organization with remote devices used by soldiers in the field. The authority could be a bank with devices used by customers as personal ATMs for dispensing electronic cash and accepting deposits into bank accounts. The authority could even be a cell phone network provider allowing users to download software to their phones while having trusted network-access software. Below, we provide a detailed usage scenario for crisis response.

For crisis response, we provide transient acess to keys which enable access to protected (encrypted) data when the crisis starts, and revoke access to those keys when the crisis ends. At all times, we provide confidentiality of these keys. Even if a device is lost or stolen, the confidentiality of the keys, and hence the sensitive data they protect, must be maintained.

First responders can use their device (e.g., secure PDA) for both critical and non-critical applications. Non-critical applications include e-mail, web browsing, instant messaging, voice chat, etc. These do not use the TSM and may involve off-the-shelf or downloaded software that must not put at risk the sensitive data made available to crisis responders. For critical applications, sensitive data will be needed, such as firefighters accessing building plans and occupant information, or paramedics accessing medical records when they get to the scene of an accident. For these situations, the authority will distribute secrets and access policies in advance of a major crisis.

The sensitive information on the device will only be available to the responder through the controlled interfaces of the TSM after authenticating. For example, this might mean that the paramedic can read a patient's allergies and medical history but not access the psychiatric portion of his medical record. Rate limiting might also be employed, so a paramedic can access a few, but not hundreds of records at any one time.

4.1 Preparation

The authority makes access control decisions in advance for many likely crisis scenarios so the response can be quick when a crisis does occur. For example, to give the firefighter access to floor plans and building occupants, data for all buildings in a city should be prepared in advance. When a crisis occurs, the city's crisis response authority can decide which data to make available based on which parts of the city are affected. The city authority will negotiate access rights with building owners and hospitals in advance and decide how to delegate rights to individual responders using devices it will distribute. It forms trust relationships with each of these third parties in advance, setting up a certificate authority (e.g. using X.509 certificates [8]) so it can specify access rights and allow devices and third parties to authenticate each other during the crisis.

When a crisis begins, the authority will create and sign certificates for each device, tagged with a crisis ID and a reasonable expiration. It then communicates with each device, which performs remote attestation that it still has the correct DRK and DRK-signed TSM. It then distributes the necessary keys, along with the certificates for third party data and their associated access policies.

4.2 Crisis Operation

During the crisis, responders can contact third party data sources directly, via their devices, to retrieve sensitive data which they are pre-authorized for by the authority's certificates. Since the certificates will be stored securely on the device by the authority and signed by it, this is sufficient to authenticate the device to a third party. However for more complicated scenarios, the authority can give each device its own public/private key pair which it can use to authenticate directly to the third party.

Devices can also contact the authority during a crisis to obtain additional keys and access rights while in the field. Unanticipated circumstances will surely occur, requiring access to data beyond what was pre-authorized. For example, "need-to-know" may be determined in the field, for responders to have access to additional data. Alternatively, some additional data can be sent in advance for offline enforcement, to be revealed by the TSM only if needed.

4.3 Revocation post-crisis

Access to sensitive information must be revoked after a crisis has ended. This is accomplished through a combination of three revocation mechanisms. First, all secrets given to the device are policy-controlled and will include limits on use, including a maximum number of accesses and/or an expiration date. So as not to cut off access while the cri-

sis is still going on, these restrictions must necessarily give a wide safety margin. This first line of defense is effective even when the device is operating off-line.

Second, the authority can directly cut off access by contacting each device to delete secrets and modify the stored policies. The TSM will confirm that the revocation was successful and that the secure storage's hash-tree and SRH register were updated. The TSM can also report back what accesses had been made during the crisis. Knowing that accesses will be audited will provide a disincentive for abuse by authorized users.

Third, access to new secrets and data will be cut off, both from the authority and third parties. The authority will contact each third party and report that a particular crisis ID is no longer valid, or will revoke certain certificates. Any certificates that specified this ID will no longer be accepted for access, while general certificates for day-to-day use are still available. A new crisis ID can be used for the next crisis or another phase of the same crisis.

5. SECURITY ANALYSIS

5.1 Protection of Authority-mode Registers

The security provided by our architecture is rooted in trust in the new processor features. Since the processor chip is in our physical security perimeter, our new authority-mode registers can only be accessed through software instructions we define. We do not define any instructions to read the DRK register, and only allow writing by someone with physical possession of the device using the secure BIOS; therefore the DRK register contents can never be observed outside the processor or copied to another device and can never be modified except under secure BIOS execution. Similarly, only TSM software can use instructions for accessing the SRH register and for deriving keys from the DRK. Since the TSM is the authority's own correct, trusted software, no other software can ever observe or modify the SRH or generate derived session keys from the DRK.

5.2 Protection of TSM

The dynamic Code Integrity Checking for the TSM instructions ensures that it remains unmodified throughout its execution. This defends against dynamic hostile code insertion attacks, in addition to static changes to code on disk. Also, Concealed Execution Mode (CEM) ensures that secure intermediate data and general registers, during TSM execution, cannot be observed or modified without detection. If a modification is detected, the processor raises an exception; the TSM operation will be aborted and processor registers cleared. These mechanisms guarantee that a TSM will be protected during its execution, similar to user-mode SP [1].

However, our authority-mode architecture provides additional trust guarantees over user-mode. Since TSM code can only be hashed with the DRK, which is only known outside of hardware by the authority, no new TSM code can be added to a device by an adversary without changing the DRK. Thus, the authority can check that its unmodified TSM is running by having the device attest to having the same DRK value shared with the authority. If an adversary tries to modify TSM code or install new TSM code by replacing the DRK, the derived keys generated by the new TSM, based on this new DRK, would be different and the device will be unable to attest itself. Also any data encrypted by the authority's TSM will no longer be accessible. As a result, data written by any TSM is bound to that TSM. No other software is trusted with the authority's data or with policy enforcement, so integrity checking is only necessary for the TSM.

5.3 Protection of Persistent Secure Data

All data from the authority that starts out under control of the TSM will remain secure as long as the TSM handles it properly on the device. The authority has written and tested the TSM to ensure this property, making use of the persistent secure storage structure. In secure storage, confidentiality is protected with DRK-derived keys so that stored data can only be read by the TSM. Integrity is based on a hash tree rooted in the on-chip SRH register. That register can only be modified by the TSM, which is continuously installed as long as the DRK is unchanged. If the TSM were replaced, a new TSM could access the old value of the SRH, however, this value is meaningless without access to decrypt the data in the rest of the local secure storage structure. While the new TSM could change the SRH, the DRK can never be changed back to the original value, except by the authority. This new adversarial TSM will be useless, since the device can no longer attest the old DRK to the authority.

5.4 Remote Attestation and Communications

Remote trust also involves secure communication and attestation, which is based on the authority and device sharing the DRK secret. Remote device attestation (and mutual authentication) requires each side to prove knowledge of the DRK through the challenge-response protocol. Hence, our authority-mode protocol enables the authority to know that the device still has the shared DRK, and enables the device to know that it is speaking to the correct authority, i.e., one that knows the shared DRK. Random nonces prevent replay attacks on the exchange and are also used to generate session keys for communication. Man-in-the-middle attacks are not possible since no other party can generate the correct derived keys, to be used as valid session keys.

Once the authority knows its original DRK is still in the device and was used to correctly generate derived session keys, used for setting up the secure communications channel, it can be certain that only the TSM code that it originally signed with this DRK is running on the device and taking part in the communications. In fact, stronger than the integrity-checking done only on program launch by TPM-based systems [4], both user-mode and authority-mode SP devices provide *dynamic Code Integrity Checking* during the fetching of cache lines for the TSM throughout its execution. It is safe for the authority to transmit new data to the TSM, knowing it is continuously integrity-checked and protected by the DRK.

5.5 Policy-controlled Secrets

In our architecture, keys and policies cannot be separated without modifying or deleting nodes from the secure storage tree, which requires modifying other nodes, and ultimately modification of the root hash stored in the SRH. More generally, spoofing attacks on secure storage would include deliberate modification or insertion of data into the structure. The integrity checking will detect any such changes, and the MAC entries themselves cannot be spoofed without knowl-

edge of a DRK-derived key. If nodes could be rearranged, duplicated or removed, it would also affect the application of access control policies. Since each node or subtree is self-contained and independently hashed, parts of it can be reused at lower levels. However, the parent of each node is a DN that lists its children and their MACs in order. Any such splicing of the secure storage tree structure would be detected when the MAC of the parent is checked. Since the TSM performs checks all the way to the root SRH which is on-chip and safe from attack, it is guaranteed to detect any illegitimate changes. Therefore, the authority knows that an access policy stored on the device cannot be modified and will be enforced by the dynamically-verified TSM.

5.6 Transient Trust

Replay attacks on memory and disk storage are a threat to revocation, which is essential for transient trust. This is especially critical for data in the persistent secure storage tree. If the entire storage tree can be replayed, then any policy updates or revocation of keys or data performed by the authority could be undone. However, since the root hash is stored on-chip, out of reach of any adversaries, this is not possible; previous trees will not be valid. The only way to add or remove data from the tree is through the TSM, but the TSM will only make legitimate modifications specified by the authority during secure communications or through previously set policy.

5.7 Transitive Trust

This is based on the TSM's use of secrets and the authority's trust relationships established with the device and the third parties. These components are all individually assured, so the TSM's implementation of transitive trust will be maintained.

5.8 Other Issues

In addition to the TSM, we expect that other trustworthy system software (e.g., a security kernel and secure I/O drivers) will be present to assist with user authentication and secure display of data to the user. These are orthogonal issues to our design and therefore not discussed in this paper. Without trusting these mechanisms, the authority's TSM cannot guarantee that data displayed to the user will not be copied by some software. Therefore we place some responsibility on the user — a reasonable tradeoff since users can always reveal the data once it is outside of the device. The user should not intentionally login and access data when he knows the system to be physically compromised, and should not replace the system software and continue to access the TSM. The TSM and policies should set up controlled interfaces to release data to the user in as safe a way as possible, mitigating any risk from data leaked while being displayed. Only data being displayed is vulnerable in this way. Other secrets accessed by the TSM or stored securely on the device will not be leaked.

Our threat model currently does not defend against denial of service attacks, which prevents a guarantee of availability. Attacks on the TSM code, TSM execution, communications protocol, secure storage structure, etc, will all be detectable, but currently we have not defined recovery mechanisms. This is acceptable under our threat model, but less acceptable for a device relied upon in a crisis. While our architecture does not require trust in system software to provide remote attestation or protect the authority's secrets, the best defense for availability is to install a secure OS that can protect itself from untrusted applications and remote exploits, and for the user to maintain physical control of the device.

Our architecture uses strong ciphers and 128-bit symmetric keys and random nonces, which should be adequate. Implementations could easily be changed to increase these sizes or use different algorithms if necessary. By using different constants depending on the protocol used, derived keys are always made specific to their purpose, helping to prevent attacks from key re-use in different protocols.

6. COST AND PERFORMANCE ANALYSIS

6.1 Cost

The authority-mode architecture is implemented in hardware with only four new registers (totaling 576 bits) plus 2 state bits and 1 lock bit (see Figure 3). The four registers are the new 256-bit SRH register and 128-bit DRK register of authority-mode, and the 64-bit Int Addr and 128-bit Int Hash registers to provide the Concealed Execution Mode common to both user-mode SP and authority-mode SP. For on-chip caches, an insignificant 1-bit cache tag per L1 cache line and 2-bit cache tag per L2 cache line is added. An encryption/hashing engine is added at the chip boundary. Nine new instructions are defined, and a secure BIOS is required. This represents a tiny and insignificant cost for any commodity microprocessor or SOC (System-on-Chip), and even for many embedded processors.

6.2 Performance

Non-TSM software is basically unaffected by the new hardware since it will never trigger CEM protection or access the new registers or instructions. Furthermore, the security of authority-mode TSMs and secrets does not depend on limiting other software or verifying the entire software stack. Therefore, there is a negligible performance impact for system software and applications that do not use the TSM.

TSM software will incur a slight performance penalty (see [1]), but this penalty only impacts the small module where security is critical and a slight delay is acceptable. One potential source of delay is for Code Integrity Checking (CIC) of TSM code and Concealed Execution Mode (CEM) checking for secure loads and stores. These both occur only at the cache-memory boundary upon the rare L2 cache miss, where the miss penalty is already several hundred cycles for typical microprocessors. Hence, some tens of extra cycles for hardware hash computation and symmetric-key decryption or encryption is not going to cause much performance degradation. Once inside the L2 and L1 caches, accessing secure instructions and data proceeds as fast as before, since the checking of the 1- or 2-bit tags for a cache line (that has already been verified) is very fast.

CIC checking inserts additional no-op instructions into the instruction stream for each cache line. These will cause some degradation in the efficiency of the instruction-fetch process (since a fraction of instructions fetched are useless), but in a modern out-of-order processor, this should have insignificant effect on execution throughput. CEM interrupt protection requires the encryption and hashing of the general registers, but interrupts are relatively infrequent compared to the other factors.

The final component affecting performance is the design of the TSM software itself. Rather than directly accessing unprotected secrets, the TSM will perform additional cryptographic operations to retrieve secrets. For example, navigating the storage tree requires traversing nodes in the tree, each of which requires generating two derived keys in hardware which are used to check hashes and decrypt the node with software operations. Additional data may have to be retrieved from disk, involving the OS to access the file system, therefore causing additional system calls and memory/disk accesses, and possibly affecting cache behavior. We have designed the secure storage tree navigation to predominantly use symmetric-key operations which are significantly faster than asymmetric-key operations. Therefore we expect the effect of these additional operations to be small while providing significant additional security. Furthermore, the storage integrity checking requires Merkle tree hash operations only on the persistent secure storage tree, not on the whole memory. This results in very significant reduction in performance overhead.

7. RELATED WORK

7.1 Past Work

We adopt the CIC and CEM architectural features of the user-mode SP architecture [1], however the user-centric local trust model and the master secrets in SP could not serve our needs for remote trust, secrets bound to policies, transient trust guarantees, and controlled transitive trust. Our new authority-mode SP architecture requires and provides these features which are not possible in user-mode SP. We provide a detailed comparison, including an attack on user-mode SP, in the Appendix.

Many other security architectures have been developed to protect execution of software on local or remote devices, such as XOM [9], AEGIS [10], and others [11] [12] [13]. None address all of our goals, especially enabling transient trust and secrets bound to policies. Past work, including TPM [4], did not protect remote execution without relying on permanent factory-installed secrets or verifying and trusting the operating system, and in fact, the entire software stack. Past proposals also used longer public-private keys and more computationally complex public-key cryptography, while we focus on scenarios where the authority has initial access to the device before it is used remotely, allowing us to use shorter symmetric keys and much simpler symmetric-key ciphers in the hardware encryption engine.

TPM also has a different threat model and assumptions. It does not handle physical attacks nor does it provide dynamic verification of code integrity, both of which are addressed by SP architecture (both user-mode and authority-mode). Physical attacks are very likely for mobile devices that are easily lost, captured or stolen – TPM's threat model considers only software attacks. TPM provides only static verification of code upon program launch, and does not defend against dynamic hostile code insertion or modification like we do with dynamic Code Integrity Checking on each fetch of an instruction line into the caches on the microprocessor chip. Instead, TPM relies on measuring and verifying the entire software stack upon program launch. This is unnecessarily limiting (and time consuming) since it requires keeping track of many variations of installed software and verifying the security of large and complex software in order to make even the most basic guarantees of policy enforcement and use of secrets. Furthermore, we expect lower performance using the TPM for all accesses to protected data. Critical key operations occur on the slower co-processor chip rather than the processor itself, over a slow bus interface, and require slower and more costly asymmetric key computations.

7.2 Future Work

There are a number of orthogonal issues raised in the discussion of remote trust that we hope to address in future work. These include providing a secure display that the authority TSM can trust to output data, and secure keyboard or biometric input for secure user authentication to the TSM. These are separate issues from the design and protection of the TSM itself, and will likely require a combination of trusted hardware and carefully designed software drivers, and probably a trusted OS kernel; a trusted Virtual Machine Monitor (VMM) might provide reliable and secure input and display. Past work in this area from desktop computers might be leveraged for use in portable devices. Another orthogonal issue is attestation by the device to the local user. This attestation would assure the user that he is using a good device and viewing secrets provided by the real authority rather than from an adversary.

In future work we will also study dual-mode devices, where the user-centric usage model in user-mode SP is coupled with our authority-mode, on the same device. A single device could be used for protection of both the user's own secrets and the authority's secrets; a single dual-mode processor chip can be manufactured that works in either mode or in both modes simultaneously.

8. CONCLUSIONS

We have developed a new architecture for remote trust in portable devices, where a central authority can provide trusted software that will faithfully enforce its policies on securely stored secrets and data. Our *authority-mode SP* architecture has a very small set of low-complexity hardware features that can be quite easily added to any microprocessor or SOC. We have demonstrated an important example, using this architecture for crisis response where access is critical but security and privacy of sensitive data must be maintained. We provide multiple revocation mechanisms, enabling reliable transient access to this data. Our new persistent secure storage structure, anchored by a root hash (SRH) stored in the processor chip, permits reliable revocation of secrets and associated policies.

A significant contribution of this paper is our demonstration that enhanced security can be provided, including remote and transient trust, with only two new hardware registers, the Device Root Key (DRK) and the Storage Root Hash (SRH) registers. Combined with the Concealed Execution Mode and derived keys, the trusted software can use these secrets to provide robust guarantees remotely to the owner of the device (which we call the authority).

While we have used our earlier user-mode SP architecture as a reference design for simple but highly effective processor-based security, we have made many significant new contributions. On one hand, we verified and leveraged some of the architectural features proposed by the user-mode SP architecture, such as the processor features for providing Code Integrity Checking (CIC) and Concealed

Execution Mode (CEM) for trusted software. On the other hand, we have provided much stronger trust propositions, solving some open problems in user-mode SP. In particular, our authority-mode SP architecture provides remote trust, transient trust, policy-bound secrets and controlled transitive trust—none of which are achievable with the user-mode SP architecture. All this new secure functionality is possible with only two root secrets, the Device Root Key (DRK) and Storage Root Hash (SRH), without increasing the complexity over the user-mode SP hardware architecture.

In conclusion, this paper demonstrates that simple hardware features can be added to provide fundamental hardware anchors that enable more secure systems, without compromising performance, cost or usability. Designing security into the core hardware and software of commodity computing and communications products is a goal of our larger SecureCore collaborative research project [14]. We hope that the authority-mode SP architecture presented in this paper stimulates further research in the design and verification of security-aware processors and hardware-rooted trust that can enhance application and system security.

9. REFERENCES

[1] R. Lee, P. Kwan, J.P. McGregor, J. Dwoskin, Z. Wang. "Architecture for Protecting Critical Secrets in Microprocessors," *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA 2005)*, pp. 2-13, June 2005.

[2] IETF Network Working Group. "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)," Request for Comments: 4279. http://www.ietf.org/rfc/rfc4279.txt

[3] R. C. Merkle. "Protocols for public key cryptography," *IEEE Symposium on Security and Privacy*, pp. 122Ű134, 1980.

[4] Trusted Computing Group. "Trusted Platform Module (TPM) Specifications," April 2006. https://www.trustedcomputinggroup.org/specs/TPM

[5] National Institute of Standards and Technology, "Advanced Encryption Standard," *Federal Information Processing Standards Publication*, FIPS Pub 197, Nov. 2001.

[6] Intel, "LaGrande Technology Architectural Overview," http://www.intel.com/technology/security/, September 2003.

[7] National Institute of Standards and Technology. "The Keyed-Hash Message Authentication Code (HMAC)," *Federal Information Processing Standards Publication*, FIPS Pub 198. http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf

[8] "ITU-T Recommendation X.509, The Directory: Authentication Framework", Int'l Telecomm. Union, Geneva, 2000; ISO/IEC 9594-8.

[9] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. "Architectural Support for Copy and Tamper Resistant Software," *Proc. of the 9th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX).,* pp. 168-177, 2000.

[10] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. "AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing," *Proc. of the 17th Int'l Conf. on Supercomputing (ICS),* 2003.

[11] R. M. Best, "Preventing Software Piracy with Crypto-Microprocessors," *Proc. of IEEE Spring COMPCON Š80,* pp. 466-469, 1980.

[12] T. Gilmont, J.D. Legat, and J. J. Quisquater "An Architecture of Security Management Unit for Safe Hosting of Multiple Agents," *Proc. of the Int'l Workshop on Intelligent Communications and Multimedia Terminals,* pp. 79-82, Nov 1998.

[13] D. Kirovski, M. Drinic, and M. Potkonjak. "Enabling Trusted Software Integrity," *Proc. of the 10th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X),* October 2002.

[14] "SecureCore for Trustworthy Commodity Computing and Communications," *collaborative project by Princeton University, Naval Postgraduate School and University of Southern California.* Project home-page at http://palms.ee.princeton.edu/securecore/

APPENDIX

A. DIFFERENCES WITH USER-MODE SP ARCHITECTURE

The previous SP [1] architecture, which we call user-mode SP in this paper, was designed to protect a user's own secrets and provide portability of his secrets across multiple devices. Portability requires decoupling trust in the device from trust in the user's secrets (e.g., his encrypted keychain). The user can trust multiple devices independently of where his secrets are stored, and could even access his secrets on an unprotected device. It is the user's own responsibility to choose where it is safe to access his secrets.

We found the CIC and CEM architectural features of the user-mode SP architecture to be effective for providing a secure execution environment for trusted code and have used them in our new authority-mode SP architecture. However, the key-chain and master secrets (User Master Key, UMK, and Device Master Key, DMK) defined and protected by user-mode SP, while suitable for a device owned by the user, could not provide the functionality we need for a device owned by a remote authority and not the local user. This functionality included mechanisms providing remote trust, secrets bound to policies, transient trust guarantees, and controlled transitive trust that our new authority-mode SP architecture provides. Before describing each of these key differences in turn, we first point out an attack on SP.

In user-mode SP, the user must maintain physical control of the device to be sure his trusted software module (TSM) has not been replaced by an adversary who gets temporary access and replaces the Device Master Key (DMK). An attacker can change the DMK to DMK', and install a malicious TSM' using DMK'. In user-mode SP, the user will not be able to detect this — he may unknowingly enter his keys for use on a device where an adversary had installed a malicious TSM'. In our authority-mode SP architecture, the authority can always detect whether his good TSM is still running, hashed by the Device Root Key (DRK), which is the secret the authority shares with the device. By enabling a trusted authority to check the DRK on the device, authority-mode SP thwarts the attack of some adversary changing the TSM.

Remote trust requires some attestation ability. User-mode SP has no device attestation capability because no mechanism exists to verify the contents of the DMK. In contrast, our authority-mode architecture makes remote attestation possible due to the shared secret (DRK) between the authority and the device. Our new *DRK_derive* instruction allows generation of derived keys for new communication sessions, based on this shared DRK secret. The generation of a correct new session key, according to the communications protocol implemented by the TSM using the DRK, proves to the authority that the device still has the correct DRK.

Without binding secrets to a TSM, policy-controlled access is also not guaranteed. In user-mode SP, the user's secrets can be accessed on any device with any TSM the user chooses to trust. Therefore, there is no guarantee that any policies will be enforced. In our authority-mode SP architecture, we bind secrets to policies in our new persistent secure storage structure, and the authority can ensure that its good TSM is still running, hence ensuring that the access control policy specified will be implemented and cannot be violated by the local user.

User-mode SP does not provide transient trust reliably, since this requires permanent revocation of keys. While user-mode SP takes care of register replay attacks during interrupts and exceptions of the TSM, it does not provide protection from memory or storage replay attacks using a key that has been revoked (i.e., deleted from the user's key-chain structure). Since user-mode SP does not bind the user's secrets (e.g., his keys) to one device, intentionally allowing the user's key-chain to be usable on multiple devices, one device will not know of changes or accesses made on a different device.

Without a hardware root hash (as in our authority mode's SRH register), no changes can be made permanent, even on a single device — the entire key-chain in user-mode SP can be copied at first and later replayed after changes are made. Because any hashes for integrity protection can be replayed along with the data, a user-mode SP device cannot tell that the data is stale and had been modified. Integrity checking will still ensure arbitrary modifications are not made, but any keys that were once valid will always be valid, making revocation impossible unless the user's root secret (the User Master Key in user-mode SP) is changed. Our authority-mode SP architecture provides reliable revocation of keys in the persistent secure storage structure, hence enabling transient trust.

Lastly, user-mode SP does not provide transitive trust because there is no remote trust mechanism at all. While a third party can give secrets to the user directly, it has no guarantee from another trusted party (e.g., an authority) that its secrets will be protected or used in a particular way. The third party can also supply its own TSM with its secrets directly to a user-mode SP device, but again has no guarantee that this TSM will not be modified or replaced by either an adversary or the device user himself. Furthermore, a user-mode SP device only protects confidentiality of secrets with the user's root secret, so the third party's secrets cannot be hidden from the user.

In summary, while our authority-mode SP architecture leverages the minimalist design philosophy and some features of user-mode SP, it provides significant new security features and defends against some potential attacks on user-mode SP. However, it requires the device user to trust an authority to set up his device in a trusted depot, and does not support portability of the user's key chain across multiple devices or privacy of the user's keys from the authority. Some of these are areas for future research.