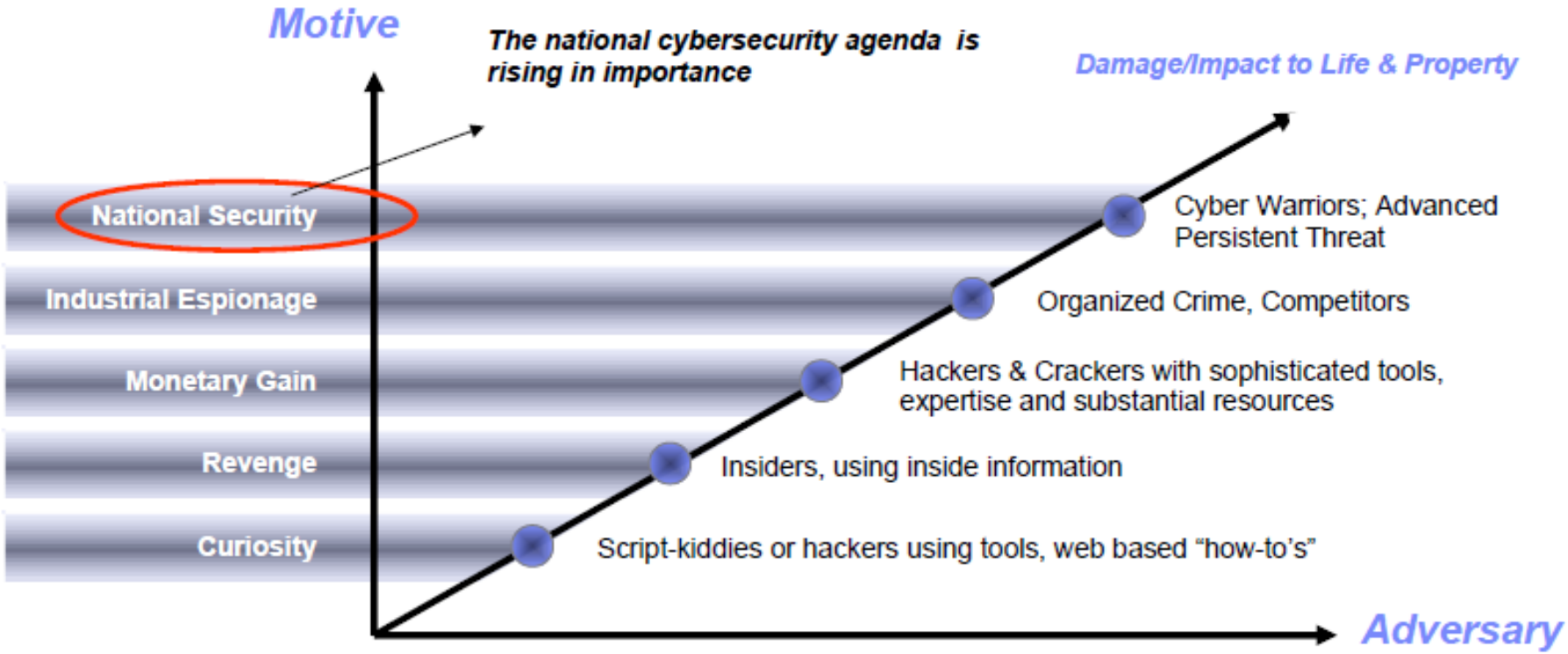
SecureBlue++
CPU Support for Secure Execution

Rick Boivie, Ph.D., IBM T. J. Watson Research Center
rhboivie@us.ibm.com

Peter Williams, Stony Brook University
petertw@cs.stonybrook.edu

09/21/2011

Cyber-Threats Are Becoming More Sophisticated



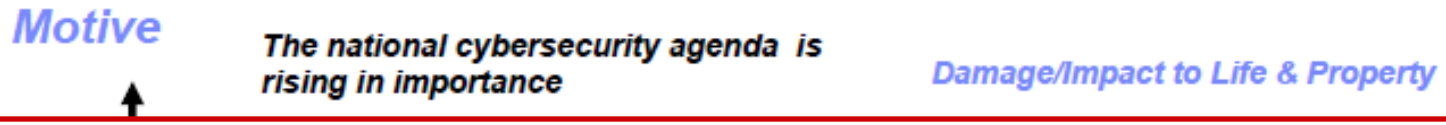
Security breaches are on the increase: cyber attacks have increased 158% since 2006[1], and worldwide cyberattacks increased 30% over the second half of 2008[2].

Sources: [1]US Department of Homeland Security, [2]IBM Internet Security Systems X-Force

Economic impact of cyber attacks on businesses has grown to over $226 billion annually.

Source: Congressional Research Service study

Cyber-Threats Are Becoming More Sophisticated



Some recent news items:

- Security Firm Identifies Global Cyber Spying, NY Times, August 3, 2011
- China Military Paper Urges Steps Against US Cyber War Threat, Reuters, 6/16/2011
- Security Tokens Take Hit, Wall Street Journal, June 7, 2011
- Data Breach at Security Firm Linked to Attack on Lockheed, NY Times, 5/28/2011
- North Korea's Cyber Army Gets Increasingly Sophisticated, foxnews.com, 5/17/2011
- Critical US Infrastructure at Risk of Cyber Attack, foxnews.com, March 22, 2011
- Security Firm is Vague on its Compromised Devices, New York Times, 3/18/2011
- Iran Recruiting Hacker Warriors for its Cyber Army, foxnews.com, March 14, 2011
- Pervasive Memory Scraping Among Most Dangerous Attacks, messagingwire.com, February, 22, 2011
- From Bullets to Megabytes, NY Times, January 27, 2011
- Internet Traffic from US Government Websites Redirected via Chinese Networks, foxnews.com, November 16, 2010

CyberSecurity: A National Priority

- The ability to design and develop secure NIT [Networking and Information Technology] systems is a national priority … The Federal NIT R&D agencies should accelerate development of an R&D infrastructure for creating, testing and evaluating new generations of inherently more secure NIT systems.
  - President's Council of Advisors on Science and Technology, Aug 2007
- Rather than continuing to approach cybersecurity problems in a reactive fashion, using variations of the same tools and approaches, the Department must fundamentally examine its approach to cyber security by moving to a proactive posture, anticipating and eliminating vulnerabilities. … The Department should develop a long term strategy that applies science and mathematics to develop information system architectures and protective measures that go beyond stopping traditional threats to rendering both traditional and new threats harmless.
  - A Scientific Research and Development Approach to CyberSecurity, Dec 2008 (Submitted to Department of Energy by authors from a long list of National Labs and Gov't agencies)

CyberSecurity: A National Priority

- From now on, our digital infrastructure – the networks and computers we depend on every day – will be treated as they should be: as a strategic national asset. Protecting this infrastructure will be a national security priority. We will ensure that these networks are secure, trustworthy and resilient.
  - President Barack Obama, March 29, 2009
- We must develop hardware-based security mechanisms … We must develop detection and response mechanisms and robust cybersecurity protection mechanisms.
  - Workshop on Future Directions in Cyber-Physical Security, Final Report, Jan 2010, Department of Homeland Security
- We have to build our systems on the assumption that adversaries will get in
  - Debora Plunkett, Information Assurance Director, NSA, Dec 2010

One Problem:

- S/W-based systems are large, complex and bug-prone
- Attackers can get their S/W (i.e. malware) into a system by leveraging bugs
- Operating systems, device drivers and applications that run with root privileges are particular problems since
  - They are large and likely to contain bugs
  - Attackers can leverage these bugs and obtain “root” privileges
- Attackers can also get malware in via other means
  - E.g. Attachments that run malware when opened, SQL injection and other code injection attacks, phishing & spear phishing attacks ...
- We need to develop systems in which applications are protected from all the “other software” on a system including
  - Privileged S/W like the OS, drivers or apps that run as root
  - Malware the obtains root privileges via a bug in privileged S/W
  - Malware that somebody with root privileges inadvertently runs by opening an attachment
  - etc.

Goals of our Project

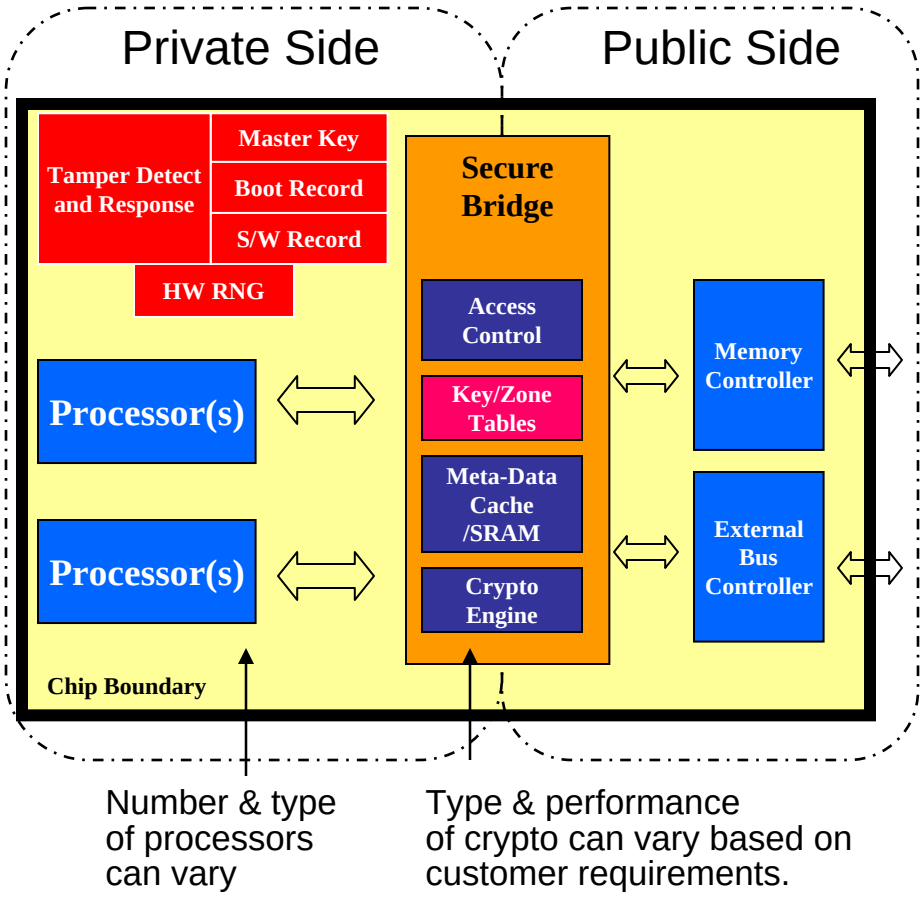
- Protect confidentiality & integrity of sensitive information in a ‘Secure Executable’ so ‘other S/W’ (including privileged S/W or malware that obtains root privileges) cannot access or undetectably tamper with it
- Minimize the amount of code we need to trust
- Use existing OS services for I/O, scheduling, paging, interrupt handling ...
  - But don’t give OS access to any sensitive information
- Run existing programs securely, without significant modification
  - Transparent to applications
- Minimize hardware changes
- Minimize performance impact
- Backwards compatible
  - Existing applications can run without change

SecureBlue++: CPU Support for Secure Execution

- A new CPU architecture that provides support for secure applications
- Protects confidentiality and integrity of information in one application from all the “other S/W” on a system including
  - Privileged S/W like OS’s, device drivers, app’s running with root privileges
  - Or malware that gets root privileges
- Other S/W can’t access or tamper with information in a secure application
  - From “outside” the secure application
  - and can’t introduce malware “inside” an application by exploiting a bug in the application’s interfaces (via e.g. buffer overflow or stack overflow attacks)
- Foundation for strong end-to-end security in a network / cloud environment

- Builds upon our proven **SecureBlue** secure processor technology

Prior SecureBlue Secure Processor Technology

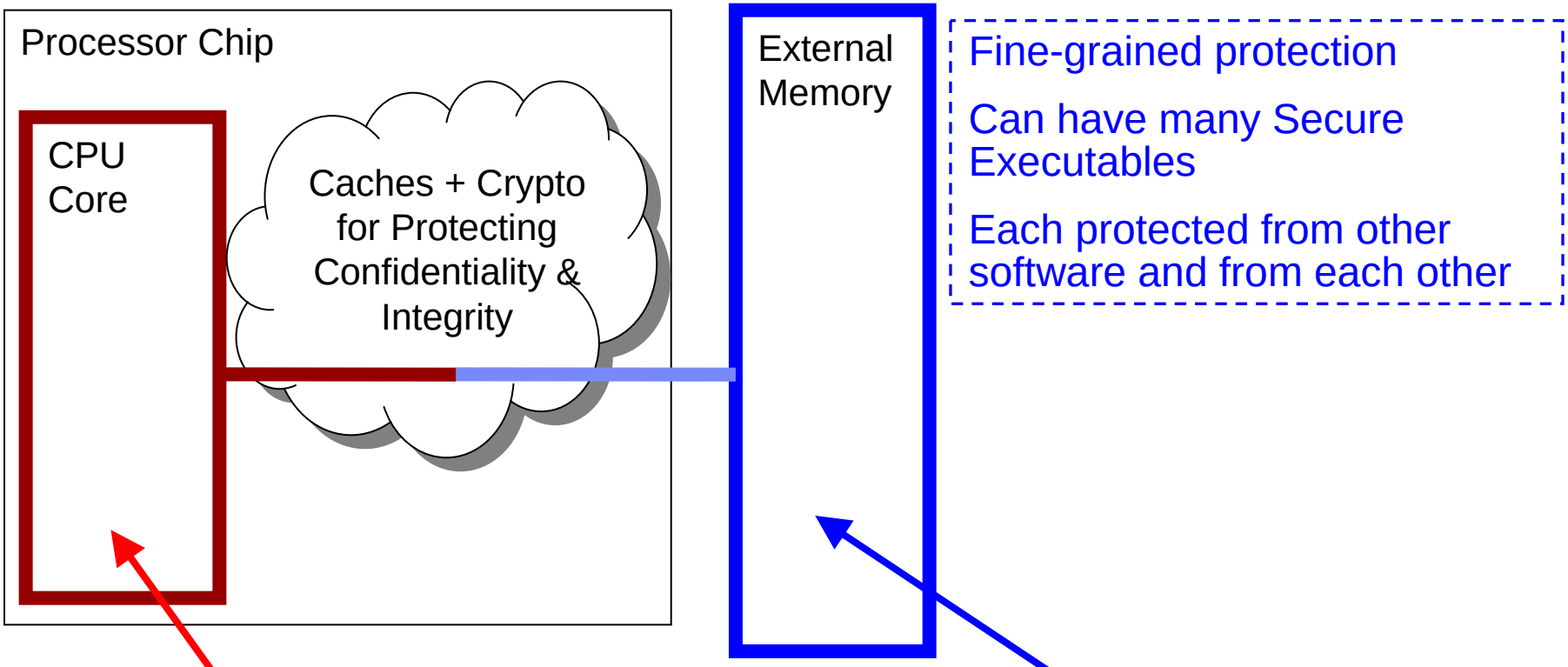


- **Provides strong security against physical attack**
  - Protects confidentiality & integrity of code and data
  - Information is in the clear inside chip
  - Cryptographically protected outside
    - •Encrypt information that goes out
    - •Decrypt & check integrity of information that comes in
  - Minimal impact on cost
  - Minimal impact on system performance
    - • Uses “inline” crypto at memory speeds
  - More than 55 million secure processor SOCs shipped
  - No known security breaches to date
    - •Although hacking systems with these chips has been a “worldwide sport” since 2005

A proven architecture in use by customers today

New Twist on SecureBlue: SecureBlue++

- Like SecureBlue, but provides more fine-grained SecureBlue-like crypto protection to protect information in one program from other S/W (including OS)
  - Protect confidentiality & integrity of information so other S/W cannot read it or undetectably tamper with it



SecureBlue++ Protection

- An application's information is cryptographically protected whenever it is outside the CPU chip, e.g. while information is
  - In memory
  - Swapped out on disk
  - In the file system prior to execution
  - In-transit, traveling across a network
- Under keys that are not available to "other software"
- An integrity tree is used to protect against tampering
- Information inside the CPU chip is in the clear but
  - "Context labels" prevent "other S/W" from accessing or tampering w/ cleartext information in on-chip caches
  - Contents of general purpose registers protected from OS and device drivers
- So, "other software" (e.g. page demon, OS, device drivers, malware with root privileges) can only see the encrypted form of an application's information and can't tamper with it

SecureBlue++ Protection (cont.)

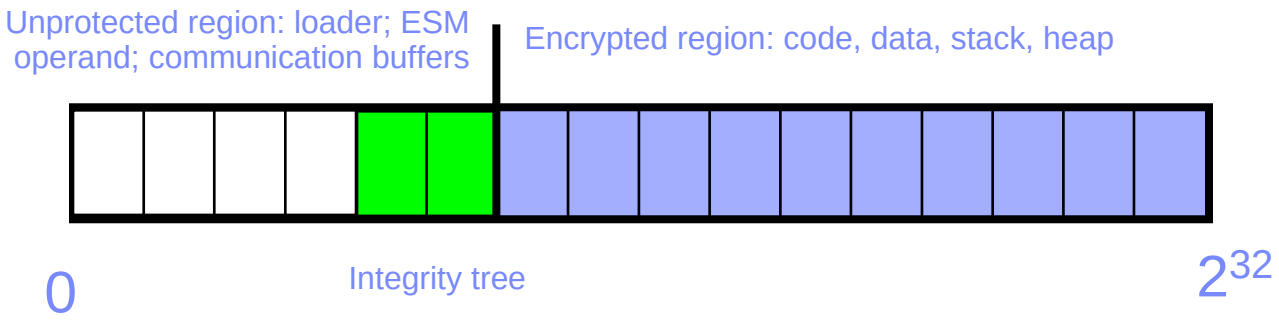
- Confidentiality protection and integrity protection is continuous
  - Not just at boot-time
- Unlike systems that "measure" boot code, OS code & device driver code at system boot and trust this code throughout execution, in a SecureBlue++ system
  - Protection is continuous
  - and an application does not need to trust the OS or "other software"
  - App is protected from all the "other software" on the system
- *Applying the Principle of Least Privilege to OS, drivers etc.*
- An app uses OS services (scheduling, paging, I/O, interrrupt handling ..) but does not trust the OS with any sensitive information
  - the way we use the Internet today for an online purchase via https (use Internet to send our packets, but don't give it access to our sensitive info)
- The only code that an app needs to trust is the code within the app itself
- We can **know** that information in an app is secure without having to prove correctness of millions of lines of OS or driver code or app's running as root

SecureBlue++ Protection (cont.)

- Since the information in an application is always protected, we can “compile in” a root (or roots) of trust that cannot be stolen or tampered with
- Since a “compiled in” private key can’t be stolen, we can provide strong protection against key theft and identity spoofing.
- Since a “compiled in” digital certificate can’t be tampered with, we can provide strong protection for a chain of trust that allows us to determine the authenticity of other entities in a network.
- Importantly, these roots of trust are protected from
  - an adversary that has root access to the system
  - a malware-infected operating system.

A Secure Application (or Secure Executable)

- The build machine creates a binary file of the Secure Executable, containing the encrypted code (and any other secrets to be readable only by the Secure Executable).
  - Builds the initial integrity tree, and attaches a loader, which will issue the ESM (Enter Secure Mode) instruction.
- Encrypted so that only the target CPU can read the binary, and only in Secure Mode.
- Statically linked executable (the host machine libc is not part of our trusted computing base)
- Standard executable binary (e.g. ELF)



ESM (Enter Secure Mode) Instruction

- This binary executes the ESM (Enter Secure Mode) instruction:

  - The Executable Key allows the CPU to decrypt the binary when in Secure Mode
  - Enables memory decryption/encryption for this executable, for a particular region in the address space
    - Encrypt and update integrity on the path from CPU to memory
    - Decrypt and check integrity on the path from memory to CPU
  - Enables memory integrity protection, using the hash tree in the cleartext region
  - Assigns an Executable ID (“EID”)
    - That the hardware and the OS will use to refer to this ‘Secure Executable’

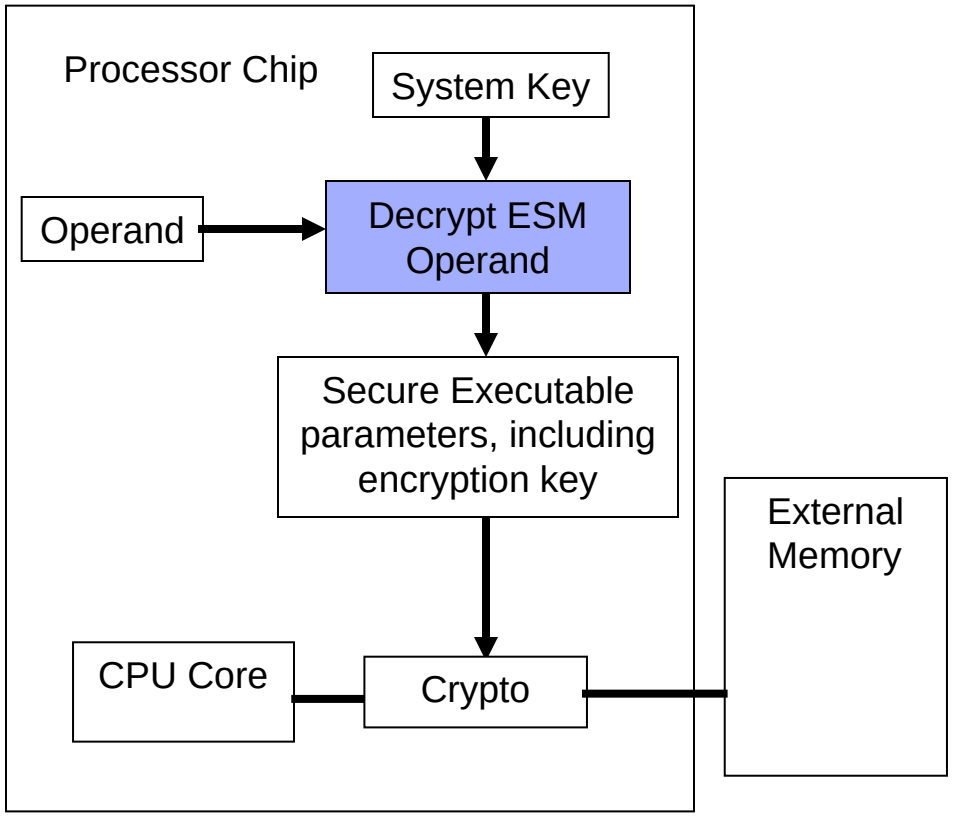
Execution of the ESM Instruction (Entering Secure Mode while protecting an application's keys)

- Esm instruction is used to Enter Secure Mode & load crypto keys
- A Secure Executable's keys are not "in the clear" in its ESM instructions
- They're protected under a "system key" not available to S/W

Execution of an ESM instruction

```
esm <operand>
<encrypted code>
<more encrypted code>
<more encrypted code>
    .
    .
```

- Other S/W cannot decrypt the operand to get a Secure Executable's keys
- Other S/W cannot use the operand with different code or data since that would cause an integrity exception
  - i.e. like when someone tries to tamper with code on SecureBlue

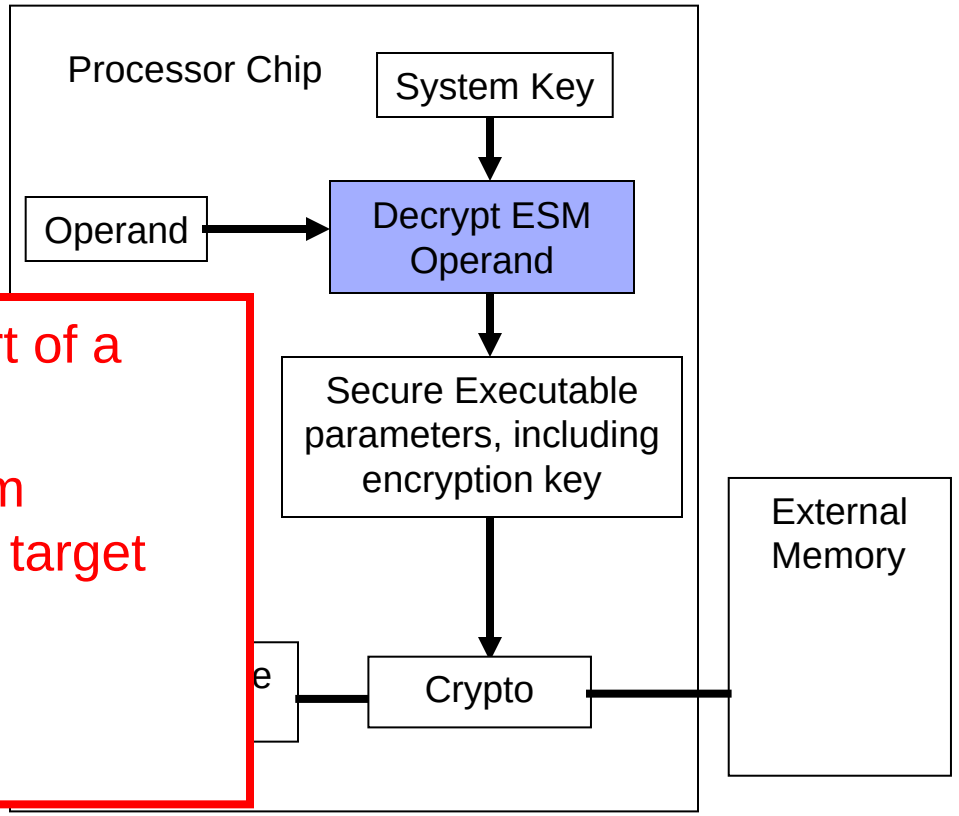


Execution of the ESM Instruction (Entering Secure Mode while protecting an application's keys)

- Esm instruction is used to Enter Secure Mode & load crypto keys
- A Secure Executable's keys are not "in the clear" in its ESM instructions
- They're protected under a "system key" not available to S/W

Execution of an ESM instruction

esm <operand>
<encrypted code>
<more encrypted code>



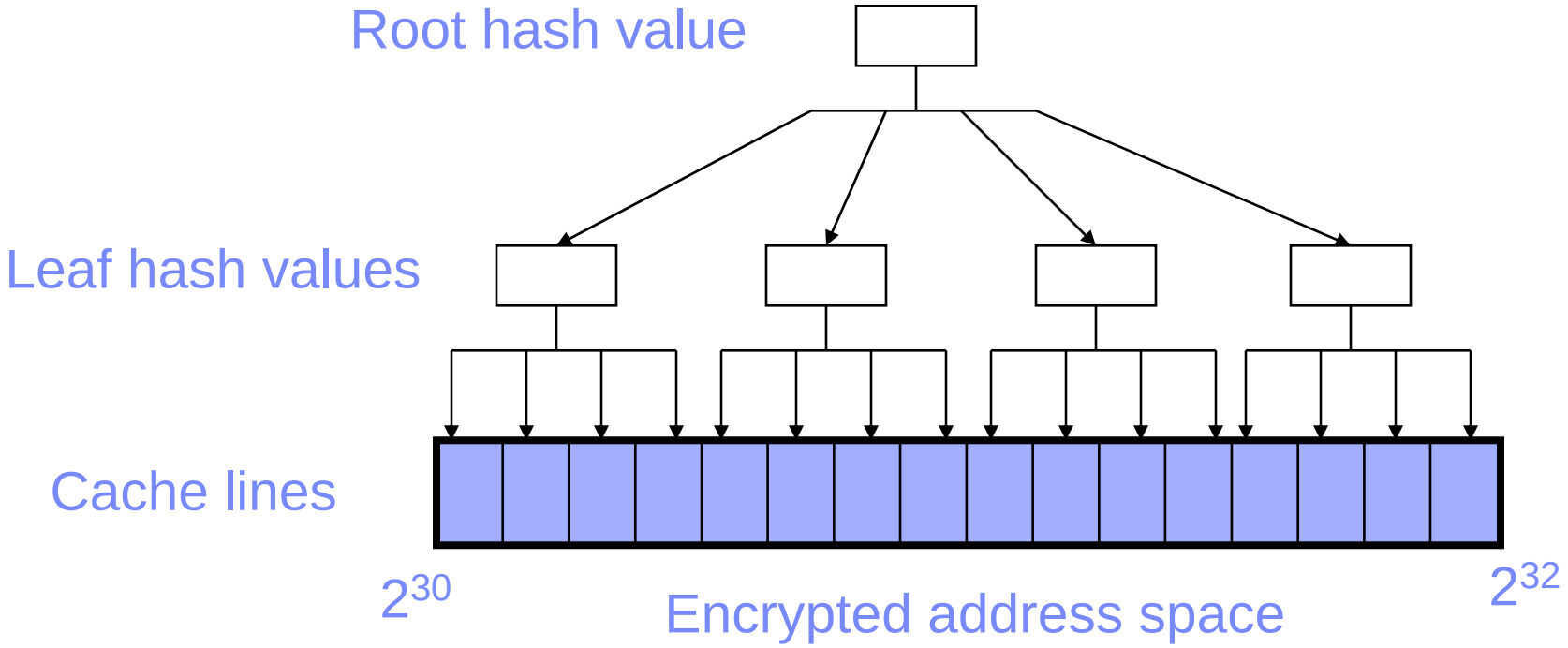
The "System Key" is the private part of a public/private key pair

The build machine encrypts the esm operand using the public key of the target machine

The private key is used by the esm instruction to decrypt the operand

Integrity Protection

- Protects the encrypted portion of the address space (e.g., 2^{30} through 2^{32})



- Cache lines are decrypted & integrity checked as they are brought into the cache
- Encryption/decryption only occur as data moves between on-chip caches and external memory
- Checking/Updating of integrity values only occur when data moves between on-chip caches & external memory
- This may involves several accesses & instruction re-starts (as in page fault handling) but
  - Necessary nodes will normally be cached and cached values can be trusted
- Crypto and integrity overhead close to 0 on cache hits

Cache Protection

- Two new fields for a cache line

Physical addr	Logical addr	EID	LRU counter / dirty bit / etc	Value

- The EID indicates that this cache line belongs to a Secure Executable – that this is a decrypted value that is only accessible when the running EID matches
- If the EID does not match, flush out the entry and treat as a cache miss.
- This other process then sees the ciphertext instead; for example, the paging demon can page out the ciphertext.
- The logical address is also used:
  - As an input to the encryption function (so two identical pages in different locations have different ciphertexts)
  - To prevent re-location attacks
  - The logical address is the address in the process that brought this line into the cache

Paging / Swapping to Disk

- The OS can allocate memory for, and page the Secure Executable in and out as any other process.
- The OS sees only the ciphertext of the process's encrypted information.
- Ciphertext can be swapped to disk and back.
- No special protection (and no code changes) are required to protect the Secure Executable while it is passing through the paging or disk driver code or while it is paged out on disk

Protecting the contents of registers on interrupts

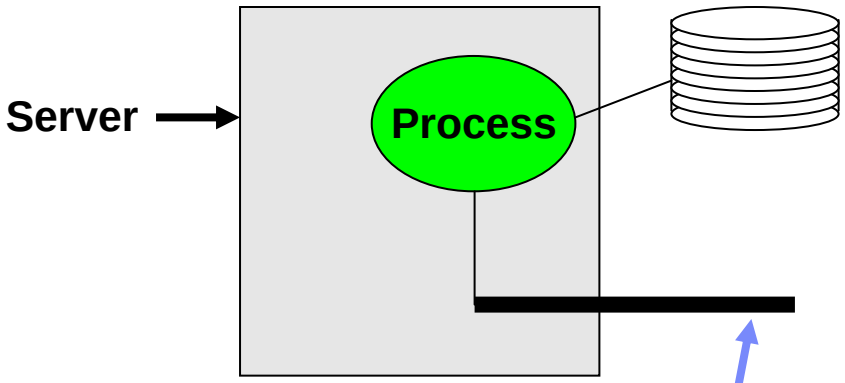
- Register values only accessible when the Secure Executable is running
- Need to hide Secure Executable registers from the OS & interrupt handlers
- On an interrupt, store the register contents in a table, zero out registers before jumping to the interrupt handler.
- The OS kernel gets the ExecutableID (“EID”), to restore the registers later
- Two new instructions
  - RestoreContext (EID) – securely restores and dispatches a previously interrupted Secure Executable
  - DeleteContext (EID) – used at process exit to free resources of a Secure Executable

System Calls / Using the Operating System Without Trusting the Operating System

- Use existing OS services and existing OS system calls without trusting the OS
- Require only minimal changes to run existing software as Secure Executables
- The OS can see only the ciphertext of the protected region; and cannot modify this region without causing an integrity exception
- Use unprotected region of the address space for outside communication
  - System calls communicate with the OS using buffers in this region
  - Optionally protect these buffers with other security mechanisms, e.g.
    - TLS for strong end-to-end protection in a network
    - or to protect information an application wants to store in a file system
    - Note: OS and other software won't be able to access the keys or the information protected by the keys
- Use OS like we use Internet today when we make an online purchase via https
  - **Use but don't trust**
- System call wrappers pass parameters to the OS through unprotected region
  - Largely transparent to applications

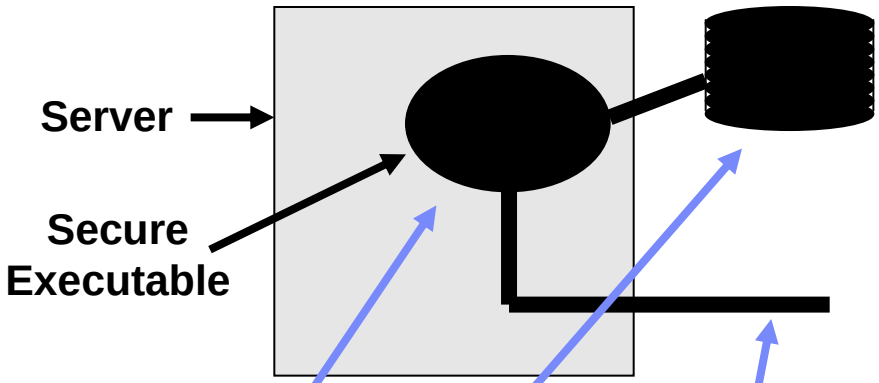
Strong End-to-End Protection of Sensitive Information (even from the operating system, device drivers & applications or malware running with root privileges)

Today



Strong Crypto Isolation/Protection here

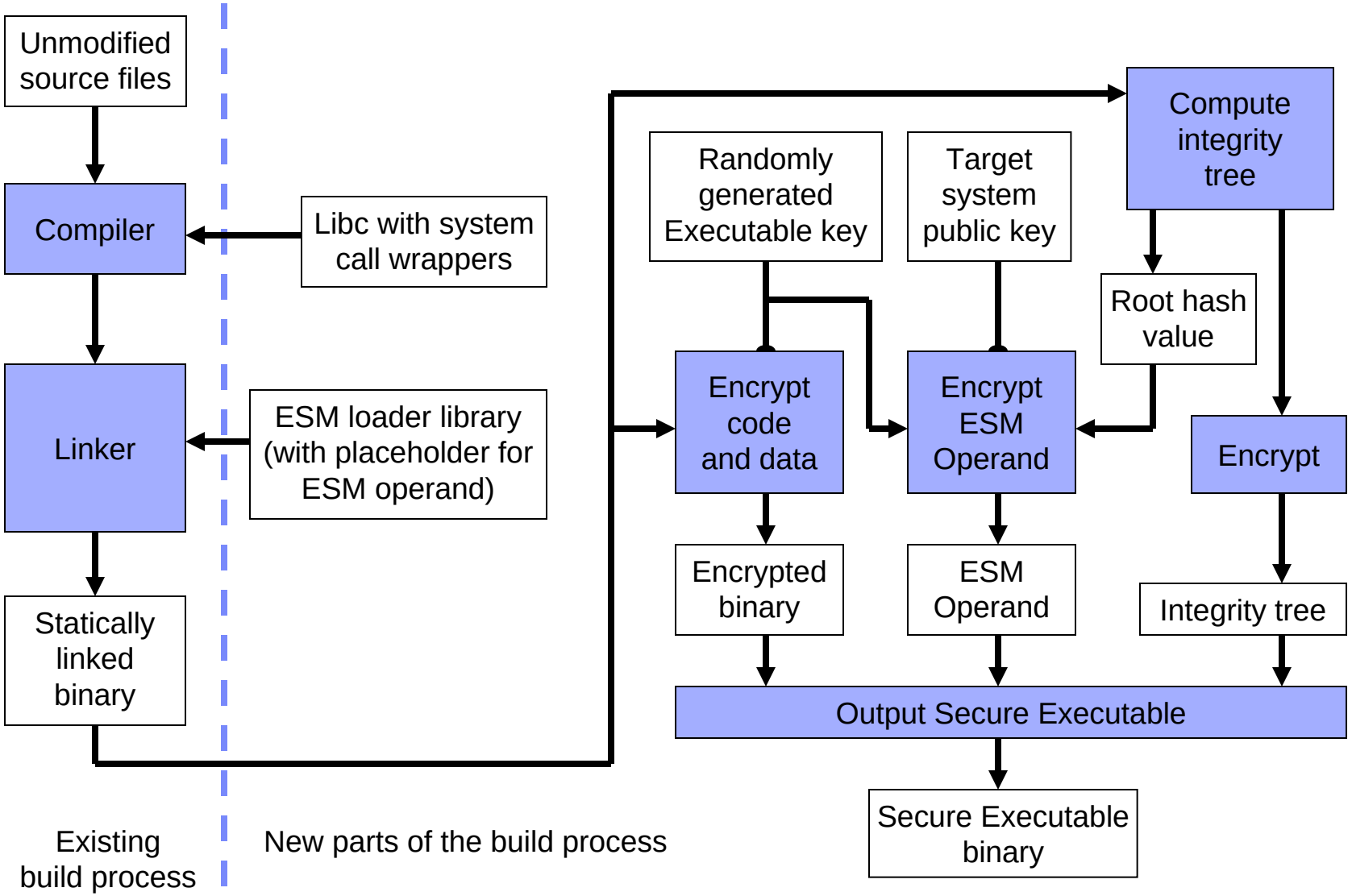
With SecureBlue++



Strong Crypto Isolation/Protection here and here and here

- Keys & data always protected
- Private keys and other data can't be stolen
- Certificates & public keys and other data can't be tampered with

Building a Secure Executable



Status

- Implemented a prototype of a SecureBlue++-enabled CPU on a CPU simulator
- Run a slightly modified Linux on this simulator
  - that uses the restorecontext and deletecontext instructions
- Run some simple demo applications on the slightly modified Linux on our prototype CPU in a real network
- Demonstrated how a simple credit card authorization application can be built as a plain, vanilla application or as a secure executable from the same source code
- Demonstrated that malware can obtain sensitive information from the plain, vanilla application while it's in the file system or while it's in execution
- Demonstrated that this malware cannot obtain information from the secure executable

SecureBlue++ Summary

- Provides fine-grained “SecureBlue-like” crypto protection for sensitive information so “other S/W” can’t access that information or undetectably tamper with it
  - A foundation for strong end-to-end security in a network or cloud environment
- We can know with a high-level of confidence that information in an app is secure
  - W/o having to prove correctness of millions of lines of boot, OS & device driver code or correctness of app’s that run with root privilege
    - & w/o having to re-prove after an OS patch & again after another OS patch … ad infinitum
- Minimal Trusted Computing Base / Applies Principle of Least Privilege to OS & “other S/W”
  - Use OS services -- but don’t trust OS with sensitive information
  - Could allow for stronger security & higher levels of certification at lower cost
- Backwards compatible -- existing applications run w/o modification
  - Don’t need to change millions of lines of existing code
  - But new code can be written, or old code re-built for strong security
- ***Can support U.S. Cybersecurity Requirements***

SecureBlue++ Summary

- Provides fine-grained “SecureBlue-like” crypto protection for sensitive information so “other S/W” can’t access that information or undetectably tamper with it
  - A foundation for strong end-to-end security in a network or cloud environment
- We can know with a high-level of confidence that information in an app is secure
  - W/o having to prove correctness of millions of lines of boot, OS & device driver code or correctn
    - & w/o having … er another OS patch … ad infinitu
- Minimal Trusted Computing Base / Applies Principle of Least Privilege to OS & “other S/W”
  - Use OS se… n
  - Could allow … at lower cost
- Backwards co… h
  - Don’t need … existing code
  - But new code can be written, or old code re-built for strong security
- ***Can support U.S. Cybersecurity Requirements***

Which, in the current environment, seems pretty important

Questions ?