Aegis: A Single-Chip Secure Processor

G. Edward Suh
Cornell University

Charles W. O'Donnell and Srinivas Devadas
Massachusetts Institute of Technology

Editor's note:
An often-used security model for trusted embedded-systems design is to trust the on-chip environment while not trusting the off-chip environment. The authors discuss the Aegis secure, single-chip processor and describe the techniques used to execute private and authenticated software from untrusted off-chip memory.

—Patrick Schaumont, Virginia Tech

As computing devices become ubiquitous and highly interconnected, two contradictory trends are appearing: On the one hand, the cost of security breaches is increasing as more sensitive information and responsibilities are placed on the devices. On the other hand, computing elements are becoming small, disseminated, unsupervised, and physically exposed. Unfortunately, conventional software protection mechanisms do not address physical threats, presenting a significant vulnerability in future computing applications. For example, in digital-rights management (DRM), a computer system owner might try to alter the system behavior to make illegal copies of protected digital content. Similarly, mobile-agent applications require that sensitive electronic transactions be performed on untrusted hosts.[1] But such hosts might be under the control of an adversary who is financially motivated to compromise a mobile agent. In such scenarios, it's easy to bypass software-only protections because attackers have full control of the operating systems and applications such as DRM players or mobile agents.

There have been considerable efforts to address these emerging threats by building a secure computing platform that lets users authenticate the platform and its software. Intel's Trusted Execution Technology (http://www.intel.com/technology/security), formerly named LaGrande Technology, uses the Trusted Platform Module (TPM) from Trusted Computing Group (TCG)[2] to provide authentication mechanisms. Microsoft's Next Generation Secure Computing Base (NGSCB) (http://www.microsoft.com/resources/ngscb/default.mspx) and ARM's TrustZone (http://www.arm.com/trustzone) incorporate similar mechanisms. If a DRM mechanism is implemented on these secure platforms, a content provider can encrypt its protected content only for a specific device executing specific trusted DRM software. Although these systems can detect attacks that tamper with the operating systems or user applications, they cannot protect against physical attacks that tap or probe chips or buses in the system.

In this article, we introduce a single-chip secure processor called Aegis. In addition to supporting mechanisms to authenticate the platform and software, our processor incorporates mechanisms to protect the integrity and privacy of applications from physical attacks as well as software attacks. Therefore, physically secure systems can be built using this processor. Two key primitives, physical unclonable functions (PUFs) and off-chip memory protection, enable the physical security of our system. These primitives can also be easily applied to other secure computing systems to enhance their security.

Secure computing models

A secure computing platform must contain a secret key so that remote parties can authenticate the platform. Also, the platform must protect the integrity and privacy of applications during execution. Here, we compare possible approaches to build a secure computing system based on the implementation of these authentication and protection mechanisms.

Tamper-proof packages

The conventional approach to building physically secure systems is to encase the entire system in

0740-7475/07/$25.00 © 2007 IEEE Copublished by the IEEE CS and the IEEE CASS

a tamper-proof package.[3] For example, the IBM 4758 cryptographic coprocessor contains an Intel 486 processor, a special chip for cryptographic operations, and memory modules (DRAM, flash, and so on) in a secure package. A secret key is stored in a battery-backed RAM. In this case, all the system components can be trusted because they are isolated from physical access.

This approach can provide a high level of physical security and has the advantage of using commodity processors and memory components. However, providing high-grade tamper resistance can be expensive,[4] and active intrusion detection circuitry must be continuously battery powered even when the device is off. In addition, these devices are not flexible—for example, it's difficult to upgrade their memory or I/O subsystems. So, this type of tamper-proof package is not appropriate for pervasive-computing devices that need to be inexpensive and flexible.

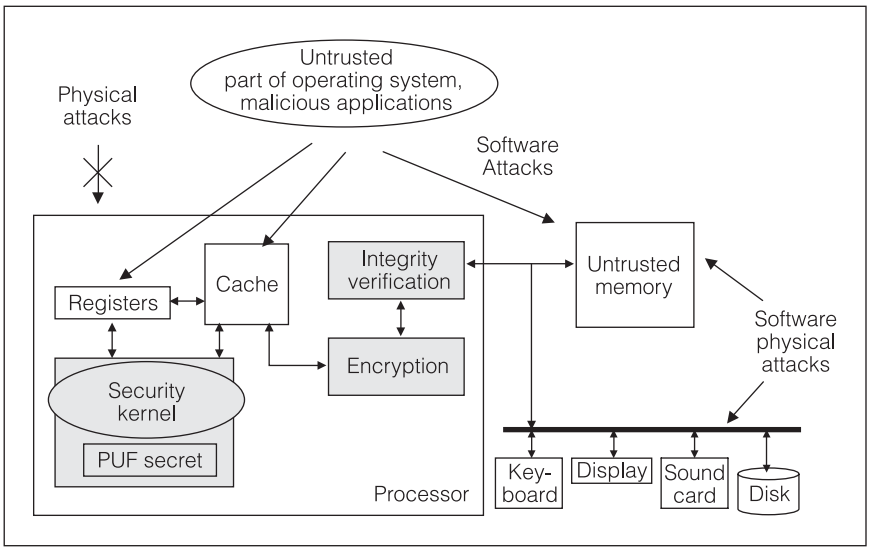

Figure 1. Aegis secure computing model. The shaded parts indicate new components that Aegis adds to a standard processor, for security. (PUF: physical unclonable function.)

Multichip approach

Recent efforts to build secure computing platforms implement security functionality in an auxiliary chip. For example, TCG mounts an additional chip (the TPM) next to the processor on the motherboard. Similar to those used in smart cards, this chip is relatively simple and contains an embedded secret key, which can serve to authenticate the platform. Even though these platforms use multiple chips when implementing the security features, they do not use expensive tamper-proof packages. They simply assume that physical attacks are difficult to execute.

Some advantages of implementing security features in a separate chip are clear: Because the main processor doesn't need special structures such as electrically erasable programmable ROMs (EEPROMs) to store secrets, this approach does not affect the cost of the main processor. Unfortunately, though, communication between the main processor and the adjoining security chip (for example, the TPM) can be easily tampered with. Similarly, communication between the main processor and main memory suffers from the same flaw. Therefore, this approach is not secure against physical attacks.

Aegis approach

Figure 1 illustrates the model upon which Aegis is built. Basically, all trusted components are contained in a single-chip secure processor that includes all security features and secret keys. This processor is protected from physical attacks whenever it is powered on so that its internal state cannot be tampered with or observed directly by physical means. On the other hand, all components outside the processor chip, including external memory and peripherals, are assumed to be insecure; an adversary can observe and tamper with them at will.

Having all the trusted components in a single processor lets us build an inexpensive and secure computing platform. Because only one chip needs to be protected when it is powered on, an expensive battery-backed tamper-proof package isn't necessary. In fact, even without additional protection mechanisms, opening up a chip and tampering with on-chip memory while the processor is running is prohibitively expensive for most low-budget attackers. Moreover, high-security systems could also use active intrusion detection, if necessary. However, unlike the tamper-proof package, the protection mechanism only needs to be active while the power is on. Finally, unlike a multichip approach, in the Aegis approach, physical attacks on external buses cannot compromise system security.

On the other hand, containing all trusted components in the processor chip presents new challenges.

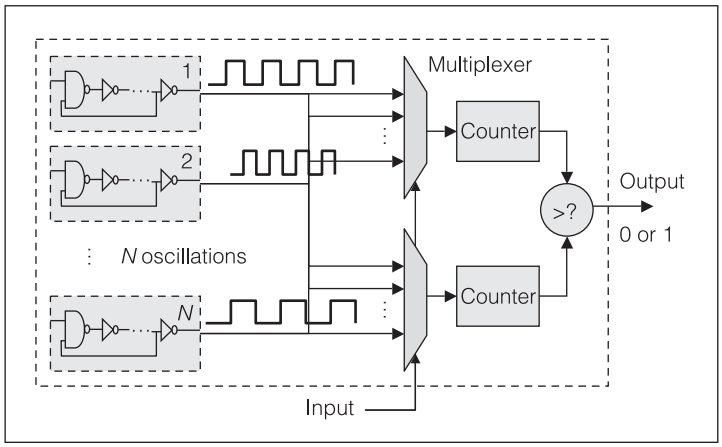

Figure 2. PUF circuit consisting of several ring oscillators (ROs).

First, the secret key must be embedded in the main processor securely without significantly increasing the processor's cost. Unfortunately, existing nonvolatile memory such as EEPROM is neither secure nor cheap to implement in the main processor. We address this problem using PUFs. Second, off-chip memory is still vulnerable to physical attacks. So, the processor must check values read from memory to ensure the execution state's integrity, and must encrypt private data values stored in off-chip memory to ensure privacy.

In this article, we do not consider attacks using side channels such as memory access patterns or power supply voltage.[5] To prevent side-channel attacks, the processor must be equipped with additional countermeasures similar to those developed for smart cards. We also do not address security issues caused by flaws or bugs in software. Finally, we assume the processor has a hardware random-number generator to defeat possible replay attacks on communication.[6]

Physical unclonable functions

An Aegis processor chip must contain a secret so that users can authenticate the processor with which they are interacting. One simple solution is to have nonvolatile memory such as EEPROM or fuses on the chip. The manufacturer can then program, into the nonvolatile memory, a chosen secret key such as a private key and introduce the corresponding public key to the users.

Unfortunately, digital keys stored in nonvolatile memory are vulnerable to physical attacks.[4] Motivated attackers can remove the package without destroying the secret and extract the digital secret from the chip. Storing a digital key in on-chip nonvolatile memory could also increase manufacturing cost and complexity, even for applications in which physical security is not as important. On-chip EEPROMs require more complex fabrication processes compared to standard digital logic. Fuses do not require more manufacturing steps but contain a single permanent key that is easy to read out.

A physical random function, or physical unclonable function (PUF), maps a set of challenges to a set of responses on the basis of an intractably complex physical system. (Hence, this static mapping is a random assignment.) The function can be evaluated only with the physical system and is unique for each physical instance. Therefore, the PUF output can serve as a unique secret for each Aegis chip. Although PUFs can be implemented with various physical systems, we use silicon PUFs that are based on the hidden timing and delay information of ICs.[7,8] Even with identical layout masks, the variations in the manufacturing process cause significant delay differences between different ICs.

PUFs provide significantly higher physical security by extracting secrets from complex physical systems rather than storing them in nonvolatile memory. A processor can dynamically generate many PUF secrets from the unique delay characteristics of wires and transistors. So, an adversary would have to mount an invasive attack while the processor was running and using the secret—a considerably harder proposition. Another advantage of PUFs is that they do not require any special manufacturing process or programming steps.

Ring oscillator PUF

Figure 2 illustrates a PUF delay circuit consisting of many identically laid-out delay loops, or ring oscillators (ROs). This PUF design is thus called RO PUF. Each RO is a simple circuit that oscillates with a particular frequency. (Because of manufacturing variations, each RO oscillates with a slightly different frequency.) To generate a fixed number of bits, a fixed sequence of oscillator pairs is selected, and their frequencies are compared to generate an output bit. The output bits from the same sequence of oscillator-pair comparisons will vary from chip to chip. Given that oscillators are identically laid out, the frequency differences are determined by manufacturing variation, and an output bit is equally likely to be 1 or 0 if random variations dominate.

It is easy to duplicate an RO as a hard macro and ensure that all ROs are identical. There is no need for

careful layout and routing. For example, the paths from oscillator outputs to counters need not be symmetric. By counting many oscillator cycles, the difference in oscillator frequencies can be amplified and will dominate any skews in routing.

Now, we will consider how many bits we can generate from this circuit. Each comparison of an RO pair generates a bit. There are $N(N-1)/2$ distinct pairs, given N oscillators. However, this circuit's *entropy*, which corresponds to the number of independent bits that can be generated from the circuit, is clearly less than $N(N-1)/2$ because the bits obtained from pairwise comparisons are correlated. For example, if oscillator A is faster than oscillator B, the comparison will yield a 1; and if B is faster than C, that comparison will yield a 1. Therefore, when A is compared with C, the comparison must also yield a 1; the bits are correlated.

Fortunately, it's possible to derive this circuit's maximum entropy assuming pairwise comparisons—that is, the number of independent bits that the circuit can generate as a function of N. There are $N!$ different orderings of ROs based on their frequencies. If the orderings are equally likely, the entropy will be $\log_2(N!)$ bits. For example, 35 oscillators can produce 133 bits; 128 oscillators can produce 716 bits; and 1,024 oscillators can produce 8,769 bits.

Also, for simplicity, we can use each RO only once to generate a single bit and avoid any correlation. For example, we can use 128 RO pairs (256 oscillators total) to generate 128 independent bits.

Reliability enhancement

RO frequencies change considerably as environmental conditions such as temperature and voltage change. Of course, we are not using absolute frequencies but rather are doing relative comparisons. The PUF output changes only if the ordering of the two ROs being compared changes.

Figure 3 shows how errors (bit-flips) can occur because of environmental changes. For instance, say ring oscillator 1 is faster than ring oscillator 2 at room temperature. But when the temperature increases, both oscillators slow down, with RO 1 slowing down faster than RO 2, because of different device or physical parameters. These RO frequencies flip when the temperature changes substantially. This flip causes an error in the generated bit.

As Figure 3 shows, ROs with base frequencies that are far apart are much less likely to flip than ROs with frequencies close together. We can use this insight to dramatically reduce the error rate of generated key bits by judiciously selecting which RO pairs to compare. Specifically, we can remove a significant portion of errors if we compare only those RO pairs with frequencies that are far apart, to generate key bits.

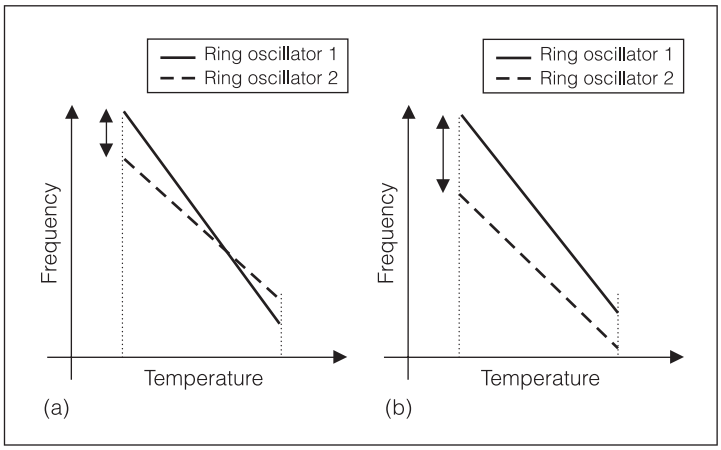

Figure 3. Relationship between RO frequency distance and probability of a PUF output flip when frequencies are close (a) and far apart (b).

The fixed sequence of RO pairs now needs to be k times longer than the desired number of bits to be generated. Then, for each k RO pairs, we choose the pair with the maximum distance between RO frequencies. The bit vector indicating these selections is saved so that the same pairs can be used to regenerate the output. Other masking schemes, such as picking n out of m RO pairs or using a distance threshold, are also possible.

Cryptographic key generation

In secure processors, PUFs must be used for cryptographic primitives such as encryption and digital signatures. Unfortunately, outputs from the PUF circuits as described are inappropriate as cryptographic keys. Because of noise, the outputs will likely be slightly different on each evaluation, even if masking is performed. On the other hand, cryptographic primitives require that every bit of a key stay constant. Moreover, some primitives such as those in the RSA (Rivest, Shamir, Adleman) public-key cryptographic algorithm, require keys to satisfy specific mathematical properties, whereas the PUF outputs are randomly determined by manufacturing variations.

Here, we discuss how PUFs can generate volatile secret keys for cryptographic operations. There are two components: First, the error correction process,

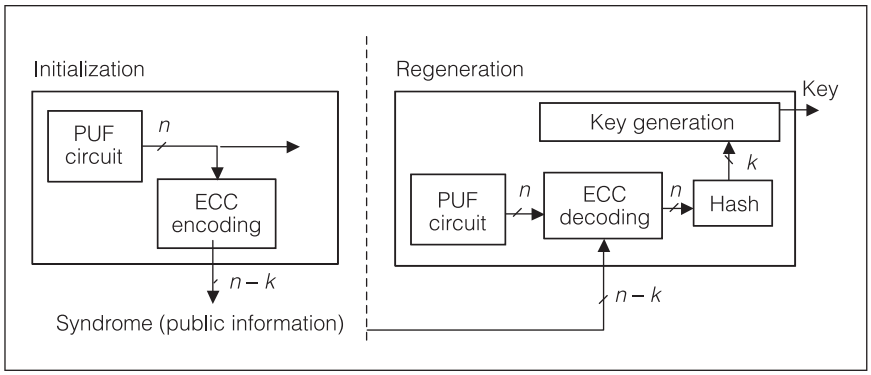

Figure 4. Cryptographic key generation with PUFs. (ECC: error-correcting code.)

which consists of initialization and regeneration, ensures that the PUF can consistently produce the same output even if there are significant environmental changes such as voltage and temperature fluctuations. Second, the key-generation process converts the PUF output into cryptographic keys. Figure 4 shows the overall process.

In the initialization step, an output is generated from the PUF circuit, and the *error-correcting syndrome* for that output is computed and saved for later; for example, the BCH (Bose, Ray-Chaudhuri, Hocquenghem) code can be used to compute the syndrome. (The syndrome is information that enables correcting bit-flips in regenerated PUF outputs.) If a masking scheme is used, the bit vector that selects RO pairs will also be stored along with the syndrome. Note that the syndrome and this bit vector are public information and can be stored anywhere (on chip, off chip, or remotely on a server).

To regenerate the same PUF output, the PUF first produces an output from the circuit. If there is a saved bit vector, it is used to select pairs. Then, the PUF uses the syndrome from the initialization step to correct any changes in the circuit output. In this way, the PUF can consistently reproduce the output from the initialization step.

Clearly, the syndrome reveals information about the PUF delay circuit output. In general, however, given the b-bit syndrome, attackers can learn at most b bits about the PUF delay circuit output. Therefore, to obtain k secret bits after the error correction, Aegis generates $n = k + b$ bits from the PUF delay circuit. Even with the syndrome, an adversary must still guess at least k bits to find the correct PUF response. For example, we can use the BCH (127, 64, 21) code to reliably generate 64-bit secrets. The BCH (n, k, d) code can correct up to $(d - 1)/2$ errors out of n bits with an $(n - k)$-bit syndrome ($b = n - k$).

Although the mask could reveal information about which RO frequencies are far apart, it does not reveal information about the sign of comparisons—that is, the bits generated. If an RO is used many times to generate bits, then an adversary could possibly extract information from the mask about ordering of ROs. However, Aegis can protect against this extraction by using each RO only once to generate a single bit.

For cryptographic operations that use a randomly selected number as a key, the output of the error-correcting code (ECC) can be simply hashed down to a desired length and used as a cryptographic key. For example, symmetric-key algorithms such as the Advanced Encryption Standard (AES) algorithm can use the hashed PUF output.

For cryptographic operations with keys that must satisfy special properties—for example, an RSA key pair—the hashed PUF output serves as a seed for a key-generation algorithm. In this way, the PUF can generate keys for any cryptographic operation. Note that PUFs simply generate keys that can be used with a standard cryptographic algorithm; no change is required in cryptographic algorithms.

Experimental validation

We tested the RO PUF circuit on 15 Xilinx Virtex4 LX25 FPGAs (90-nm technology),[9] where all FPGAs were exactly the same model and therefore identical designs. For the experiments, we placed 1,024 ROs on each FPGA and used the 1-out-of-8 mask scheme.

The experimental results show that two identical PUF circuits on two different FPGAs produce a different output bit with a probability of 46.15% on average (interchip variation)—quite close to the ideal average of 50%. On the other hand, multiple measurements on the same chip differ only with 0.48% probability (intrachip variation), even for the worst-case environmental change. We studied intrachip variation by changing the ambient temperature of the FPGAs from −20°C to 120°C and by changing the core voltage from 1.08 V to 1.32 V (± 10%). The results show that the intrachip variation is far lower than the interchip variation, even for the worst-case environmental change.

From the interchip and intrachip variations, we can estimate how reliable the PUF-generated cryptographic

keys will be. For example, if the BCH (127, 64, 21) code is used to generate a key, 10 errors in a 127-bit PUF output can be corrected, and the probability of failing to regenerate the same key is less than 5×10^{-11}.

Processor architecture

The Aegis processor can shield against software and physical attacks by protecting a program before it is executed, during execution, and during processor mode transitions. When an application initially runs, the processor uses a program-hashing technique to verify that the program was not corrupted while it was held in unprotected storage. During execution, the processor uses integrity verification, memory encryption, and access permission checks to guarantee security under four different secure execution modes. Finally, the transition between secure execution modes is carefully structured and monitored.

Typical processors contain user and supervisor modes, which control access to special functions such as virtual-memory mechanisms. Within user and supervisor modes, Aegis additionally provides the following modes:

- standard (STD), which has no additional security measures;
- tamper evident (TE), which ensures program state integrity;
- private tamper-resistant (PTR), which additionally ensures privacy; and
- suspended secure-processing (SSP), which allows an application running under TE or PTR mode to safely execute insecure regions of the program, thus reducing the need for a large amount of trusted code and allowing drivers and third-party libraries to run safely.

Here, we summarize the protection capabilities of each of these modes. The TE mode has all the capabilities of STD mode, and PTR mode has all the capabilities of TE and STD modes. STD and SSP modes have the following characteristics:

- They provide read and write access to unprotected memory.
- Standard code can be executed (in an unprotected fashion).
- Software in STD or SSP mode can call only one of the security instructions that are entering or reentering TE or PTR mode.

TE mode provides read and write access to verified memory, as well as access to most security instructions. PTR mode provides read and write access to private memory and access to PUF instructions.

Authentication

Aegis lets users authenticate the processor and software. For this purpose, each processor has a unique secret key securely embedded using a PUF, as we discussed earlier. For example, each processor can have its own private key, with a corresponding public key that is known to users. Then, the processor can sign a message with the private key to authenticate itself to the users.

To support software authentication, Aegis combines program hashes with a digital signature, as in Microsoft NGSCB or TPM. When the operating system starts and enters a secure execution mode (TE or PTR), Aegis computes the cryptographic hash of the operating system's trusted part, which is called the security kernel. This program hash is stored in a secure on-chip register and is always included in the signature. Therefore, when users verify a signature from the processor, they know that the message is from a particular security kernel running on a particular processor.

The security kernel provides the same authentication mechanism to user applications by computing their hashes when user applications enter a secure computing mode. Although we have described an authentication scheme using private and public keys, it is also possible to use different protocols optimized for PUFs.[10]

Memory protection

The TE and PTR security modes must guarantee the integrity and privacy of instructions and data in memory under both software and physical attacks. To defend against software attacks, the processor performs additional access permission checks within the memory management unit (MMU). To defend against physical attacks, Aegis uses integrity verification (IV) and memory encryption (ME) techniques. These defenses are not enabled at startup, but rather are initiated when a supervisor program switches into TE or PTR mode.

The processor separates physical memory space into regions designated *IV protected* and *ME protected* (allowing overlap), which have boundaries specified upon entrance into TE or PTR mode. The processor

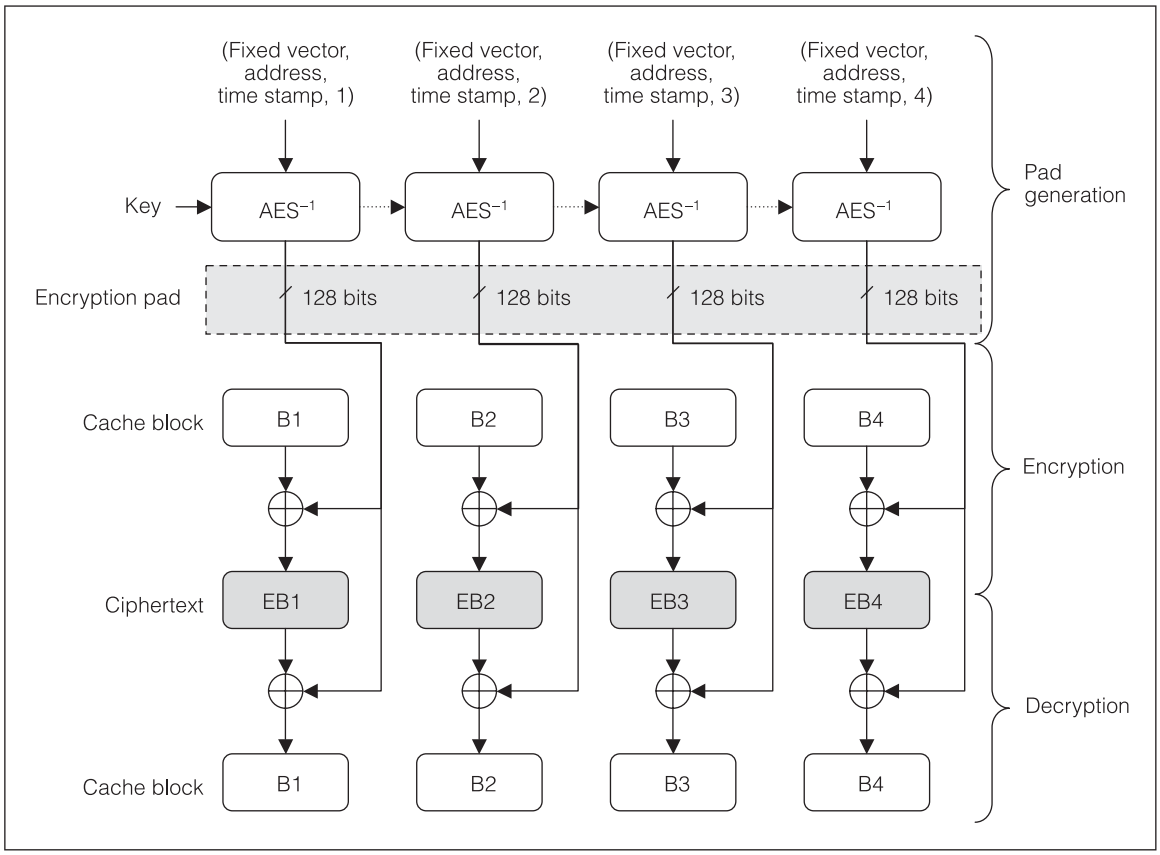

Figure 5. One-time-pad (countermode) encryption mechanism. (AES^{-1} indicates decryption using the Advanced Encryption Standard algorithm.)

has an IV mechanism to detect any tampering that changes the content of the IV regions, and an ME mechanism to guarantee the privacy of the ME regions. For efficiency reasons, we further divided the IV and ME regions into static and dynamic subsections—corresponding to read-only data (such as application instructions) and read-write data (such as heap and stack variables), respectively.

For memory encryption, Aegis encrypts and decrypts all off-chip data transfers in the ME regions using a one-time-pad, or countermode, encryption scheme.[11] Figure 5 shows how an evicted cache block is sent through XOR logic with an AES encryption of its memory address, a time stamp, and some constant bit vector **V**. The time stamp is small and is stored in memory. During a cache block fetch, decryption latency is hidden because the time stamp can be fetched and used to recompute a pad while the larger cache block is still being loaded from memory. For the static ME region, the pad computation can start even earlier because no time stamp is required.

The processor protects the dynamic IV region by creating a hash tree and saving the root hash node on chip (see Figure 6).[12] In this way, any tampering of off-chip memory will be reflected by a root hash that does not match the saved one. The same hash tree also protects the encryption time stamps for the dynamic ME region that overlaps with the dynamic IV region. Static IV regions are protected differently. Because the static region is read only, replay attacks (substituting the new value with an old value of the same address) are not a concern. In this case, cryptographic message authentication codes (MACs) are taken over the static IV region's address and data values, and are stored in a reserved portion of the unprotected memory.

To reduce verification latency, the IV mechanism runs in the background, stalling main execution only when necessary to catch up when a security instruction must be executed or when a store occurs to nonprivate memory while in PTR mode. This guarantees that all security instructions have been verified, and protects private data from leaking into nonprivate

memory. Finally, access permission checks guarantee that processes operating within either SSP or STD mode cannot access any of the IV- or ME-protected memory regions.

Multitasking

Secure multitasking on the Aegis processor can be ensured with the help of a trusted security kernel to handle such functions as virtual-memory management (VMM). In this model, a trusted security kernel is started after boot-up, and it transitions the processor into TE or PTR mode before starting the VMM system. Both the security kernel and user applications can use four protected regions in virtual-memory space to provide different levels of security:

- read-only (static) verified memory,
- read-write (dynamic) verified memory,
- read-only (static) private memory, and
- read-write (dynamic) private memory.

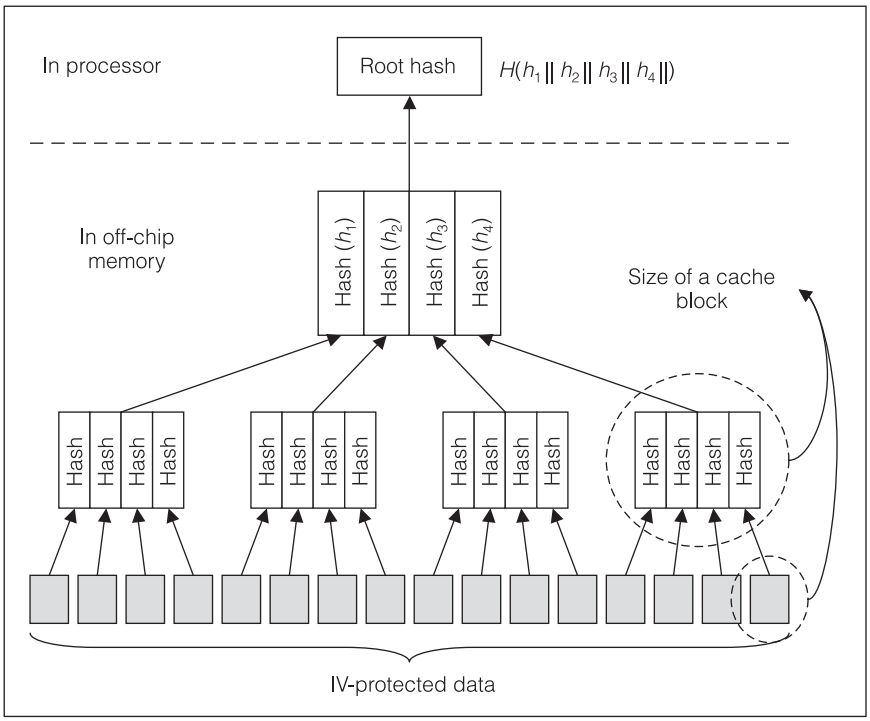

Figure 6. Hash tree protection of integrity verification (IV) region.

Figure 7 shows how the Aegis processor separates physical memory to allow a security kernel to safely map virtual addresses.

Only single dynamic IV and ME regions are required because a security kernel can share this memory space with user processes. However, the processor provides user- and supervisor-level static IV and ME regions separately because these regions depend on specific decryption keys, which can differ between the security kernel and a user application. The security kernel also protects against malicious programs by isolating the memory space of each user process. This includes separate regions within the dynamic IV and ME regions, as well as separations within the user processes' static IV and ME regions. Finally, on a context switch, the security kernel is responsible for saving and restoring the user's secure mode and the memory protection regions as part of the process state.

Debugging support

The Aegis processor supports full debugging by default while in STD mode, but requires it to be specifically enabled while in protected modes. The processor includes whether debugging is enabled or

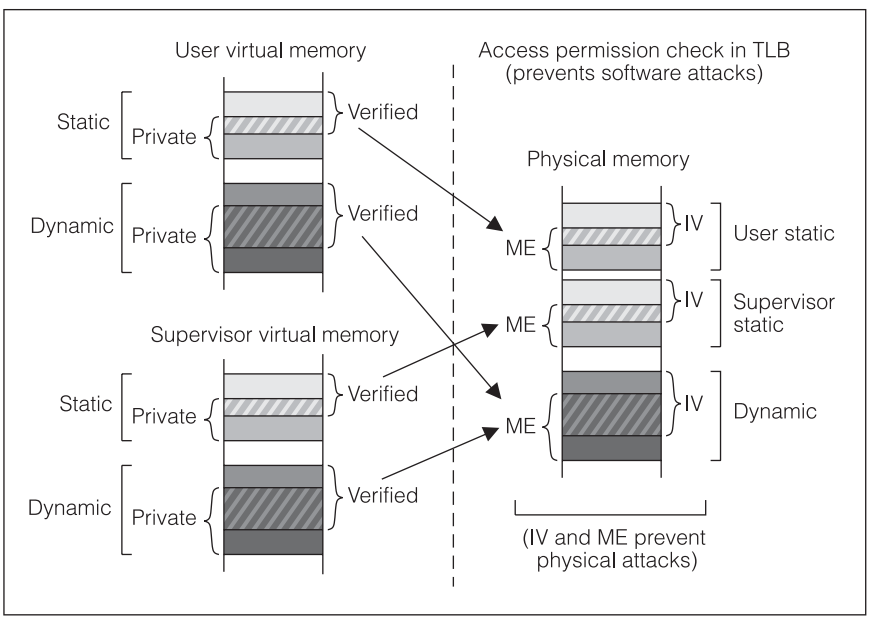

Figure 7. Protected regions in virtual and physical memory. (ME: memory encryption.)

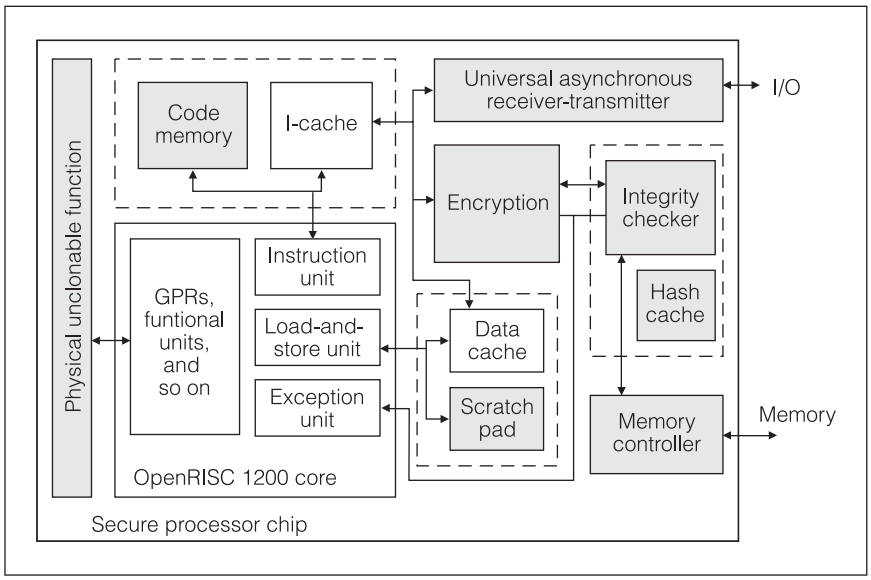

Figure 8. Aegis core implementation overview. Shaded areas indicate new components added for Aegis. (GPR: general-purpose register; RISC: reduced-instruction-set computing.)

not in its computation of the program hash. Thus, the security kernel will have different program hashes, depending on whether debugging is enabled or not. In this way, the security kernel can be debugged when it is developed, but the debugging will be disabled when the security kernel needs to execute securely. This idea is similar to the one incorporated in Microsoft NGSCB (http://www.microsoft.com/resources/ngscb/default.mspx).

Overhead

The security capabilities we've discussed have associated costs. The added hardware mechanisms increase the processor core's size and marginally degrade program performance. To analyze this overhead, we implemented an embedded Aegis processor core on a Xilinx Virtex2 FPGA based on the OR1200 processor core from the OpenRISC project (http://www.opencores.org/projects.cgi/web/or1k). We added the PUF circuit, the IV mechanism, and the ME mechanism to the core, as Figure 8 shows. We implemented security instructions in firmware because such instructions are complex and infrequently used. However, the embedded-memory requirement to hold and execute these instructions was only about 12 Kbytes.

PUF circuit and key-generation code

The PUF circuit is particularly small compared to the size of an unmodified OR1200 core. After we ran this Aegis core and the OR1200 core through an ASIC synthesis tool, the PUF circuit size was only 2,691 gates, or roughly 4.5% of the embedded OR1200 core's size. (This result is actually from a previous PUF circuit design based on delay paths and an arbiter instead of the RO PUF design presented in this article.[10] We expect the size of the RO PUF circuit to be comparable to the previous design.)

The PUF initialization and key generation are implemented in firmware, and take 1.1 million and 3.2 million cycles, respectively. Although this overhead might seem high, these operations are performed a few times in an entire program. So, this overhead is actually negligible when compared to the long execution times of typical programs.

Hardware costs

The IV mechanism, the ME mechanism, and the permission access checks within the MMU are the only other modifications that required adding more logic to the processor core. Using an ASIC synthesis tool, we found that the IV mechanism required 107,756 gates, whereas the ME mechanism and access checks required 86,655 and 11,587 gates. All told, the hardware modifications are modest compared to the size of current commercial cores.

System performance

The main performance overhead of the Aegis processor comes from the two off-chip memory protection mechanisms, in two different ways:

- *Bus contention*. The IV and ME mechanisms share the same memory bus to store metadata such as hashes and time stamps.
- *Memory latency*. Encrypted data must be decrypted before the processor can use it.

Because bus traffic depends on the rate of cache block evictions, the performance overhead also heavily depends on the cache miss rate. A higher miss rate increases the amount of processor data that is sent off chip and that needs to be verified and encrypted.

To estimate the worst-case overhead, we used a synthetic benchmark that simply reads a large array

in the memory for varying cache miss rates. The percentage slowdown of a program while running in TE mode ranges from 3.8% for a data cache miss rate of 6.25%, to a maximum overhead of 130% when the processor has no cache at all. Similarly, PTR mode exhibits a slowdown of 8.3% and 162%.

More realistic embedded benchmarks, such as the EEMBC (Embedded Microprocessor Benchmark Consortium) benchmark suite, show an average percentage slowdown of only 0.1% for programs running in TE mode, and 1.3% for PTR mode. Results from a wider range of benchmarks are also promising, and can be found elsewhere.[10]

THE AEGIS PROCESSOR ARCHITECTURE can help build computing systems that are secure against both software and physical attacks, with minimal performance overhead for typical embedded applications. However, some aspects of the current Aegis design can be improved. First, its performance overhead can be noticeable for memory-intensive applications, leaving room for more efficient encryption and verification mechanisms. Second, the Aegis architecture assumes that the processor chip is secure from physical attacks; protecting an IC from invasive attacks or side-channel attacks is an active research area. Finally, wide deployment of secure processors requires a key infrastructure that can easily certify public keys of trusted processors. We are in the process of improving Aegis and building a secure embedded-processor ASIC. ■

■ References

1. J. Claessens, B. Preneel, and J. Vandewalle, "(How) Can Mobile Agents Do Secure Electronic Transactions on Untrusted Hosts? A Survey of the Security Issues and the Current Solutions," *ACM Trans. Internet Technology*, vol. 3, no. 1, Feb. 2003, pp. 28-48.
2. Trusted Computing Group, *TCG TPM Specification*, v1.2, rev. 103, by Trusted Computing Group, 2003-2006, https://www.trustedcomputinggroup.org/specs/TPM.
3. S.W. Smith and S.H. Weingart, "Building a High-Performance, Programmable Secure Coprocessor," *Computer Networks*, vol. 31, no. 8, Apr. 1999, pp. 831-860.
4. R.J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, John Wiley and Sons, 2001.
5. P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Proc. 19th Ann. Int'l Cryptology Conf. Advances in Cryptology* (CRYPTO 99), LNCS 1666, Springer-Verlag, 1999, pp. 388-397.
6. C.S. Petrie and J.A. Connelly, "A Noise-Based IC Random Number Generator for Applications in Cryptography," *IEEE Trans. Circuits and Systems I*, vol. 47, no. 5, May 2000, pp. 615-621.
7. B. Gassend et al., "Silicon Physical Random Functions," *Proc. 9th ACM Conf. Computer and Communication Security*, ACM Press, 2002, pp. 148-160.
8. J.-W. Lee et al., "A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications," *Proc. Symp. VLSI Circuits*, IEEE Press, 2004, pp. 176-179.
9. G.E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," *Proc. 44th Design Automation Conf.* (DAC 07), ACM Press, 2007, pp. 9-14.
10. G.E. Suh et al., "Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions," *Proc. 32nd Ann. Int'l Symp. Computer Architecture* (ISCA 05), IEEE CS Press, 2005, pp. 25-36.
11. G.E. Suh et al., "Efficient Memory Integrity Verification and Encryption for Secure Processors," *Proc. 36th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, IEEE CS Press, 2003, pp. 339-350.
12. B. Gassend et al., "Caches and Hash Trees for Efficient Memory Integrity Verification," *Proc. 9th Int'l Symp. High-Performance Computer Architecture* (HPCA 03), IEEE CS Press, 2003, pp. 295-306.

G. Edward Suh is an assistant professor in the School of Electrical and Computer Engineering at Cornell University. His research interests include secure embedded processors, architectural techniques for security and verification, and new programmable substrates for simplified synthesis. Suh has a BS in electrical engineering from Seoul National University, and an SM and a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology. He is a member of the IEEE and the ACM.

Charles W. O'Donnell is pursuing a PhD in computer science at the Massachusetts Institute of Technology. His research interests include computer architecture, security, algo-

rithms, and computational biology. O'Donnell has a BS in computer engineering from Columbia University and an SM in electrical engineering and computer science from the Massachusetts Institute of Technology. He is a member of the IEEE.

Srinivas Devadas is a professor and associate head of the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. His research interests include CAD of VLSI computing systems, computer architecture, and computer security. Devadas has a BTech in electrical engineering from IIT Madras, India, and an MS and a PhD in electrical engineering and computer science from the University of California, Berkeley. He is a Fellow of the IEEE.

■ Direct questions and comments about this article to Edward Suh, School of Electrical and Computer Engineering, Cornell University, 338 Rhodes Hall, Ithaca, NY 14853; suh@csl.cornell.edu.