available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Providing secure execution environments with a last line of defense against Trojan circuit attacks

Gedare Bloom[a,*], Bhagirath Narahari[a], Rahul Simha[a], Joseph Zambreno[b]

[a]The George Washington University, Department of Computer Science, Academic Center 7th Floor, 801 22nd Street NW, Washington, DC 20052, United States
[b]Iowa State University, Department of Electrical and Computer Engineering, 2215 Coover Hall, Ames, IA 50011, United States

ARTICLE INFO

Article history:
Received 30 January 2009
Received in revised form
12 March 2009
Accepted 13 March 2009

Keywords:
Trojan circuit
Dual-encryption
HW/SW co-design
IC trust
Information assurance

ABSTRACT

Integrated circuits (ICs) are often produced in foundries that lack effective security controls. In these foundries, sophisticated attackers are able to insert malicious Trojan circuits that are easily hidden in the large, complex circuitry that comprises modern ICs. These so-called Trojan circuits are capable of launching attacks directly in hardware, or, more deviously, can facilitate software attacks. Current defense against Trojan circuits consists of statistical detection techniques to find such circuits before product deployment. The fact that statistical detection can result in false negatives raises the obvious questions: can attacks be detected post-deployment, and is secure execution nonetheless possible using chips with undetected Trojan circuits? In this paper we present the Secure Heartbeat And Dual-Encryption (SHADE) architecture, a compiler–hardware solution for detecting and preventing a subset of Trojan circuit attacks in deployed systems. Two layers of hardware encryption are combined with a heartbeat of off-chip accesses to provide a secure execution environment using untrusted hardware. The SHADE system is designed to complement pre-deployment detection techniques and to add a final, last-chance layer of security.

1. Introduction

As the integrated circuit (IC) manufacturing industry pursues ever smaller dimensions, the economics of the industry has resulted in the separation of IC designers from the foundries that manufacture the chips. Foundries that can fabricate high quality, specialized ICs have become prohibitively expensive to build and maintain. As a result, most foundries are located offshore and contract services to multiple consumers. Whether by competition between chip designers or between nations, the relationship between designer and foundry can become adversarial. This situation raises the possibility that such foundries, which are not necessarily subject to tight security, may be compromised.

One type of harmful compromise arises when a foundry exploits the sheer complexity of modern circuits to insert a *Trojan circuit* (also called hidden circuit, malicious circuit) into the IC. Such insertion is relatively straightforward because a chip's design is conveyed to a foundry in a well-understood digital format that is routinely edited as part of normal foundry procedures. It is easy for a single individual to subvert the editing process and to insert a circuit that lingers for the remainder of the fabrication process. The ease with which Trojan circuits can be inserted has not gone unnoticed by those who assess such risks. Some reports and articles already describe and analyze such attacks and their consequences to the semiconductor manufacturing industry (Adee, 2008; Defense Science Board, 2005; King et al., 2008; US Senator, 2003).

* *Corresponding author.*
E-mail address: gedare@gwmail.gwu.edu (G. Bloom).

0167-4048/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2009.03.002

What attacks are possible with a Trojan circuit? Perhaps the simplest attack is to effectively shut down a chip at an opportune moment. Another type of attack is to leak sensitive information. For example, in fully-encrypted execution platforms, both data and code are encrypted in memory and decrypted inside the processor; a Trojan circuit inside the processor can bypass the encryption logic to write keys or any data directly to memory thus compromising the very foundation of security in such a system. Trojan circuits can also be set up to scan for electromagnetic signals or software signals (Adee, 2008) so that a processor can be shutdown when provided the right external cue; for deployed embedded systems, such an attack could be disastrous.

Our work focuses on two particular consequences of a Trojan circuit attack:

- *Information leakage*. A simple leak of data can be used to launch subsequent attacks or harvest confidential information. Information leakage is also a means for reverse engineering: the code for an important algorithm that is otherwise encrypted can be leaked in its entirety.
- *Denial-of-Service (DoS)*. A Trojan circuit can be activated at some interval after deployment and can completely shut down the IC, denying service to its device. Such an unexplained ''malfunction'' that occurs long after deployment is advantageous to a competitor that colludes with the foundry to engineer such an attack.

King et al. (2008) demonstrate Trojan circuits that enable a variety of other attacks, however we currently limit the scope of our work to protecting against information leakage and DoS. This limitation allows us to provide a system that, using both encryption and a heartbeat algorithm, can reduce the impact that an inserted Trojan circuit might have on a deployed IC. SHADE prevents information leakage to memory locations, but the cost of dual-encryption is that peripherals on the system bus no longer work properly. Peripherals can be supported at the cost of no longer preventing information leakage, by supplying a dual-decryption module on the peripheral. An area for future research is in supporting peripherals while preventing information leakage.

Currently, the few research efforts aimed at the Trojan circuit problem all appear to focus on detecting such circuits immediately after fabrication. Detection efforts use either functional testing or the physical characteristics of circuits, such as current or delay fingerprints (Agrawal et al., 2007; Li and Lach, 2008; Wang et al., 2008). However, these approaches are inherently statistical and have only been demonstrated on small circuits. Thus, it is fair to say that we are very far from being able to detect Trojan circuits reliably in large chips; indeed, several research efforts addressing this problem are underway (Adee, 2008).

Even if it was possible to detect Trojan circuits reliably, it is expensive both to test chips in large enough samples for statistical significance, and to produce ''golden'' (known correct) chips against which the manufactured chips will be tested. In contrast, we ask the question: if systems are to be built with possibly Trojan-infected chips, to what extent can we guarantee secure execution and attack detection at the time the Trojan circuit activates?

We present the Secure Heartbeat And Dual-Encryption (SHADE) architecture, a compiler–hardware system that assumes untrusted ICs will be used and yet has the ability to execute applications in secure mode by preventing and detecting some types of Trojan circuit attacks. In particular, SHADE detects information leakage and DoS attacks, and also prevents attacks that attempt to write confidential information out to memory.

In the SHADE system, memory accesses are doubly encrypted in hardware, and the compiler back-end inserts non-cacheable memory accesses to create a heartbeat that is checked to detect a DoS attack. In SHADE's current implementation, to check the heartbeats one layer of the dual-encryption must be deterministic so that the compiler can generate heartbeat verification. Naturally, this dual-encryption and heartbeat-checking incur an overhead – one of the goals of this paper is to evaluate such overhead. Using the SimpleScalar simulator (Austin et al., 2001), overheads are shown to vary across a variety of benchmarks, sometimes as low as 4.5% and, at other times, as high as 70%.

The hardware aspect of SHADE is implemented by adding two additional logic modules to the system board to support encryption and heartbeat verification. We also make a non-collusion assumption that two different foundries are used for the two modules. Then, a full decryption (for information leakage) can occur only if both modules collude. We assume the engineers and processes directly employed by the chip's designer and by the board's developer are trusted, so that the board is assembled at a trusted location under the control of either the designer or developer.

Compilation tools and the compilation process need to be instrumented to provide the encryption of code and static data, and also to generate the heartbeats. Compilation is also explicitly trusted and must be performed by a trusted machine. Encryption is relatively simple to add, as it is performed after the code is generated. The heartbeat generation requires making additions in the compiler back-end, which we discuss in more detail in Section 3. As currently conceived, heartbeats use non-cacheable memory writes which may be a design limitation for some systems.

The rest of this paper is organized as follows: Section 2 provides more background on the Trojan circuit problem and related work; the SHADE architectural design is presented in Section 3 and its performance and effectiveness are evaluated in Section 4. Conclusions are drawn in Section 5, where we also discuss possible future work.

2. Related work

Some of the earliest publications on the consequences of semiconductor industry stratification include reports from US Senator Lieberman (2003) and from Defense Science Board (2005). The term Trojan circuit itself appears to have been first used in military circles, and later by DARPA (Adee, 2008). A more recent report from the semiconductor industry (Innovation at Risk, 2008) alludes to this problem and the broader implications of industry stratification, including theft of intellectual property.

The past few years have seen a flurry of research activity aimed at demonstrating attacks or detecting circuits through subjecting chips to certain kinds of physical tests. King et al. (2008) demonstrate the ‘‘Illinois Malicious Processors’’, which are capable of bootstrapping complex attacks based on two Trojan circuits: one that provides shadow execution (cycle stealing) and the other corrupts the memory controller. Simpler forms of attacks have been shown by Agrawal et al. (2007), who tested the effectiveness of their detection scheme by implementing Trojan circuits using a 16-bit counter, an 8-bit sequential comparator, and a 3-bit combinational comparator.

The current trend in defending against Trojan circuits is to detect the added circuitry in a laboratory setting prior to deploying the final product. Much of this work centers on the idea of using physical measures, such as leakage current or delays, to characterize chips. In most cases, such efforts assume that the characteristics are known for ‘‘golden’’ chips fabricated securely and assumed to be Trojan-free. Then, a Trojan circuit leaves a physical fingerprint (additional leakage current, for example) that can be detected in a chip made by the suspect foundry.

Wang et al. (2008) identify three general approaches to Trojan detection: failure-analysis hardware verification, automatic test pattern generation (ATPG), and side-channel analysis. Failure-analysis based techniques are identified as increasingly less useful due to cost in resources and time. The authors point out that although ATPG and other logic testing based detection techniques can work well against malicious alterations of circuitry, such techniques have difficulty in detecting malicious additions. Side-channel analysis is identified as an effective method for detecting such added circuitry. However, due to normal physical variations, side-channel analysis is probabilistic for small fingerprints: the smaller the added circuitry, the more its fingerprint will appear to be normal noise. SHADE is designed for detecting Trojan circuits that survive to full deployment and, as such, is complementary and compatible with these pre-deployment (‘time-zero’) detection schemes.

ATPG (or logic testing based detection) can be effective when searching is directed toward areas of the chip that are susceptible to Trojan insertion. Wolff et al. (2008) analyze the Trojan circuit problem and describe some methods for generating input triggers for Trojan circuits. Chakraborty et al. (2008) present similar work in a design methodology intended to produce conditions that are highly likely to trigger Trojan circuit activation, with the added benefit of obfuscating the original design.

Side-channel analysis techniques often rely on Physical Unclonable Functions (PUFs) (Gassend et al., 2002), which uniquely identify complex ICs by physical characteristics and are used to produce fingerprints that rely on non-functional attributes (side channels). These fingerprints are used to determine challenge-response protocols that rely on the physical characteristics of the IC to provide for authentication. Agrawal et al. (2007), in their seminal paper on this general approach, use power analysis in their detection scheme and propose to investigate other side-channel signals. Rad et al. (2008) extend power analysis to multiple power port inputs across a chip, creating a more scalable, distributed solution. Some other common PUFs include temperature (heat dissipation) and delay characteristics. Li and Lach (2008), for example, propose using both delays measured across combinational circuits (gates) as well as temperature variations.

Di and Smith (Di and Smith, 2007; Smith and Di, 2007) consider the related problem of insider attacks on the design side of the industry. In this case, a rogue employee of a design company can insert a Trojan circuit into the design, either in the high-level synthesis phase (into the netlist) or at the place-and-route stage (into the geometry). Their approach attempts to verify the integrity of design files by reverse engineering and comparison to known structural units.

Another loosely related area consists of securing core boundaries when multiple processing units are co-located on a single chip. Research by Huffmire et al. (2007) describes a simple isolation primitive (moat) that can be used to ensure separation of cores after the place-and-route stage of chip manufacturing. Moats can be used to isolate functionally independent pieces of the design, which provide a framework for designing secure systems out of untrusted components. To then enable communication between the disparate cores, a shared memory bus (drawbridge) is presented. Their work is designed to prevent covert channels within a chip, and their target architecture is FPGA (field programmable gate array) technologies. Using the idea of moats and drawbridges enables implementing SHADE on a single FPGA.

Our work, in contrast to existing techniques, is aimed at detecting the actions of Trojan circuits that survive detection through to deployment; as such it is complementary and compatible with the variety of pre-deployment detection schemes described above. Note that physical characteristics can be used on-line by designing circuits that measure and report these characteristics (Wang et al., 2008). Similarly the age-old fault-tolerant technique of replicating entire processing elements can help to detect some types of Trojan circuit attacks (King et al., 2008). Our approach is complementary to these runtime techniques and can be used simultaneously. Note that our approach is also suited to vendors of board-level products, who may or may not believe the security claims of chip vendors.

A recent trend in hardware security is the use of tamper-proof technology and encryption to provide increased software security. Some examples include special-purpose hardware for cryptographic computations (Dandalis et al., 2000; Grembowski et al., 2002), encrypted execution and data (EED) platforms (Gelbart et al., 2005; Lie et al., 2000; Suh et al., 2003), secure co-processors (Smith and Weingart, 1999; Smith, 1996; Tygar and Yee, 1993; White and Comfort, 1987; Yee and Tygar, 1995), and hardware for security primitives and authentication (Trusted Computing Group; Witchel et al., 2002; Zambreno et al., 2005). When combined with appropriate tamper-proof technology, these techniques are also useful for providing security versus physical attackers that capture a device. While these solutions provide added security, they are not designed to defend against Trojan circuits.

Although we do not explicitly consider physical attackers, our system shares some similarities with EED platforms. In EED platforms, memory is stored in encrypted form and only decrypted within the processor (inner guard) boundaries. The goal of EED is to prevent physical attackers from gaining easy access to application secrets through data or code manipulation. In general, the CPU core is presumed to be tamper-resistant, so that a physical attacker is relegated to activities along the system bus, memory, and I/O devices. With respect to application code the attacker can attempt control flow attacks by replaying observed instruction memory fetches or by injecting new instructions. For the application data, the attacker can attempt data attacks such as data injection, data substitution, and data replay. Code and data injection are made more difficult by the encryption, but are still feasible; the other attacks require additional security mechanisms. EED platforms are designed to resist such attacks through the use of memory freshness indicators, data integrity checks, and control flow verification (Gelbart et al., 2005, 2008; Suh et al., 2003). However, if the attacker has planted a Trojan circuit within the CPU core then EED will provide little extra security: the Trojan circuit can leak the platform's encryption keys, leading to compromised security. Combining EED with the SHADE model may create a platform that is resilient to physical attacks aided by Trojan circuits, which we currently leave as future work.

Finally, we note that some concepts in SHADE were initially outlined in prior work by some of the authors (Simha et al., 2006), without much detail and without experimental results. In this paper, we flesh out system details and evaluate performance through a cycle-accurate simulation.

3. SHADE architecture

SHADE has two principal components – an *architecture* and a specially instrumented *compiler* – and its security rests on two core assumptions. The architecture itself consists of two processing elements, an *inner guard* and an *outer guard* that are set up to adversarially check each other for correctness and responsiveness. As shown in Fig. 1, both guards sit in sequence between the CPU and the system bus, able to check all data moving in or out of the CPU. Both guards are loaded with separate secret keys, the *Inner-key* and *Outer-key* respectively. The trusted compiler is responsible both for preparing applications by encrypting executables once with each key, and for inserting the heartbeats that detect attacks; heartbeats, which are periodic signals to indicate liveness, are presented in Section 3.3. The compilation process is not executed by the system that is instrumented with SHADE but is instead part of assembling the board, which we discuss after explaining the guards and the two core assumptions.

The two guards are just logic elements which can be built from application-specific integrated circuits (ASICs), from re-programmable logic such as FPGAs, or from some co-processing components. For example, the guards can be built using FPGAs (such as the Xilinx Virtex II Pro) with Moats and Drawbridges (Huffmire et al., 2007) to provide isolation. However, other configurations are certainly feasible, and might provide greater generality, extensibility, and performance. In our experiments, we model the architecture design using separate FPGAs for the two guards, which is simpler but less efficient than some other designs.

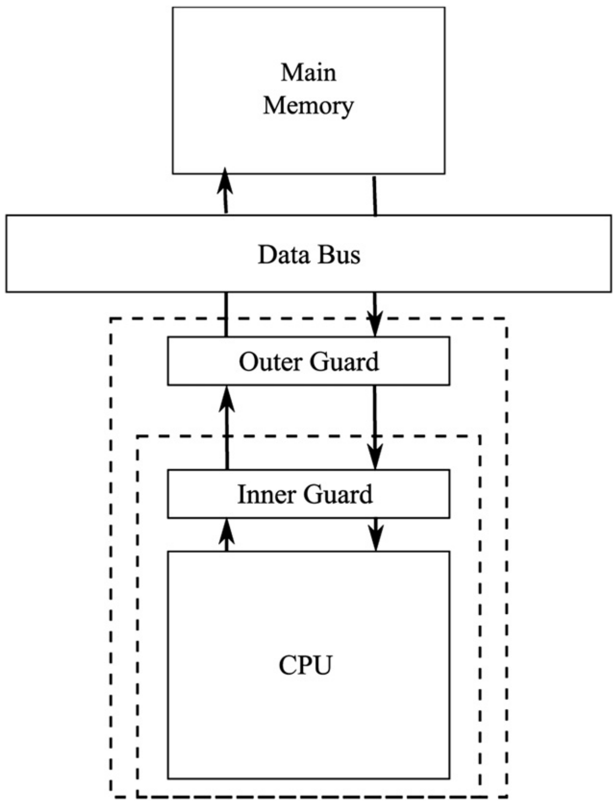


Fig. 1 – Two Guard Architecture. Note that the inner guard can be part of the CPU, for example in an encrypted execution and data (EED) platform, where all memory accesses are encrypted and a cryptographic co-processor is integrated with the CPU. Using programmable logic with moats and drawbridges, the outer guard can also be integrated with the CPU.

Of the two core assumptions, the first is that even if chips are fabricated at untrusted foundries, the resulting board is built in a trusted location. The second is a *non-collusion* assumption: the two guards are fabricated in different foundries and any added Trojan circuitry in one does not collude with the other. These assumptions are discussed and justified below.

At a high-level, our approach would be used as follows. System designers will incorporate our architectural modifications in designing a board, and will order the two guards from different foundries to prevent collusion. Once the potentially untrusted ICs arrive, the system (board) is constructed and a trusted compiler is used to generate dual-encrypted executables for the board. Because the prevention mechanism works at all times, the system can be deployed with real applications, as long as those applications are dually-encrypted by the trusted compiler.

The cornerstone of our approach is the manner in which the execution flow is controlled: the CPU and inner guard are shielded from the bus by the outer guard. All communications into and out of the inner guard's pins have to go through the outer guard logic, which acts as a highly localized firewall to ensure that a compromised IC only leaks useless information or that an attack is detected rapidly. In a nutshell, information leakage is prevented because no single component knows both the inner-key and outer-key, and denial-of-service is detected because the compiler-instrumented code ensures that heartbeat signals are regularly sent and expected by other components. The remainder of this section describes the details of our approach; but first we evaluate the reasonableness of our assumptions.

The compiler instrumentation for information leakage prevention is rather straightforward – once the Inner and Outer-keys are known (and these can vary by individual boards), all the compiler has to do is to encrypt the executable as part of the last phase in compilation (after code optimization) once with the Inner-key and once with the Outer-key. There are some small details such as the appropriate block size for encryption, but these will eventually be determined by fine-tuning. The instrumentation needed for heartbeats involves more compiler-level detail, as we will see, but once optimized the compiler can be used henceforth for all applications.

3.1. *Assumptions*

The effectiveness of SHADE, as mentioned earlier, relies on two key assumptions: (1) compilation and board construction are performed in a trusted location and (2) the two guards are non-colluding.

The first assumption is reasonable because either both board construction and compilation occur in a trusted location (by the consumer), or board construction occurs in one place and board designers delegate authority using a trusted tool chain.

Now consider the second assumption of non-collusion. Note that by non-collusion we mean that even though both guards can have Trojan circuits, the Trojans in each cannot cooperate at runtime through a protocol. Board designers customarily use chips from multiple foundries already, for example a CPU from one foundry, a memory chip from another, and so on. Thus, it would not add much of a burden, beyond maintaining physical compatibility, to obtain the two guards from two different foundries.

Even in the case of using a single foundry, one or both guards could be built out of reconfigurable logic and programmed post-fabrication. Collusion between the two guards for the purpose of defeating the dual-encryption or heartbeats requires a fair degree of sophistication and is all but impossible to guess if the fabric is reconfigurable. Using the concept of moats and drawbridges (Huffmire et al., 2007) also enables instrumenting the system on a single reconfigurable chip, where the CPU, inner guard, and outer guard each are isolated by moats, and drawbridges are used to control access to off-chip resources.

Key management is an interesting problem in its own right. Note that the two keys can either be pre-loaded (already in the design, and obfuscated) or, for more security, can be loaded from ROM as part of the boot procedure. In the latter case, using reconfigurable logic has an advantage: the keys and cryptographic logic are all loaded post-foundry (i.e. in a trusted location).

Assumptions made about the attacker's capabilities are also relevant. We consider two possibly distinct attackers acting in concert: the first (A_1) adds the Trojan circuit to the hardware, and the second (A_2) uses the Trojan circuit to enable an attack. A_1 is a sophisticated attacker that compromises the untrusted foundry, either physically or by subverting the software chain used in the foundry. A_2 will typically be less sophisticated, but in some instances may possess advanced equipment to launch successful attacks (e.g., knowing the trigger to cause information leakage).

Heartbeat detection becomes complicated in a general time-sharing environment, where interrupts (and exceptions) cause non-determinism which cannot be completely anticipated ahead of time. We have not yet solved this problem, but we suggest that the operating system would need to be trusted to store heartbeats with the state of each task during context switching. This involves adding the operating system to the trusted computing base, which is not necessary in the more restricted scenario we examine in this paper.

3.2. *Details of dual-encryption and leakage protection*

There are many ways in which a Trojan circuit can leak information. One of the most straightforward ways is to write leaked information to memory, or to dump processor contents onto the bus. In this manner, information is available to accompanying software that is able to read the memory or to external probes that can sniff the bus. If we can ensure that everything written by the processor is encrypted, then any leakage from a Trojan circuit will consist only of encrypted information, presumed useless to an attacker. Even if the encryption of the processor is subverted, the outer guard will correctly encrypt the data and so the attacker is always left with encrypted data.

The purpose of dual-encryption is to ensure that if a Trojan circuit leaks information it is encrypted by the time it reaches memory or even the bus. Note that a suitable (i.e. cryptographically strong) cryptographic function must be chosen that supports encrypting twice without degrading security; such encryption is already routinely performed in practice by, for example, onion routing techniques. The inner and outer encryption need not be the same function, the choice of cryptographic function is left to the implementer.

In the approach we propose, when the board is assembled, the inner and outer guards are loaded with the Inner-key and Outer-key respectively. Secure applications are encrypted by the compiler using a trusted system; all code and data are encrypted first with the Inner-key, then with the Outer-key. Compilation may need to be replicated in case Inner- and Outer-keys differ between assembled boards. The encrypted binaries are placed in storage from which they will ultimately be loaded by the CPU. Although the operating system is also dual-encrypted, the BIOS can find the OS boot code in storage. The hardware performs the decryption, so additional configuration is unnecessary to get the system booted. Fig. 2 shows where we modify the compiler to support such encryption – at the end of the compilation process.

Fig. 3 shows the process of how memory accesses are encrypted (decrypted) on the write (read) path. During execution, every memory access goes through the two guards. On every memory load, the outer guard first decrypts with the Outer-key then forwards the result to the inner guard, which

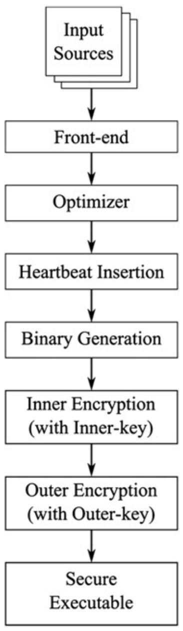


Fig. 2 – Compiler flow in SHADE.

decrypts with the Inner-key. Every store to memory follows the reverse process: the inner guard first encrypts then forwards the encrypted data to the outer guard, which encrypts and completes the store.

Dual-encryption prevents information leakage attacks from succeeding. On one hand, if the CPU or inner guard attempts to leak anything, the outer guard will encrypt the data and so the attack will fail. On the other hand, the outer guard only ever receives data that are encrypted by the inner guard; the outer guard could still leak some internal data (for example, configuration variables, internal registers, or the Outer-key), but using an FPGA frustrates the ability of a Trojan circuit to predict where the internal data reside. Thus, each guard could have Trojan circuits that leak, but to be successful requires collusion between those circuits across both chips, a scenario that is made highly unlikely by our requirement that the chips are made by different foundries or are built out of reconfigurable logic.

Note that although dual-encryption prevents information from being leaked, it does not detect the moment such leakage occurs. Thus, for example, an unencrypted write to memory from the inner guard will merely result in improper values (garbage) being written during a leak. To detect such an attack

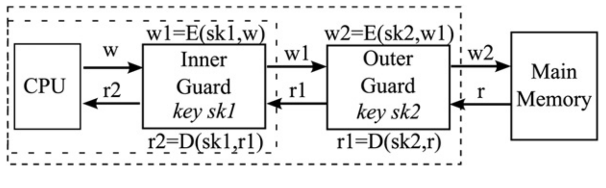


Fig. 3 – Flow of dually-encrypted memory reads and writes with Inner-key *sk1* and Outer-key *sk2*. All of main memory is encrypted (the trusted compiler encrypts static data and code).

from the inner guard, one of many standard techniques from network security can be adapted here if we were to treat the two guards as two nodes in a network. For example, communication between the two guards can be made to include signed message digests. This approach of course involves additional overhead beyond that already incurred by the dual-encryption, but it is easily generalizable to communication between any pair of chips – such authenticated communication could also be used for the writes from the outer guard to memory, in which case the memory unit has to be outfitted with logic for secure communication.

When the outer guard is reconfigurable and programmed post-foundry, an attacker would find it difficult to create a Trojan circuit that is capable of leaking anything other than the configuration file. An alternative is for detection to be performed in software, with an operating system module that periodically checks memory for improperly encrypted data.

3.3. *Details of heartbeats and DoS detection*

Recall that one of the simplest actions a Trojan circuit can take is to disable a chip, a denial-of-service (DoS) attack. In some ways, a DoS scenario is diametrically opposite to that of information leakage: there is no practical way to prevent it if the Trojan circuit survives testing, but detecting the attack at runtime is both possible and useful.

We introduce a heartbeat that enables the guards to check each other (and the CPU) for continuing execution. The heartbeat is created by the compiler, which inserts memory writes to addresses located in a non-cacheable region (similar to memory-mapped instructions). These memory stores are used as signals to the outer guard, indicating a time range until the next heartbeat. If a heartbeat is not sent within the time range indicated by the previous heartbeat, then the outer guard detects an attack.

In order to make the timing deterministic, a heartbeat is added to every extended basic block (contiguous block of instructions without a jump). The heartbeat indicates to the outer guard the number of cycles of the current basic block. When the next basic block executes, the outer guard will be expecting the next heartbeat. For the outer guard to recognize the heartbeats, the compiler prepares a *heartbeat table* which associates the memory stores, encrypted with the inner-key, with a range of cycles containing the delay until the next heartbeat. As part of initial configuration, the outer guard is either set up with the heartbeat table or with the ability to fetch the table from memory. The outer guard checks every memory write to the non-cacheable region against the heartbeat table to see if the write is a heartbeat. This heartbeat table is small, is stored only on the outer guard, and is checked in parallel to the critical path of the memory fetching pipeline. Each heartbeat sets a countdown timer to the duration indicated by the table; if the timer reaches zero before the next heartbeat, the outer guard raises a signal to indicate that an attack has happened.

Note that these heartbeats require either that the inner encryption provides the same encryption for each identical memory write, such as in block cipher electronic codebook (ECB) mode, or that the compiler knows the initialization vectors and is able to predict the runtime value of encrypted heartbeats. Also, the heartbeat indicates an expected cycle delay for the current basic block, but non-deterministic factors (e.g. cache misses) make precise timing difficult. Thus the heartbeat should either be the worst-case delay, a range between best and worst-case, or be aware of runtime changes that impact execution time.

Fig. 4 shows an illustrated, if contrived, example of some C code, the compiler generated heartbeat table, and the runtime behavior for that code. In this example we suppose that the exit function does not produce a heartbeat.

Heartbeats are designed to detect DoS attacks; they also provide some coincidental information leakage detection. If the inner guard does not properly encrypt the heartbeat then the outer guard will not observe the heartbeat and will detect that an attack has occurred. Information leakage detection is incomplete because an information leakage write could occur between consecutive heartbeats. That is, the Trojan circuit would have to be sophisticated or random enough that it turns on encryption when the heartbeat occurs.

Although detecting transient leakages is not currently solved, consistent leakages are detected. A possible approach to improving leakage detection is to profile code for deterministic memory writes and instruct the outer guard to watch for them. However, the inner guard's encryption will render the deterministic writes unrecognizable unless the entire cache block is deterministic. Also, due to caching, the outer guard cannot reliably know when to expect the writes to occur; thus the deterministic writes are not easily used as heartbeats.

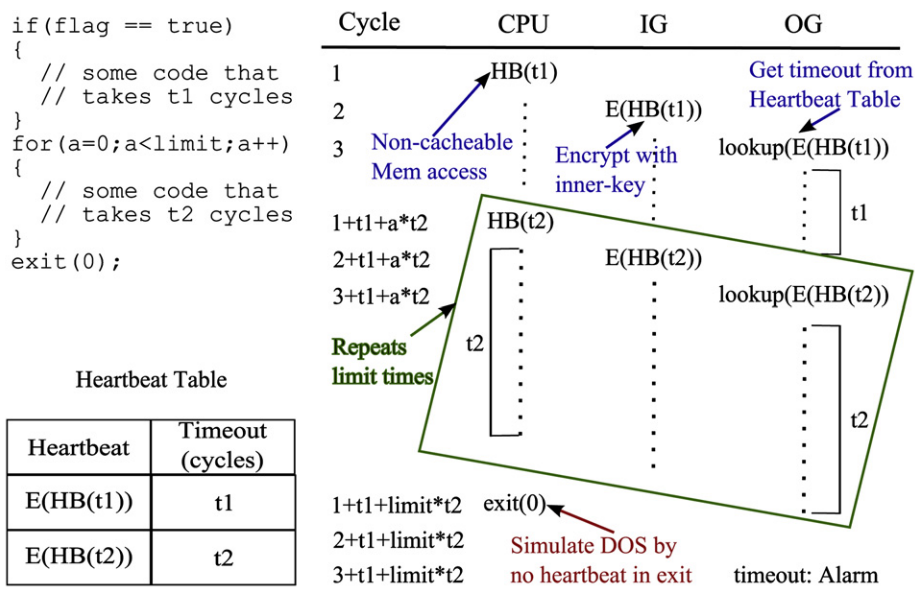
Our current heartbeat solution imposes a number of limitations to the general applicability of the SHADE system. Non-cacheable memory regions must be designated for the heartbeats and must be known at compile time. Interrupts and task switching require extending the operating system to support managing and swapping the heartbeat tables as part of the context switch. Because the outer guard needs to match encrypted data coming from the inner guard against the contents of the heartbeat table, the inner guard's encryption must be deterministic and computable by the compiler. Finally, adding heartbeats to every extended basic block introduces overhead, which we examine in the next section.

4. Evaluation

The compiler-inserted heartbeats provide detection of a variety of attacks, including information leakage, DoS, and any attack that causes enough timing perturbation to delay the heartbeats. Two layers of encryption protect against successful information leakage. We measure the overhead introduced by our scheme through cycle-accurate simulation and a suite of benchmarks designed for embedded systems. We also discuss SHADE's effectiveness at preventing or detecting attacks identified in Section 1.

4.1. *Performance overhead*

We simulated SHADE using SimpleScalar (Austin et al., 2001) and the gcc cross-compiler for the ARM processor. Recall that we model the two guards as distinct FPGAs, a less efficient but simpler design choice for implementing the SHADE



```
if(flag == true)
{
  // some code that
  // takes t1 cycles
}
for(a=0;a<limit;a++)
{
  // some code that
  // takes t2 cycles
}
exit(0);
```

Heartbeat Table

Heartbeat	Timeout (cycles)
E(HB(t1))	t1
E(HB(t2))	t2

Fig. 4 – Heartbeat example (IG = inner guard, OG = outer guard). For this simple code, the basic block inside the taken if-branch consumes t1 cycles, and inside the for loop consumes t2 cycles on each iteration. The compiler inserts a non-cached memory access at the start of each basic block, encrypts the memory access with the inner-key, and then stores the encrypted value with the number of cycles of the basic block in to the heartbeat table. The number of cycles of the basic block is used as a timeout to detect attacks. The CPU issues the non-cached memory writes when the basic blocks begin, the inner guard encrypts (with the inner-key), and the outer guard observes a match in the heartbeat table. If no heartbeat is matched by the outer guard before the last timeout expires, then an alarm is raised to indicate a failure has occurred. In this example, we assume exit does not generate a heartbeat (i.e. it simulates a DoS attack).

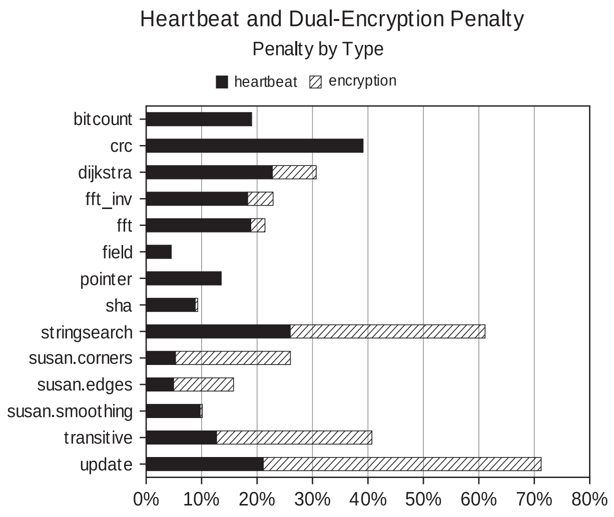


Fig. 5 – SHADE overhead.

architecture. We modified the sim-outorder simulator to add FPGA modules for the guards, and the compiler was modified to add the heartbeats. The processor was chosen to represent a typical embedded system: a 400 MHz processor with two 200 MHz FPGAs (for the guards), an external bus and memory running at 100 MHz, and one level of instruction and data caches, each 32 KB in size, 32-way associative, with 32-byte cache lines. We evaluated the system's performance using the MiBench (Guthaus et al., 2001) and Data Intensive Systems (DIS) (Data Intensive Systems Benchmark Suite) benchmarks for embedded processors.

For encryption, we used an AES implementation in Electronic Codebook (ECB) mode. Although ECB introduces the possibility of replay attacks, the point of dual-encryption is to protect sensitive data from leaving the system. One possible weakness is that the inner guard could replay heartbeats, if it can find out the heartbeat timing. Although such an attack is difficult, it is possible; however, the inner guard receives unencrypted memory requests, so the choice of encryption algorithm does not help or hinder that situation. External attackers can only replay memory accesses that are observed, but are unable to discern the encrypted data. Patterns of traffic are vulnerable to analysis, but ECB mode is sufficiently secure for the purposes of Trojan circuit detection. More secure encryption schemes can be used if further guarantees are needed, and even a hybrid approach might be sufficient (for example use ECB at the inner guard and AES CBC at the outer guard).

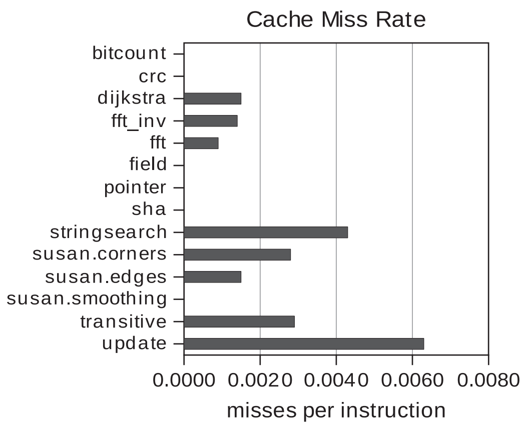
Fig. 5 shows the overhead of both heartbeats and encryption, compared against the standard sim-outorder simulator with unmodified gcc. The average overhead is 15.3% for the heartbeats and 11.4% for the encryption, for a combined (average) overhead of 26.7%. The combined overhead minimum and maximum are 4.5% and 69.9% respectively.

Execution performance is impacted by both the added overhead for encrypting and the extra instructions for heartbeats. Encryption occurs along the memory path and thus, not unexpectedly, the benchmarks with larger cache miss rates exhibit larger encryption overhead. Fig. 6 shows the cache miss rates of the benchmarks without the added heartbeats, which would change instruction cache miss rates. Note that although cache miss rate tells part of the story, the delay between cache misses is also important: a full pipeline of memory accesses and decryptions hides some of the overhead.

Heartbeats expand the code size, so benchmarks that tend to execute smaller basic blocks have more frequent heartbeats and therefore greater overhead. Fig. 6 shows the frequency of heartbeats for the benchmark applications, based on profiled execution. As expected, the overhead introduced by heartbeats is directly proportional to the frequency of heartbeat execution. The average number of heartbeats per 100 instructions is 8.5.

4.2. *Security evaluation*

To detect information leakage attacks, the outer guard implicitly verifies that the heartbeats are properly encrypted (if they are not, the outer guard will not make a match in the heartbeat table). DoS and attacks that substantially delay normal program execution are detected trivially due to delayed or missing heartbeats.



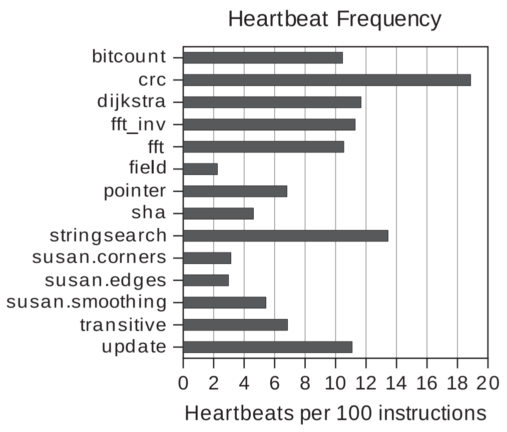


Fig. 6 – Cache miss rates and frequency of heartbeats (basic blocks) per 100 instructions for the unmodified benchmarks. Higher cache miss rates increase overhead caused by encryption, and greater frequency of heartbeats with respect to instruction count increases the overhead caused by heartbeats.

There is no mechanism to detect if the outer guard leaks information (internal configuration, outer-key, etc). However, by our assumption that the two guards do not collude, and the relative simplicity of the outer guard, such an information leak is less critical to the security of our system. A simple fix to this concern is to add more guards (more redundant checks), but this also will increase the memory path latency.

SHADE is unable to prevent most attacks from occurring, but is designed to detect anomalous behavior indicating an attack has happened. Attacks are detected by missing or corrupted heartbeats. Thus, the time between an attack and its detection relies on the distance between heartbeats, which is the basic block length. With the benchmark applications used for testing performance overhead, the basic block length ranges between 2 and 842 instructions (including the two instructions used for simulating the heartbeats), with an overall average of approximately 11 instructions per basic block. At runtime, the number of instructions executed between heartbeats averages between 5.3 and 44.5, with an overall average of 17 instructions between heartbeats.

5. Conclusion and future work

In this paper we presented the SHADE architecture for detecting and preventing Trojan circuit attacks in deployed ICs fabricated at untrusted foundries. Our goal was to provide an approach to address attacks launched by Trojan circuits during deployment. Our approach has several advantages. As a leakage prevention technique it is as effective as the cryptographic strength of keys used and, therefore, very secure. In addition, when the additional hardware we use is reconfigurable, the encryption keys are loaded post-foundry as part of a larger circuit in a trusted location. In terms of detection, our approach is non-destructive and compatible with all pre-deployment approaches that are situated across the design spectrum from system specification through packaging.

Our performance study indicates that heartbeats represent a fairly substantial overhead, and so in the future, we aim to investigate compiler optimization techniques to reduce this overhead. Some other areas to improve performance include memory pre-fetching by the outer guard and integrating the inner guard with the CPU. If the inner guard is in the CPU, then its encryption can be more efficient, which will be most helpful if the outer guard can offset its own encryption overhead by pre-fetching memory blocks for decryption. An added benefit to placing the inner guard in the CPU is the possibility of using the inner guard as the cryptographic co-processor when integrating SHADE with Encrypted Execution and Data (EED) platforms (Gelbart et al., 2005; Lie et al., 2000), where instructions and data are encrypted in memory.

An interesting challenge not solved by SHADE is if the Trojan circuit causes a subtle change to disrupt software correctness that does not impact the CPU timing mechanism. For example, a Trojan circuit might be designed to disable the carry-bit of an adder given some rare environment (such as specific register contents being equal to a given account number). Assuming that the result is not being used to encrypt the heartbeat, SHADE will not detect such an attack because there is no noticeable change to the CPU timing. Defending against such attacks is an area for future research.

The scope of information leakage protection provided by SHADE is currently limited to attacks against the memory data flow. The current SHADE system breaks memory-mapped IO and DMA transfers from memory to device. Similarly, transfers to memory will not be dual-encrypted and therefore will not be properly read by the CPU. SHADE is sufficient for disabling information leakage attacks against non-networked devices, but supporting networked devices (and more generally any device with an external peripheral, such as a serial port) provide a much greater challenge, another area for future research.

Acknowledgement

This research is partially supported by NSF grant ITR-025207, NSF grant CNS-0934725, and AFOSR grant FA955006-1-0152.

REFERENCES

Adee S. The hunt for the kill switch. IEEE Spectrum May 2008.

Agrawal D, Baktir S, Karakoyunlu D, Rohatgi P, Sunar B. Trojan detection using IC fingerprinting. In: Proceedings of IEEE Symposium on Security and Privacy; May 2007. p. 296–310.

Austin T, Larson E, Ernst D. SimpleScalar: an infrastructure for computer system modelling. IEEE Computer Society; February 2001. p. 59–67.

Chakraborty RS, Paul S, Bhunia S. On-demand transparency for improving hardware Trojan detectability. IEEE International Workshop on Hardware-Oriented Security and Trust (HOST); June 2008. p. 48–50.

Dandalis A, Prasanna V, Rolim J. An adaptive cryptographic engine for IPSEC architectures. In: Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM); 2000. p. 132–44.

Data Intensive Systems Benchmark Suite. Atlantic Aerospace Electronics Corporation, http://www.aaec.com/projectwed/dis/.

High Performance Microchip Supply. Defense Science Board; 2005.

Di J, Smith S. A hardware threat modeling concept for trustable integrated circuits. IEEE Region 5 Technical Conference; April 2007.

Gassend B, Clarke D, van Dijk M, Devadas S. Silicon physical random functions. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington, DC; November 2002.

Gelbart O, Ott P, Narahari B, Simha R, Choudhary A, Zambreno J. CODESSEAL: a compiler/FPGA approach to secure applications. In: Proceedings of the IEEE Conference on Intelligence and Security Informatics (ISI); 2005.

Gelbart O, Leontie E, Narahari B, Simha R. Architectural support for securing application data in embedded systems, IEEE International Conference on Electro/Information Technology, 2008. EIT 2008; 18–20 May 2008. p. 19–24.

Grembowski T, Lien R, Gaj K, Nguyen N, Bellows P, Flidr J, et al. Comparative analysis of the hardware implementations of hash functions SHA-1 and SHA-512. In: Proceedings of the International Conference on Information Security (ISC); 2002. p. 75–89.

Guthaus MR, Ringenberg JS, Ernst D, Austin TM, Mudge T, Brown RB. MiBench: a free, commercially representative embedded benchmark suite. In: Proceedings of 4th IEEE Workshop on Workload Characterization; December 2001. p. 10–22.

Huffmire T, Brotherton B, Wang G, Sherwood T, Kastner R, Levin T, et al. Moats and drawbridges: an isolation primitive for reconfigurable hardware based systems. In: Proceedings of the 2007 IEEE Symposium on Security and Privacy SP'07; 2007. p. 281–95.

Innovation at risk: Intellectual Property Challenges and Opportunities, Semiconductor Equipment and Materials Industry (SEMI); 2008.

King ST, Tucek J, Cozzie A, Grier C, Jiang W, Zhou Y. Designing and implementing malicious hardware. In: Proceedings of 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats, San Francisco, California; April 2008.

Li J, Lach J. At-speed delay characterization for IC authentication and Trojan Horse detection. IEEE International Workshop on Hardware-Oriented Security and Trust (HOST); June 2008. p. 8–14.

Lie D, Thekkath C, Mitchell M, Horowitz M. Architectural support for copy and tamper resistant software. In: Proceedings of the International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS); November 2000.

Rad R, Plusquellic J, Tehranipoor M. Sensitivity analysis to hardware Trojans using power supply transient signals. IEEE International Workshop on Hardware-Oriented Security and Trust (HOST); June 2008. p. 3–7.

Simha R, Narahari B, Zambreno J, Choudhary A. Secure execution with components from untrusted foundries. In: Proceedings of Advanced Networking and Communications Hardware Workshop (ANCHOR); June 2006.

Smith S, Di J. Detecting malicious logic through structural checking. IEEE Region 5 Technical Conference; April 2007.

Smith S, Weingart S. Building a high-performance programmable secure coprocessor. Computer Networks 1999;31:831–60.

Smith S. Secure coprocessing applications and research issues. Los Alamos Unclassified Release LA-UR-96-2805; 1996.

Suh GE, Clarke D, Gassend B, van Dijk M, Devadas S. AEGIS: architecture for tamper-evident and tamper-resistant processing. In: Proceedings of the 17th Annual International Conference on Supercomputing, San Francisco, CA, USA; 23–26 June 2003; ICS'03. ACM, New York, NY; 2003. p. 160–71.

Trusted Computing Group, http://www.trustedcomputing.org.

Tygar J, Yee B. Dyad: a system for using physically secure coprocessors. In: Proceedings of the Harvard-MIT Workshop on Protection of Intellectual Property; April 1993.

Lieberman J. Whitepaper on National Security Aspects of the Global Migration of the US Semiconductor Industry, http://lieberman.senate.gov/documents/whitepapers/semiconductor.pdf; June 2003.

Wang X, Tehranipoor M, Plusquellic J. Detecting malicious inclusions in secure hardware: challenges and solutions. IEEE International Workshop on Hardware-Oriented Security and Trust (HOST); June 2008. p. 8–14.

White S, Comfort L. Abyss: a trusted architecture for software protection. In: Proceedings of the IEEE Symposium on Security and Privacy; 1987. p. 38–51.

Witchel E, Cates J, Asanovic K. Mondrian memory protection. In: Proceedings of the International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS); 2002.

Wolff F, Papachristou C, Chakraborty RS, Bhunia S. Towards Trojan-free trusted ICs: problem analysis and a low-overhead detection scheme, DATE Europe; 2008.

Yee B, Tygar J. Secure coprocessors in electronic commerce applications. In: Proceedings of the USENIX Workshop on Electronic Commerce; July 1995. p. 155–70.

Zambreno J, Choudhary A, Narahari B, Memon N, Simha R. SAFE-OPS: a compiler/architecture approach to embedded software security. ACM Transactions on Embedded Computing Systems February 2005;4(1).

Gedare Bloom received the B.S. degree in computer science from Michigan Technological University, Houghton, MI in 2005, and is currently pursuing the Ph.D. degree in computer science at The George Washington University. His research interests include software security, system security, compilers, computer architecture, and distributed systems.

Bhagirath Narahari received the B.E. degree in electrical engineering from the Birla Institute of Technology and Science, Pilani, India in 1982, and the Masters and Ph.D. degrees in Computer Science from the University of Pennsylvania, Philadelphia, in 1984 and 1987 respectively. He is currently a Professor of Computer Science at The George Washington University, where he was the Department Chairman from 1999 to 2002. His research interests lie in the areas of embedded systems, compiler optimization, computer architecture, and software security.

Rahul Simha is Professor of Computer Science at The George Washington University. He received his Ph.D. in Computer Science from the University of Massachusetts in 1990. His research interests include embedded systems, compilers, languages, architecture, security and complex systems.

Joseph Zambreno is an Assistant Professor in the Department of Electrical and Computer Engineering at Iowa State University, Ames, where he has been since 2006. He received the B.S. degree (Hons.) in computer engineering in 2001, the M.S. degree in electrical and computer engineering in 2002, and the Ph.D. degree in electrical and computer engineering from Northwestern University, Evanston, IL, in 2006. His research interests include computer architecture, compilers, embedded systems, and hardware/software co-design, with a focus on runtime reconfigurable architectures and compiler techniques for software protection.