On Implementing Trusted Boot for Embedded Systems

Obaid Khalid*, Carsten Rolfes*, Andreas Ibing†
*Fraunhofer Research Institution for Applied and Integrated Security (AISEC), Munich, Germany
†Chair for IT Security, Technische Universität München, Munich, Germany

***Abstract*—This paper presents an implementation of trusted boot for embedded systems. While in PCs the trusted computing hardware functionality is spread over CPU, memory controller hub (MCH), IO controller hub (ICH) and Trusted Platform Module (TPM), for embedded systems it is desirable to integrate the whole functionality in one system on chip. Our implementation is a two-processor design with LEON3 open source soft cores (SPARC V8 instruction set), coupled over an AHB interface. One of the processors acts as application processor, the other one as 'secure' coprocessor. The application processor is synthesized with a boot ROM as static root of trust for measurement. The 'secure' coprocessor runs TPM f rmware and enables the application processor to boot and run different software while sealing corresponding keys and other secrets to the respective software identity (computed as hash value). We evaluate the design in a Virtex5 FPGA with respect to different measures like resource consumption, code sizes and start times. The 'trusted boot' functionality is realised with a boot time increase of around 25% for a Linux system.**

I. Introduction

With increasing spread of rootkits and bootkits, the need for some form of system startup security has been widely acknowledged. The required immunity from software attacks necessitates a root of trust which is implemented in hardware and not modif able by software.

Security hardware has been developed in the past mainly as secure co-processors for encryption/decryption and secure key storage. Hardware security modules (HSM) come in different shapes like e.g. smartcards or Trusted Platform Modules (TPM). A survey of secure co-processors is provided in [1]. More recently, a tighter integration with other system components led to the notion of a 'secure application processor'. A survey of this is given e.g. in [2].

For servers and personal computers, the Trusted Computing Group (TCG) released specif cations both of architecture and implementation requirements [3], [4]. This architecture relies on the Unif ed Extensible Firmware Interface (UEFI, [5]) and TPM [6], and has led to extensions in mainstream processors and chipsets (e.g. [7]). On the software side, 'trusted boot' functionality has been implemented e.g. in the Linux integrity subsystem (starting from kernel 2.6.30) and the corresponding Trusted Grub boot loader [8]. In PCs, the hardware functionality for 'trusted boot' is spread over four chips, i.e. CPU, northbridge (memory controller hub, MCH), south bridge (IO controller hub, ICH) and TPM [7].

For embedded systems, integration in one system-on-chip is desirable both for cost and for size reasons. While the Trusted Computing Group is working on this topic (e.g. [9]), there is yet no specif cation available. Current embedded products use a 'secure boot', where processors are equipped with a boot ROM to perform a signature check of the software before boot (e.g. [10]). If verif cation fails, the boot process is stopped. Some operating system providers make 'secure boot' mandatory (e.g.[11]) on embedded architectures like ARM [12] systems.

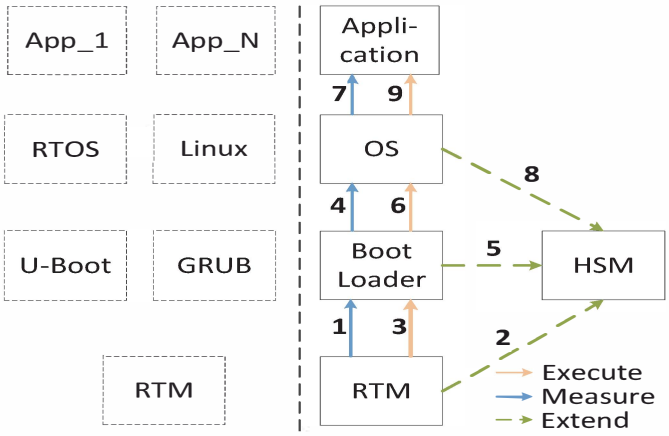

Fig. 1. Chain of trust.

'Trusted boot' and 'secure boot' differ in who makes the trust decision. While 'secure boot' relies on certif cates from certif cate authorities, 'trusted boot' identif es software components by their hash value and offers additional f exibility for the user to make his own trust decision. 'Trusted boot' further allows to boot different software and to 'seal' cryptographic keys and other secrets to the respective software identity. These differences become more interesting with an increasing amount of malware with valid signatures from stolen/misused certif cates, recent examples of which include [13], [14].

Other related work includes [15], [16], [17]. In [15], protection of off-chip memory has been added to a processor using a hash tree. In [16], bus encryption is added to a SPARC v8 processor. [17] describes the addition of a memory integrity tree to the soft core OpenSPARC processor, evaluated in an FPGA.

This paper describes and evaluates an implementation of 'trusted boot' for embedded systems. The design is a single-chip two-processor LEON3 (SPARC v8) system with boot

978-1-4799-0601-7/13/$31.00

ROM, where one LEON3 processor acts as HSM and runs Trusted Computing f rmware.

The remainder of this paper is organized as follows: the functionality of the proposed design is described in Sec. II, the resulting architecture in Sec. III. Sec. IV describes and evaluates in detail the example implementation in a Virtex5 FPGA, with respect to resource consumption by the additional hardware and f rmware, and with respect to startup duration. The results are discussed in Sec. V.

II. Functionality

We implement a minimal functionality of 'trusted boot' for embedded systems. Integrity measurement at start time is based on a 'root of trust for measurement' (RTM), which can be seen as a minimalistic boot loader implemented in hardware. From the RTM, a 'chain of trust' is extended to software components during 'trusted boot'. The identity of a piece of software in cryptographic terms is its hash value. The trust chain consists of f rst computing the hash value of a component before executing it, and of a component hashing the following one (Fig. 1). Software components which are isolated from each other (like different applications by an operating system, or different operating systems by a hypervisor) reside on the same level, while dependent software resides in different levels. The integrity of a software component could be violated from a lower level (whose identity is therefore checked f rst). The chain levels in normal conf guration are therefore: RTM, boot loader, hypervisor (if used), operating system, applications. With different software components on one level, one could also speak of a 'trust tree'. Nodes in the tree are identif ed with a single hash value, which is computed from the corresponding path hash values using a 'hash extension operation' [3].

The HSM is used to perform the hash computations and to securely store the extended hash values. It further offers cryptographic operations to the application processor and generates and stores the corresponding secret keys. Based on the extended hash values, the HSM may grant or stop starting a component on the application processor. It is possible to boot and run different application software, and the HSM can deny cryptographic operations if the corresponding keys do not belong to the requesting software ('sealing').

The minimal functionality we implement does not include remote attestation (providing remote access over the network to check extended hash values). This functionality could be added in a straight forward way if needed.

The selected functionality is intended to provide security against software attacks. It provides only very weak security against hardware attacks (with physical access). The trust boundary is the chip. External (off-chip) application software is booted and identif ed by on-chip functionality.

Since integrity checks are performed at start time, a time of check / time of use (TOCTOU) problem does exist. 'Trusted computing' functionality does not rectify security vulnerabilities in application processor software. Application processor software corruption in RAM is not detectable with these methods. Exploits are only detected if injected malware is written 'to disc'. Still, secret keys cannot be stolen from the application processor side even in case of exploit, as they do not leave the HSM (they can 'only' be used in the HSM).

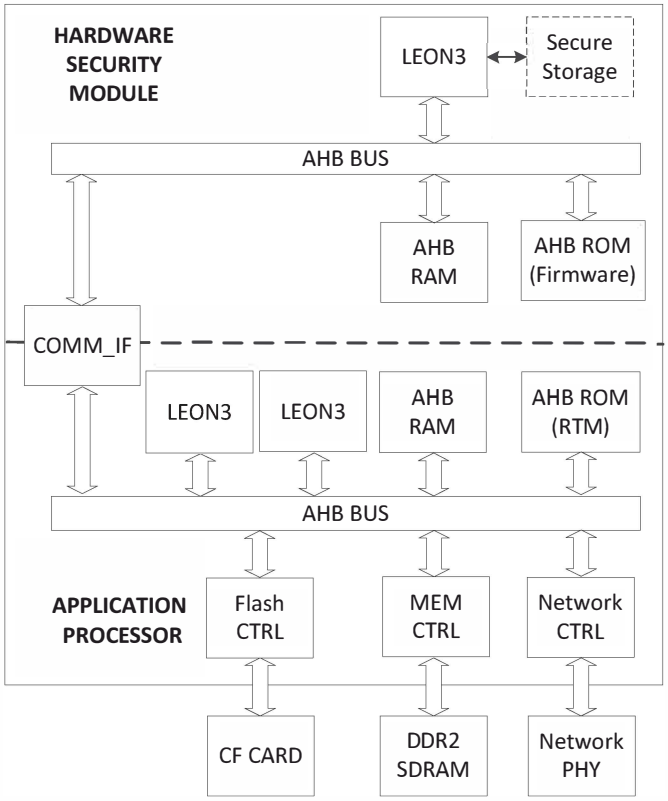

Fig. 2. Hardware Architecture

III. Architecture

The architecture comprises hardware, f rmware and software components which are described in the following subsections.

A. Hardware

The hardware architecture is depicted in Fig. 2.

1) Processors: The open source soft core LEON3 processor [18] is used both for the application processor and for the HSM. It is a highly conf gurable 32 bit processor compliant to the SPARC v8 architecture. For the FPGA evaluation in Sec. IV we synthesize the application processor in dual-core conf guration and the HSM in single-core conf guration. The LEON3 processor has a 7-stage pipeline and implements the AMBA AHB [19] master interface for its cache system. We synthesize the processors with f oating-point units.

2) Buses and Interfaces: The functionality as described in Sec. II necessitates separate address spaces for the two processors, so that each one has its own AHB and APB bus. The application processor further has access to a network interface and f ash controller, and to off-chip DRAM.

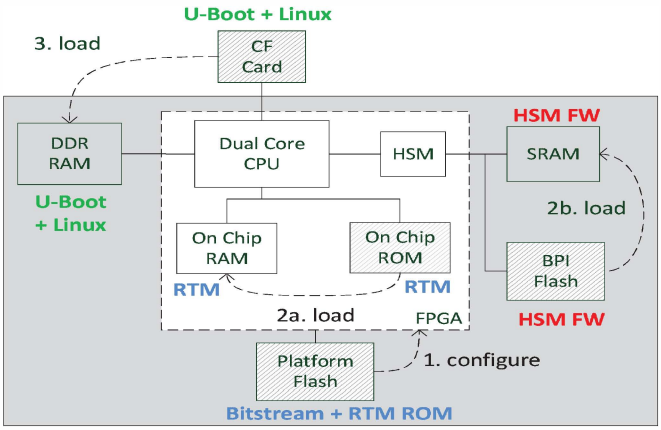

Fig. 3. Mapping of architecture components in the FPGA based evaluation.

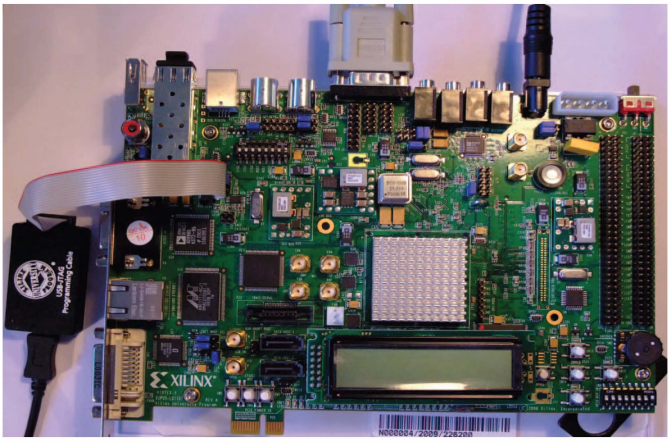

Fig. 4. Test set-up with FPGA development board, with JTAG connection (left) to the application processor debug support unit and serial connection (top) to that of the HSM processor.

a) Communication between application processor and HSM: the two processors communicate over a special communication interface which is connected to both AHBs (denoted as COMM_IF in Fig. 2). The design of this interface is based on the TPM interface from [20]. One of the differences is that we use memory mapped registers instead of port I/O. The interface contains transmit and receive FIFOs, distinct register f les for application processor and HSM, and software interrupts in both directions. The FIFOs have a size of 2KB each to transmit TPM related command and response messages (which is more than the TCG PC client TPM interface specif - cation [21] would require). The register f le for the application processor contains f ve 32-bit registers, the register f le for the HSM contains four. They implement a subset of the f elds from [21].

3) Memory: The separation of address spaces has the consequence that application processor and HSM execute their code from different memory modules. The application processor is connected to an on-chip boot ROM (mapped at its reset vector), which comprises initialization code and the RTM. The application processor executes operating system and applications in an off-chip RAM. The HSM on the other hand uses on-chip non-volatile memory for its f rmware storage and also for secure storage of keys and extended hash values. It executes its f rmware in on-chip RAM.

B. Firmware

The HSM f rmware implements TPM functionality [6]. We use a 'bare metal' implementation without operating system to keep the code size small. The code is based on the TPM emulator code from [22]. For the communication interface to the application processor, the driver of [20] has been adapted.

C. Software

The application processor boots from external storage (in Sec. IV we use Compact Flash cards) or over network. Depending on the HSM f rmware conf guration, any boot loader and operating system can be started. To make use of the trust hierarchy (Sec. II) and the possibility to 'seal' secrets to their identity, they should implement integrity measurements. We run tests in the next section with the U-Boot boot loader and Snapgear Linux (both support LEON3, [23], [24]). U-Boot and Snapgear Linux have been modif ed to include the driver for the communication interface to the HSM, so they can send commands to the HSM to perform SHA-1 measurements of the next piece of code.

D. Boot Process

Application processor and HSM are reset together. Both start executing independently from their on-chip ROMs. After initialization, the HSM signals ready over the communication interface and waits for a command from the application processor. Application processor core 0 starts executing the RTM, while core 1 (and any further cores) is deactivated after reset and is activated by a symmetric multiprocessing (SMP) operating system during kernel startup.

IV. Evaluation in FGPA

A. FPGA and Development Board

We use the XUP5-LX110T development board (Fig. 4) for evaluation, which is a version of the ML505 board [25] and

TABLE I
HSM Firmware Size

Component	Size [byte]	Ratio [%]
Firmware	441,021	100.0
- Command interpreter	206,060	46.72
- GMP library	162,860	36.9
- DAA	33,816	7.67
- TPM Data	12,056	2.73
- Printf	11,048	2.51
- RSA	6,044	1.37
- SHA1	6,913	1.57
- RNG	1,212	0.27
- Comm_IF	376	0.09
- HMAC	336	0.08
- RC4	300	0.07

TABLE II
TIME MEASUREMENTS

With integrity measurement	# of proc. cycles	time (sec)	% of total boot time
System initialization			
- RTM hardware initialization	2224862	(0.03)	-
- HSM hardware initialization	7023631	0.09	0.08
RTM execution			
- HSM f rmware initialization	1434390609	17.93	16.51
- Fetching U-Boot to DRAM	(6449862)	(0.081)	-
- U-Boot integrity measurement	68803197	0.86	0.79
- Misc. RTM operations	10160892	0.13	0.12
U-Boot execution			
- Start U-Boot + Misc.operations	290195973	3.63	3.34
- Fetching kernel to DRAM	6201091065	77.51	71.37
- Kernel integrity measurement	276547803	3.46	3.19
Kernel execution	399200000	4.99	4.60
Total boot time with integrity meas.	8687413170	**108.6**	100
Without integrity measurement	**# of proc. cycles**	**time (sec)**	**% of total boot time**
System initialization	1960219	0.02	0.02
First-stage boot loader			
- Fetching U-Boot	6449862	0.081	0.09
- Misc. boot loader operations	2204949	0.028	0.03
U-Boot execution			
- Fetching kernel	6201091065	77.51	89.91
- Misc. U-Boot operations	286094694	3.58	4.15
Kernel execution	399200000	4.99	5.79
Total boot time without integrity meas.	6897000789	**86.21**	100

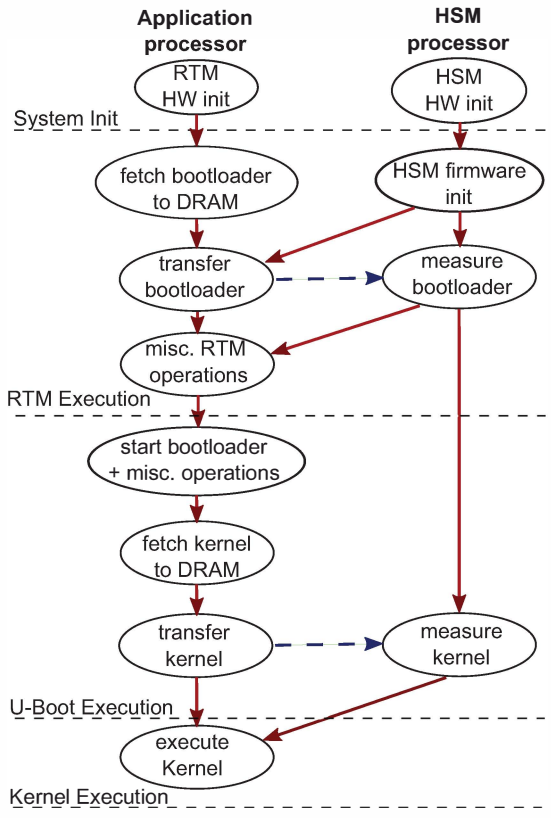

Fig. 5. Boot task dependency graph in our implementation (which does not contain DMA controllers). Red solid arrow means dependency, blue dashed arrow is implemented as streaming.

carries a Virtex5-LX110T FPGA. Apart from the FPGA, the board contains other components including SRAM, DRAM, different f ash memory chips (platform f ash, BPI f ash and SPI f ash), a Gigabit Ethernet interface and an interface for Compact Flash cards.

B. Tools and Testing

We use the BCC cross-compiler from [26], which consists of GCC 4.4.2 with the Newlib C library. Boot ROM code is generated using the MKPROM utility [27], which results in a two-step initialization process where the ROM code is f rst decompressed into a RAM before execution. We synthesize both processors with debug support units, whose interfaces would certainly be deactivated in a product.

C. Mapping to FPGA Resources

The mapping of architecture components to FPGA resources is illustrated in Fig. 3. Both processors use the same clock source. Due to the specif c FPGA resources without on-chip non-volatile memory, the FPGA evaluation has to deviate from the architecture from Sec. III in a few points. The hardware design (bitmap f le) including the RTM is stored in the platform f ash. The HSM f rmware is stored in BPI f ash. For its non-volatile 'secure' storage, the HSM also uses the BPI f ash chip. Due to FPGA resource limitations in BlockRAM and distributed RAM, the HSM f rmware is executed in the SRAM chip after startup from BPI f ash. The application processor executes the RTM in FPGA BlockRAM. Application code (boot loader, OS and applications from external storage) are executed in DRAM.

TABLE III
RESOURCE UTILIZATION

Relative to Virtex5-LX110T FPGA	Slice Reg (%)	LUTs (%)	LUTRAM (%)	BRAM/FIFO (%)	DSP48E (%)
Application processor	10.75	23.6	1.27	17.57	0
- On-chip RAM	0	0.03	0	21.62	0
- On-chip ROM	0	0	0	10.81	0
- Comm_if	0.21	0.25	0	0.68	0
HSM Processor	5.6	13.21	0.63	8.1	1.56
Relative to system	**Slice Reg (%)**	**LUTs (%)**	**LUTRAM (%)**	**BRAM/FIFO (%)**	**DSP48E (%)**
Application processor	50.23	49.65	41.83	19.85	0
- On-chip RAM	0.02	0.07	0	24.43	0
- On-chip ROM	0	0	0	12.21	0
- Comm_if	0.99	0.53	0	0.76	0
HSM Processor	26.21	27.82	20.73	9.16	100

1) Boot process in FPGA implementation: At power on, the hardware design including RTM is loaded from platform flash into the FPGA.

The HSM fetches its firmware from BPI flash, and decompresses it into SRAM and executes it. It initializes TPM data structures similar to the description in [28] including the Endorsement key [29], and verifies the SHA-1 functionality. Then it signals ready over the communication interface and waits for a command from the application processor.

The application processor decompresses the RTM code into FPGA BlockRAM and starts executing it. The RTM has FAT file system support and fetches the boot loader binary (U-Boot in this case) from a predefined location on the external Compact Flash card into DRAM (we assume that the first partition on the card is the boot partition). The RTM code then pipes the boot loader binary from DRAM through the communication interface to the HSM, which performs SHA-1 measurement and stores the hash value. After approval from the HSM, the application processor executes the U-Boot binary from DRAM, which then fetches the Linux kernel from external Compact Flash into DRAM and again pipes a copy to the HSM for measurement (extension of the hash value). After HSM approval, the kernel is booted. In our HSM firmware configuration, booting anything is always approved (but 'sealing' can be performed).

D. Results

Of special interest is the 'overhead cost' necessary to realize the 'trusted boot' functionality, in terms of hardware resource consumption and additional boot duration. We therefore compare a plain normal boot with a 'trusted boot'. We clock the design with 80 MHz (at 100 MHz, timing issues would arise).

1) Software sizes: The size of the normal U-Boot binary is 141 KB, and with added integrity measurement functionality (communication interface driver etc.) it becomes 146 KB. The size of the Linux kernel we boot is 3.2 MB.

2) Firmware sizes: The size of the uncompressed RTM code (with integrity measurement functionality) is 73 KB, for normal boot (denoted as first stage boot loader in Tab II) it is 71 KB. The HSM firmware size is 443 KB. A detailed breakdown of components of the HSM firmware is given in table I. Largest component after the command interpreter is the GMP library for computations with large numbers. The firmware further comprises TPM data, the interface driver, and functions for cryptographic operations.

3) Hardware resource consumption: A breakdown of hardware resource consumption is given in table III, both relative to the FPGA size and relative to the system itself. The HSM processor size ('trust functionality' hardware cost) is about 1/3 of the design size (due to the dual-core application processor). The RTM is denoted as application processor on-chip ROM in the table. It is synthesized into BRAM in our evaluation. If synthesized into lookup tables (LUT), it would comprise 8192 6-input LUTs.

4) Time measurements: We perform cycle-accurate time measurements with a decrementer register. A loopback test of the communication interface from the application processor through the HSM needed 2358 cycles (29.5 microseconds) for a message of 10 byte length, and 236826 cycles (2.96 milliseconds) for 2 KB. The complete boot time both with 'trusted boot' functionality and without is given in Tab. II. The table also provides a breakdown into parts as described in Sec. IV-C1. With 'trusted boot', RTM execution time comprises HSM firmware initialization, U-Boot integrity measurement time and time taken by miscellaneous RTM operations. On the other hand, U-Boot execution time with 'trusted boot' comprises the time taken by the fetching of Linux kernel, Linux integrity measurement and miscellaneous U-Boot operations. Values in brackets in that table do not count for the boot time, because the corresponding activity can be performed in parallel 'for free'. To clarify this, the boot activity dependency graph is shown in Fig. 5. The time needed to boot is determined by the longest path (critical path) through this graph. Normal boot time is 86s, while 'trusted boot' takes 109s (Tab. II). Our implementation therefore increases the boot time by 25% to realize the 'trust functionality'.

V. DISCUSSION

This paper presented an implementation of trusted boot for embedded systems based on the open source LEON3 processor, evaluated in a Virtex5 FPGA. While current embedded systems use a signature check out of a boot ROM for 'secure

boot', a 'trusted boot' provides additional security against fraudulent signatures/certif cates (e.g. in case of a security breach at a certif cate authority). The aim of the presented implementation is protection against software attacks. For such a scenario, a 'secure' coprocessor and static RTM can be implemented and synthesized as soft macro. On the other hand, the presented implementation provides no protection against physical intrusion, side channel or fault injection attacks like security chips (e.g. smartcards) would do.

The presented architecture realizes the 'trusted boot' functionality with approximately 1/3 of additional hardware cost and additional boot delay of around 25%. The code we adopted from [22] for the HSM f rmware implementation could be largely reduced in size by optimizations, which would save on-chip RAM and ROM area. The absolute boot time can be improved by employing direct memory access (DMA) controllers. A better area / boot time tradeoff is probably achievable with a smaller HSM processor and hardware accelerator for hash computation.

ACKNOWLEDGMENT

This work is partly funded by the German Federal Ministry for Education and Research (BMBF) within the project ARAMiS under funding ID 01IS11035Ü.

REFERENCES

[1] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov, "Cryptographic processors – a survey," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 357–369, 2006.

[2] N. Bourbakis and R. Kannavara, "Surveying secure processors," *IEEE Potentials*, vol. 28, no. 1, pp. 28–34, 2009.

[3] "TCG Specif cation Architecture Overview, version 1.4," Trusted Computing Group, 2007. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14

[4] "TCG PC Specif c Implementation Specif cation, version 1.1," Trusted Computing Group, 2003. [Online]. Available: http://www.trustedcomputinggroup.org/f les/resource_f les/87B92DAF-1D09-3519-AD80984BBE62D62D/TCG_PCSpecif cSpecif cation_v1_1.pdf

[5] "UEFI specif cation version 2.3.1," United EFI Forum, June 2012.

[6] "TPM main part 1 design principles, version specif cation version 1.2 revision 116," March 2011. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm_main_specif cation

[7] D. Grawrock, *Dynamics of a Trusted Platform*. Intel Press, 2009.

[8] D. Challener, K. Yoder, R. Catherman, D. Safford, and L. V. Doorn, *A Practical Guide to Trusted Computing*. IBM Press, 2008.

[9] "TPM MOBILE with Trusted Execution Environment for Comprehensive Mobile Device Security (white paper)," Trusted Computing Group, June 2012. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm_mobile_with_trusted_execution_environment_for_comprehensive_mobile_device_security

[10] S. Pan, "Trust architecture for i.mx application processors," June 2012. [Online]. Available: http://www.freescale.com.cn/cstory/ftf/2012/pdf/0211.pdf

[11] "Windows 8 hardware certif cation requirements: Client and server systems," Microsoft, July 2012. [Online]. Available: http://download.microsoft.com/download/A/D/F/ADF5BEDE-C0FB-4CC0-A3E1-B38093F50BA1/windows8-hardware-cert-requirements-system.pdf

[12] ARM Ltd., "ARM architecture," April 2013. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/index.html#reference

[13] "Skywiper (a.k.a. f ame a.k.a. f amer): A complex malware for targeted attacks," Budapest University of Technology and Economics, Lab of Cryptography and System Security, May 2012. [Online]. Available: http://www.crysys.hu/skywiper/skywiper.pdf

[14] "The smartphone who loved me: FinFisher goes mobile? (research brief)," The Citizen Lab, University of Toronto, Aug. 2012. [Online]. Available: https://citizenlab.org/wp-content/uploads/2012/08/11-2012-thesmartphonewholovedme.pdf

[15] G. Suh, C. O'Donnell, and S. Devadas, "AEGIS: A single-chip secure processor," *IEEE Design and Test of Computers*, vol. 24, no. 6, pp. 570–580, 2007.

[16] R. Kannavaral, N. Bourbakis, A. Dollas, and P. Athanas, "SCAN – secure processor," in *IEEE National Aerospace and Electronics Conference*, 2008.

[17] J. Szefer, W. Zhang, Y. Chen, D., K. Chan, W. Li, R. Cheung, and R. Lee, "Rapid single-chip secure processor prototyping on the OpenSPARC FPGA platform," in *Rapid System Prototyping Symposium*, 2011.

[18] *GRLIB IP Core User's Manual, Version 1.1.0 - B4113*, January 2012. [Online]. Available: http://www.gaisler.com/products/grlib/grip.pdf

[19] *AMBA Specif cation*, ARM Limited Std., Rev. 2.0. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.amba/index.html

[20] "Trusted platform module AT97SC3204 LPC interface summary," ATMEL, Mar. 2011. [Online]. Available: http://www.atmel.com/products/Other/embedded/trusted_platform_module.aspx

[21] "TCG PC client specif c TPM interface specif cation (TIS) 1.21," Trusted Computing Group, April 2011.

[22] M. Strasser and H. Stamer, "Software-based TPM Emulator, version 0.7.4." [Online]. Available: http://tpm-emulator.berlios.de/index.html

[23] "Das U-Boot – the Universal Boot Loader," 2012. [Online]. Available: http://www.denx.de/wiki/U-Boot/WebHome

[24] *SnapGear Linux for LEON, Version 1.37.0*, November 2008. [Online]. Available: http://gaisler.com/anonftp/linux/linux-2.6/snapgear

[25] "ML505/ML506/ML507 Evaluation Platform User Guide ," May 2011. [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf

[26] J. Gaisler and K. Eisele, "BCC - bare-C cross-compiler users manual," 2012. [Online]. Available: http://gaisler.com/doc/bcc.pdf

[27] *MKPROM2 Overview, Version 2.0.35*, April 2012. [Online]. Available: http://gaisler.com/doc/mkprom.pdf

[28] *TPM Main Part 2 TPM Structures*, Trusted Computing Group, March 2011. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm_main_specif cation

[29] S. L. Kinney, *Trusted Platform Module Basics: Using TPM in Embedded Systems*. Newnes, 2006.