embeddedworld2021 Exhibition&Conference ...it's a smarter world DIGITAL

Secure Boot Concept on the Zynq Ultrascale+ MPSoC

Thierry Delafontaine
ZHAW School of Engineering
Institute of Embedded Systems
Winterthur, Switzerland
thierry.delafontaine@zhaw.ch

Matthias Rosenthal
ZHAW School of Engineering
Institute of Embedded Systems
Winterthur, Switzerland
matthias.rosenthal@zhaw.ch

Although many embedded devices lack the required security, manufacturers realize that security is becoming an issue. Following the Platform Security Architecture (PSA) framework from Arm® various security features of the Xilinx Zynq Ultrascale+ were analyzed and implemented. The outcome is a modular reference design, where different security features can be added, depending on the individual use-case.

Security; Embedded Devices; Trusted Electronics & Secure Elements

I. Introduction

Security is a significant challenge in industrial systems. Attacks against critical systems are a harsh reality and can cause much harm to customers and the production companies. The security level of such systems is often very low. This is mainly due to the lack of know-how and the missing maturity in the market. According to Kaspersky [1], the maturity in the industrial market is low but increasing, due to the strong negative impact of incidents. Limiting factors are the lack of skill and collaborations. Thus, security has to take up a more critical role in the development of future systems.

This work focuses on the Zynq Ultrascale+ produced by Xilinx. This device is a System on Chip (SoC), which combines a multitude of different processors, as well as Programmable Logic (PL/FPGA) and hardened cores dedicated, to specific functions. The Zynq Ultrascale+ is equipped with up to four Arm® Cortex™-A53 Application Processing Unit (APU) cores, a dual-core Arm® Cortex™-R5 Realtime Processing Unit (RPU) and an Arm® Mali™-400 MP2 Graphical Processing Unit (GPU). The chip is equipped with a dedicated Platform Management Unit (PMU) for system and power management to ensure functional safety. A dedicated Configuration Security Unit (CSU) handles the hardened security features like the AES core, the RSA core, and the hashing core. It provides secure key storage and different methods to minimize key usage. Additionally, different parameters can be monitored, like voltage and temperature, to detect tampering and prevent data disclosure. The chip also integrates the Arm® TrustZone technology, as well as the Arm® Cryptography Extension to support different cryptography methods [2].

II. Related Work and Literature

Previous related work and literature There is a lot of good documentation about a security-aware design process, which originates from the IT world. This [3] preceding master thesis applies such concepts, e.g. Data Flow Diagram (DFD), to IoT devices. In a previous project, we also analyzed the Xilinx 7000 SoC and programmed a Linux driver to use the hardened AES core within Linux to decrypt sensitive user data.

Provided product documentation and implementation examples Xilinx provides a lot of documentation [2, 4, 5], as well as implementation examples [6, 7, 8, 9]. However, the examples are usually restricted to bare-metal applications. In contrast, this project concentrates on the implementation with Linux as an Operating System (OS) and U-Boot as a Second Stage Boot Loader (SSBL).

III. Secure Development

Fig. 1 shows the four steps in the PSA framework [10] provided by Arm®. In a first step, the future product is analyzed regarding, its assets, environment, and connectivity.

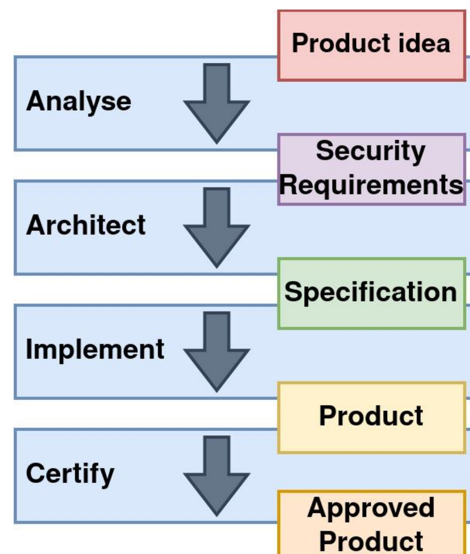

Fig. 1. Development steps in the PSA framework

Theoretical security requirements are translated to hardware specifications in the second step. These specifications influence the hardware and the software implemented in step three. The implemented solutions for identified problems are tested in the fourth step.

A. Thread Modelling

Define security goals Defining security goals is necessary, to focus on the important threats. Security goals are common specifications for a product. They can emerge from industry or company standards. Usually, three to six security goals should be specified. Security goals can be different for each company or system and are heavily dependent on the environment, the customer, and his specifications.

Visualize the product and its assets Concerning the complexity of systems, the easiest way to understand them is to visualize them in a simple representation. A Data Flow Diagram (DFD) is created to point out intersections between various subcomponents. These subcomponents range from databases to servers to external human interaction like a system administrator. A simple example is given in Fig. 2.

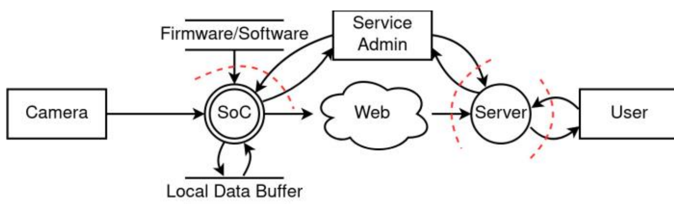

Fig. 2. Example of a DFD

In the visualization, important assets can be identified. Trust boundaries can be determined, where two components should not automatically trust each other. Thus, access to important assets or infrastructure can be restricted. Trust boundaries usually mark, where an attacker may try to access the system.

Identify adversaries Now that the assets are known, the adversaries can be identified and analyzed. There is a multitude of threat agents described in the ENISA threat landscape [11]. Each threat agent has a certain level of motivation, resources, and skills.

Identify threats STRIDE can be used for a structured approach to threat identification. STRIDE was introduced and used by Microsoft to find threats in their services and products [12]. Because of that, it is not a native threat identification approach for embedded systems. However, it can be easily applied. STRIDE is an acronym and stands for:

- Spoofing: Impersonating something or someone else to gain a non-legitimate advantage. Spoofing violates the authenticity of an application.
- Tampering: Modification of data or code while it is transmitted or stored. Thus, tampering violates the integrity of an application.
- Repudiation: Deny to have performed an action and leaving no evidence to prove against. No evidence can be linked to an attacker. This violates the non-repudiation of an application.
- Information disclosure: Exposing information to someone not authorized to see it or getting access to data without authorization. Information disclosure violates the confidentiality of an application.
- Denial of Service: Deny or degrade service to legitimate users and violating the availability of an application.
- Elevation of Privilege: Gain capabilities without proper authorization. This violates the authorization of an application.

All six categories can be applied to the elements of the DFD.

Evaluate threats The identified threats are usually too many. Thus, the threats have to be prioritized. In a good threat evaluation, the threats are prioritized according to the asset value, the adversaries, and the security goals.

Define requirements Finally, the requirements have to be defined. Requirements are simple statements and have to contain the least amount of technical details. The technical details will later be defined. For example, the tread is information disclosure, thus the requirement would be to encrypt data. How the data is encrypted and whether the chosen encryption is sufficient, is decided in the second step of the PSA framework.

IV. Feature Analysis

In contrary to IoT specific microcontroller units (MCU), the Zynq Ultrascale+ SoC combines a multitude of systems on a single chip. Therefore, only the most prominent features are described in this section.

A. Secure Boot

The boot-process of the Zynq Ultrascale+ device can be divided into four stages. Because the Zynq Ultrascale+ contains multiple processing Units, the boot process takes place on various processors. Fig. 3 shows the boot process.

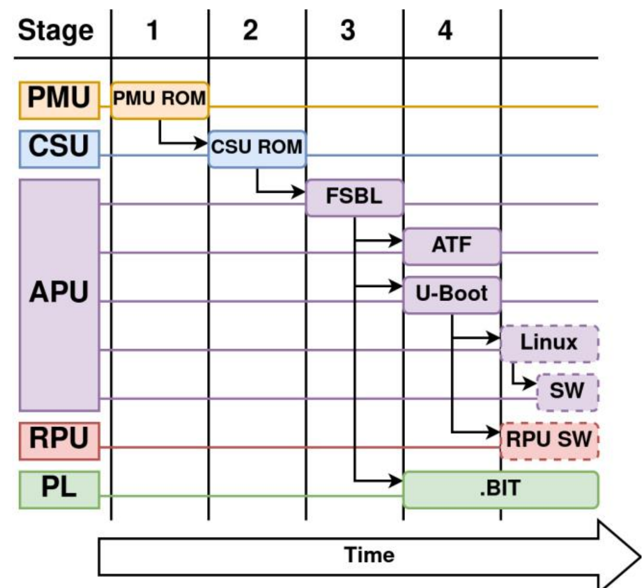

Fig. 3. Bootflow of the Xilinx Zynq Ultrascale+

Secure boot heavily depends on the principle root of trust. Root of trust ensures that each software in the "boot tree" is authenticated and therefore, trusted to execute the correct follow-up software. Up until stage two, the executed code is loaded from on-device ROMs. Thus, the code executed is trusted. Starting with stage three, the software is loaded from external sources and can be changed by the user. The first stage boot loader (FSBL), is the changing point where we decide if we want to follow a secure boot flow and what type. The Zynq Ultrascale+ offers two secure boot options.

Encryption only When you choose the "encryption only" boot process, AES encryption is enforced for images loaded from external memory. However, because the implemented AES mode is the Galois Counter Mode (GCM) the images are not only encrypted but also authenticated, with the Galois Message Authentication code (GMAC). The drawback of this mode is, that the AES key has to be stored in the eFuse.

Root of Trust This mode enforces RSA authentication of the images loaded from external memory. The key source for the optional AES encryption is choosable. Additionally, this secure boot mode offers a lot more features, like key revocation.

Stage four and the following programs are user-specific and can be either loaded by the first stage boot loader if they are small enough. Or by a following second-stage bootloader (SSL) like U-Boot.

B. *Key Storage*

The Zynq Ultrascale+ mainly offers two cryptography methods with keys. RSA as an asymmetric method for authentication and AES as a symmetric method for encryption and authentication.

For AES the key storage plays an important role because the key has to be kept secret to maintain security. The Zynq Ultrascale+ offers various key sources and modes to reduce key usage. The following table shows different modes and sources.

TABLE I. AES KEY SOURCES

AES Key Source	Description
BBRAM	Plain text key stored in Battery Backup RAM (BBRAM). This key can be reprogrammed and is lost in case the battery power is removed.
eFUSE	The key in the eFuse is either stored plain text, encrypted with the PUF key (black key), or obfuscated with the family key. The eFuse key can only be programmed once.
AES_KUP register	The key update register serves as an interface to hand over a user-provided key. A user-provided key can only be used during run-time and not during boot.
Operational Key	This key is stored in the encrypted boot header on the external storage. Therefore, this key is either decrypted with the key in the eFuse or the BBRAM key.
Rolling Key	The rolling key technique is used to prevent key exposure with Differential Power Analysis (DPA). In cases where partitions of the image reach a certain length, DPA can successfully discover the key. This mostly targets bitstreams. In such cases, the images can be split into small block sizes, which are encrypted using their key. The key is always stored inside the previous block [8].
Family Key	The family key can only be used to obfuscate the other keys. This key is the same on all devices and is managed by Xilinx.
PUF	The Physical Unclonable Function (PUF) key can only be used to encrypt the other keys. This key is device-specific.

For RSA the key storage does not play an important role, because only the public key is used during boot. It only has to be ensured, that the correct public key is enforced. Therefore, a hash of the public key can be stored inside the eFuse register, which will be checked when the authentication is enforced.

C. *Crypto API*

To improve the performance of cryptography algorithms, processors have hardware support to accelerate cryptography functions. A user-space software can calculate cryptography functions in software at any time, but only the kernel provides access to the cryptography hardware. Therefore the Linux kernel provides the Linux Crypto API to user-space programs, to make the cryptography hardware available in user-space.

Fig. 4 shows, that the Linux Crypto API is a layer between the applications requiring the cryptography hardware and the drivers for the cryptography hardware. This has the advantage, that the Crypto API provides a unified interface for hardware drivers and applications. A unified interface simplifies development and helps to keep applications portable between different platforms [13].

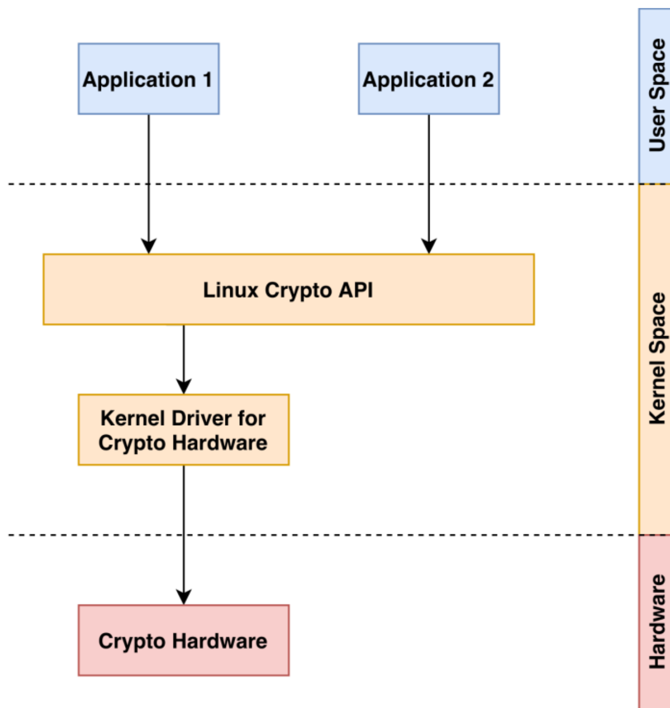

Fig. 4. Crytpo API as interface between applications and hardware drivers

Xilinx implemented drivers for the Linux Crypto API to use the underlying hardened cores for AES_GCM, RSA, and SHA.

D. TrustZone

TrustZone is a concept to isolate different parts of a system in Arm® based systems. Although, Linux also implements isolation between users and applications with different user permissions. However, with the complexity of the Linux kernel, the attack surface is huge. This makes it difficult to protect critical data if untrusted applications are running alongside trusted applications. TrustZone distinguishes between two worlds, the secure world, and the non-secure world, shown in Fig. 5.

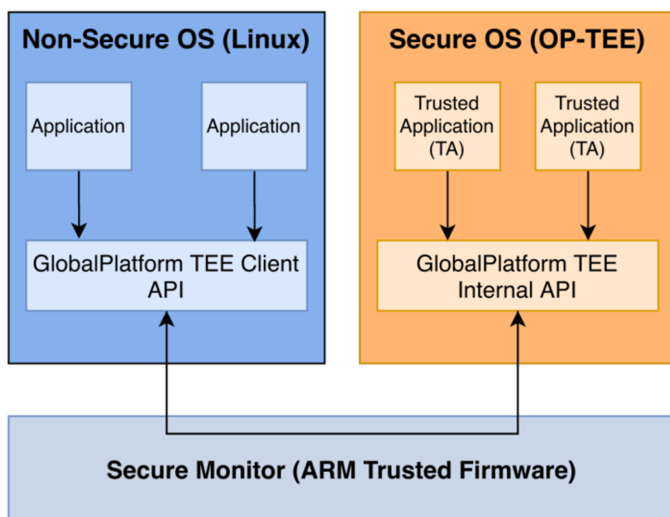

Fig. 5. TrustZone's secure and non-secure world

Most parts of the system, including Linux, are running in the non-secure world and only the critical parts of an application are executed in the secure world. Usually, if an attacker compromises Linux and gains root access to the system, the attacker has full access to the hardware. However, with TrustZone, Linux has only access to the non-secure world, and the applications in the secure world are safe [7].

Our implementation focuses on OP-TEE as a secure world OS. OP-TEE is an open-source implementation of a Trusted Execution Environment (TEE).

E. Connecting Threats, Requirements and Features

None of the previously described features would be useful if they could not be linked to detected threats. Thus, Fig. 6 gives an overview of features linked to the requirements and threats.

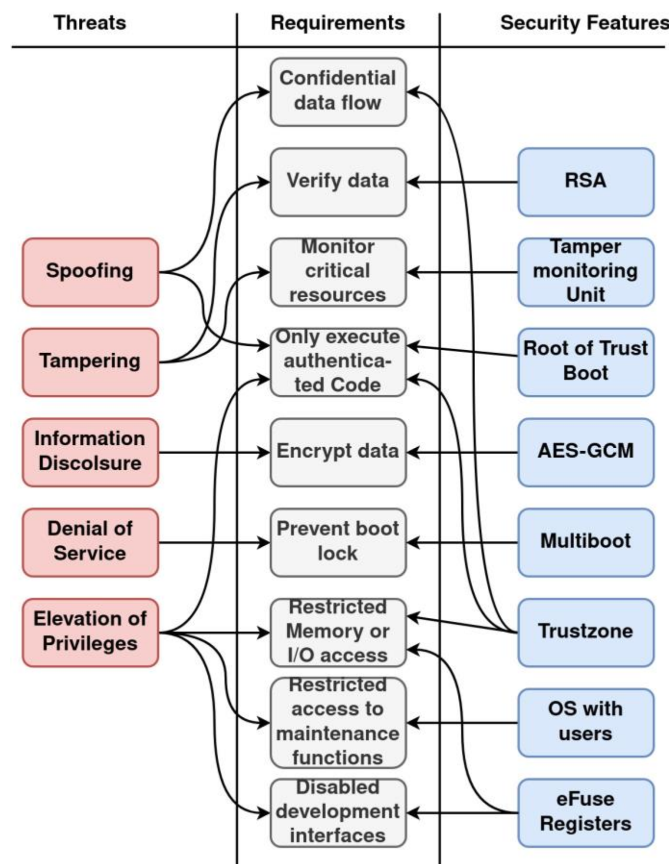

Fig. 6. Linking of threats, requirements, and security features

V. Result

Prove of concept The implemented features were only tested according to their described functionality. No extensive penetration testing has been performed to show the reliability of the implementations in real-world usage.

Reference design The work resulted in an open-source reference design. The project is accessible via Github (a link can be found at the end of this article). The idea of this reference design is to provide a basic implementation to build upon. Additionally, the instructions to the various features should

serve as a knowledge base, a summary, or as a pointer to the description provided in the product documentation.

VI. Conclusion

Security is a part of the development process, which cannot be neglected. The PSA framework helps with a structured approach. The analysis of the features shows that the Zynq Ultrascale+ has well built-in security. The modular reference design created eases the expense to quickly build a secure boot image and implement other additional security-related features.

For further information, please contact matthias.rosenthal@zhaw.ch, hans.gelke@zhaw.ch or visit the GitHub page at https://github.com/InES-HPMM/ZYNQ_USplus_secure_boot_reference_design.

References

[1] W. Schwab, M. Poujol, "The state of industrial cyber security", Kaspesky, 2018.

[2] "Zynq UltraScale+ device technical reference manual" (UG1085), Xilinx, 2019.

[3] T. Schläpfer, A. Rüst, "Embedded security with secure MCUs", Zurich University of Applied Science, Institute of Embedded Systems, 2018.

[4] "Zynq UltraScale+ MPSoC software developer guide" (UG1137), Xilinx, 2019.

[5] "Isolate security-critical applications on Zynq UltraScale+ devices" (WP516), Xilinx, 2019.

[6] "Programming BBRAM and eFUSEs" (XAPP1319), Xilinx, 2017.

[7] "Isolation methods in Zynq UltraScale+ MPSoCs" (XAPP1320), Xilinx, 2019.

[8] "Developing tamper-resistant designs with Zynq UltraScale+" (XAPP1323), Xilinx , 2018.

[9] "External secure stroage using the PUF" (XAPP 1333), Xilinx, 2018.

[10] "ENISA threat landscape report 2018", ENISA, 2019.

[11] "Platform security achitecture overview", Arm Limited, 2019.

[12] L. Osterman, "Threat modelling again", Microsoft, https://docs.microsoft.com/de-ch/archive/blogs/larryosterman/threat-modeling-again-stride, Accessed 2020.

[13] S. Mueller, M. Vasut, "Linux kernel Crypto API", https://www.kernel.org/doc/html/latest/crypto/index.html, Accessed 2020.