www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 5 Issues 6 June 2016, Page No. 16990-16995

Protecting Data in Modern Computing Devices

M. Suganiya[1] , Dr. A. Pravin[2]
Student, CSE Department,
Sathyabama University,
Chennai, India
suganiya11@gmail.com
Assistant Professor, CSE Department,
Sathyabama University,
Chennai, India
pravin_ane@rediffmail.com

ABSTRACT

The modern era is moving towards computing devices that have a high degree of mobility. The drawbacks from the security features that has to be handled is that mobile devices need to be power conscious and it can also be easily stolen due to less weight and small form factor. Due to limited power availability the amount of cryptographic computation should be minimized as these consume more power. The other major drawback from the security angle on physically missing the compute device, the data stored in both dynamic and static storage of the device can be retrieved by unauthorized persons leading to breach of privacy and confidentiality. This paper explains the modern compute device mechanisms using which the privacy and confidentiality concerns be addressed with limited amount of usage of cryptographic functions. The limited usage of cryptographic functions also has an advantage of low battery consumed.

The multi-board design with CPU module and carrier board is developed at RISE LAB, Department of CSE, IIT Madras.

Keywords: *Modern Computing Devices, Tamper Detection, High Assurance Boot, Digital signature, Root File System Data, Freescale iMX6 ARM platform.*

1. Introduction

The computing devices are stolen more easily nowadays. This leads to misuse of file system data by the unauthorized users. The data can be of personal contacts, images, videos and other secure files. These data can be protected from others by the cryptographic encryption techniques. When the data is residing on the storage media, the data will be in encrypted form. If the data has to be accessed, the private key is required for decrypting the data and then subsequently mounted. This secret private key is only known to the owner of the computing device. In case of phone lost, an attacker cannot retrieve the data without knowing the secret key.

Usually in the context of data security, one actually refers to programming in a very secure manner alone. What we are attempting
to do here is the complete product security –Hardware from being physically attacked, the verification of OS image to be that of the known factory-shipped image instead of any corruptions, and the root file system data also encrypted when it is stored.

2. Related Work

2.1 On Chip AES Scheme

The past activities have carried out the idea of putting away AES keys in CPU registers. In AES-for-

x86-particular, most of these past arrangements have failed to guard access-secured state and liable to the transport monitoring attacks. Also it is not clear how to augment these answers for protecting the voluminous access-secured state.

AESSE [2], Muller et al. AES implementation for x86 processors is done by leveraging the SSE functionality of Pentium III and to store both encoding and decoding keys to perform the AES computation. The initial implementation, on FIPS documentation [4], performs all the calculations sequentially on the information data, by creating a moderate implementation. The final implementation utilizes the table lookup optimization, to improve performance from a 100 log jam to a 6 log jam over regular AES. For this implementation, all the AES lookup tables are secured in DRAM.

The further work, TRESOR [3] proceeds on the AESSE functionality to bolster 64-bit CPUs with backing for Intel's encoding guidelines (AES-NI). This interpretation is adaptable to memory attacks because all AES's state in CPU is secured in AES-NI. On the other hand, this approach is not extensible past ensuring minimal AES implementations.

Simmon [10] stores a solitary 128-bit AES key. This key decodes different keys and performs encoding in DRAM. This increase the performance to 2 stoppage over formal AES. Thus the implementation portrayal indicates that no short-term data is put away in DRAM, although the paper does not give clear explanation on how large round tables are being secured in model-particular registers

2.2 Different models of Resistance

Encrypted scheme aims at securing data encoded in DRAM. The two main contrasts between encrypted RAM and our approach are: (1) performance overhead of encrypted RAM is usually high and (2) needs further investigation on encrypted DRAM that it can ensure over DMA attacks. DMA transfers ought to be automatically decoded. DMA does the quick memory access.

Cryptkeeper [5] extends the conventional memory ranking by suggesting and scrambling a huge divide of DRAM. Their implementation keeps DRAM's encryption keys in the secure parcel of RAM.

Chen [6] suggested the memory's encoding idea for techniques when removed from the cache. They use cache locking to guarantee those tricky data leaving the SoC is encoded. On the other hand, this past work varies from Sentry in four features. To start with, it just depicts a preparatory layout without execution. Second, the estimation is done in simulation on top of a Pentium-based architecture while artificially infusing is based on costs. Third, it is targeted at embedded systems and does not meet the Smartphone needs. There is an alternate arrangement of acceptance are made. Finally, the cache-locking technique talked about is hypothetical, and in this manner it is not clear how the cache bolts its substance.

Different process has been suggested to store encryption keys in a mystery place. ZIA [7] and Transient Authentication [8] utilize client take away, small hardware tokens, and CleanOS [9] utilizes cloud servers for securing encryption keys. Dissimilar to Sentry**,** those schemes rely on additional outer infrastructure as a Web association [9] or short-range remote interfaces [7, 8]. Transient Authentication also not bolsters running background application safely when the telephone is bolted. A further distinction remains in the sort of concept used. CleanOS gives APIs that allow execution characterize as delicate data with assurance. Instead, Sentry by encoding applications' state diminishes the programmer's weight.

Keypad [11] maintains record of system document accesses. This approach is not unequivocally safeguard over memory attacks; and it enables the victim to control what bit of data has been compromised when the mobile is lost.

2.3 Supportive Threat Models

A huge collection of work targets at guaranteeing the respectability and confidentiality of trusted applications' code and data [13, 14, 15]. These works safeguard over a compromised OS or a

compromised favored VM, and Sentry relies upon the OS to shield over DRAM attacks.

3. Proposed Work

We propose to implement the following features. They are tamper detection, high assurance boot and encrypted root file system data. We design the carrier board with tamper functionality. We propose to provide end-to-end secure computing device from the hardware to application suites at multiple levels of protection.

At hardware level, tamper detection mechanism is inbuilt in processor. On detecting physical tamper, an appropriate action is programmed inside the processor. If the device is secure, then tamper is detected immediately and all confidential data is erased automatically or made inaccessible.

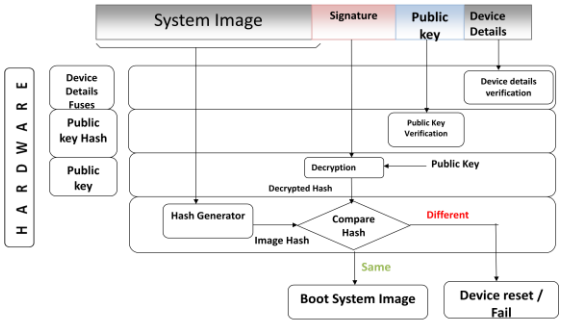

Fig 1. Secure Boot Sequence

Booting is made secure by comparing OS image with the OEM signed image. If any image modification occurred then booting is not allowed and device will be hanged. Fig 1 illustrates the verification of OS loaded image with the OEM signed image to perform secure booting.

Memory data is protected and verified by hash algorithm. The entire file system is encrypted with OTPMK (One-Time Programmable Master Key) to avoid possibility of modified rootkits. This key cannot be modified and only known to the authenticated user. Any attempt for modification or corruption, data will be erased as tamper is programmed. Decryption also requires the same private key.

4. Methodology

This paper deals with the following methodology to achieve the better performance.

4.1 Tamper Detection

In order to ensure that the physical device is not tampered, there is a tamper detection mechanism inbuilt as part of the processor itself. On detection of a physical tamper, appropriate action can be programmed inside the processor. Typically for a very secure device, the default action for the detection of a tamper event would be that all private confidential information are automatically erased or made inaccessible. This mechanism available in the processor is used as one means of protection of critical data stored in the mobile compute device when it is lost.

4.2 High Assurance Boot

The Operating System in computing device for booting is to be protected from unauthorized modifications. The OS image that is attempting to get booted should not have been modified as compared to the original image that was shipped out from factory. If there have been any modifications to the image, the device would not be booting up. This kind of protection to the device is referred as High Assurance Boot.

4.3 Root File System Data

The next level of protection to be handled is stored data in the memory card. This protection level is highly required as most confidential data will be present here.

For example, when an owner of the device is using a compromised application in the device, might lose critical information from the device sent through this compromised application to some remote location. In order to avoid the possibility of modified rootkits, the entire root file system is encrypted and stored. This area gets decrypted only after the private key is entered at the time of mounting the root file system. Through this mechanism, the data in Root File System is protected from getting compromised.

5. Attacks Handled

The computing devices are facing several memory attacks. Some of the memory attacks discussed in our paper are cold Boot, Bus monitoring and DMA attacks.

5.1 Cold Boot Attack

Cold Boot attack is a physical attack where even after removing the memory from device on switch off, the data in it will not be erased immediately. So the attacker can both mount this memory to another device and retrieve the data or by resetting the system and rebooting with the attacker-controlled OS can read and output the memory. But on encrypting the memory the data read out cannot be understood.

5.2 Bus Monitoring Attack

Data is carried within CPU, memory and other peripherals by the bus mechanism. The attacker can track this bus to snoop the data. On encrypting and decrypting the entire CPU the attack is not possible to occur. This is one way of securing the data from attacker.

5.3 DMA Attack

Direct Memory Access (DMA) Attack is called Firewire attacks that changes DMA locations to read arbitrary memory locations containing sensitive data. ARM Trust zone is the system security for wide area of client and server computing. ARM Trusted Zone will deny access to DMA and classify the memory as normal world and secure world. Normal world contains user readable files and secure world contains files that can be only read by secure OS. Mostly confidential data are stored in secure world.

6. System Setup

Freescale iMX6 ARM processor

The IITM System on a Module (SOM) is Freescale's IMX6Q based platform. This module houses a low cost Quad-Core ARM® Cortex A9 processor at 1GHz with 1 MB of L2 cache, and on board 64-bit DDR3.The SOM has a small factor, roughly measuring around 6.76 cm x 5.7 mm. The SOM is always used with a carrier board. The carrier board also houses standard communication interfaces like GSM WIFI/Blue tooth, USB2.0 high-speed serial interface. Support options for integrated market specific I/O like HDMI, SD3.0, GigE, SATA-II with integrated PHY,PCI Express ® with integrated PHY are also present. The carrier board has two MIPI front and back cameras, a 7x11 touch screen, an audio module, fuel gauge for battery life indication, NFC support and SD card slots. IITM SOM and the Carrier board are together housed in an ergonomic enclosure. They are together referred as IITM Tablet. The IITM tablet has a tamper detection mechanism, high assurance secure boot, encrypted file system support. This makes it as a perfect lab platform for developing secure systems.

The following are some of the processor specific features that our system is leveraging on for providing the secure functionalities:

i. ARM TrustZone including the TZ structural planning.
ii. SJC-System JTAG Controller. Regulating so as to shield JTAG from investigate port assaults or obstructing the entrance to the framework troubleshoot highlights
iii. CAAM - Cryptographic Acceleration and Assurance Module, fusing 16KB protected RAM
iv. SNVS - Secure Non-Volatile Storage, including Secure Real Time Clock
v. CSU - Central Security Unit. Arranged amid boot and by e-combines, and will focus the security level operation mode and also the TZ approach
vi. A-HAB-Advanced High Assurance Boot - HABv4 with the following implanted upgrades: SHA-256, 2048 bit RSA key, variant control instrument and warm boot.

Security is upheld by:

i. High Assurance Boot (HAB4) System
ii. ARM TrustZone (TZ) Trusted Execution environment

iii. DDR Memory secure area assurance by TrustZone Address Space Controller
iv. On-chip RAM (OCRAM) single area TrustZone assurance
v. Peripheral access arrangement control, utilizing Central Security Unit (CSU)
vi. One-Time Programmable (OTP) electrical circuit exhibit (Total of 3840-bit e-wires) by means of the On-chip electrical wire controller (OCOTP_CTRL)
vii. CAAM and SNVS security construction modeling, giving:
viii. 16KB Secure RAM
ix. Secure Real Time Counters
x. Security State Controller
xi. Encryption and Hashing capacities, Random Number Generator (RNG)
xii. Security Violation/Tamper Detection & Reporting
xiii. System JTAG controller (SJC)
xiv. Secure Real Time Clock

7. Implementation

This section deals with the implementation of secure U-boot. The structure of signed secure U-boot image is shown in Fig 2. The three main parts of implementation are:

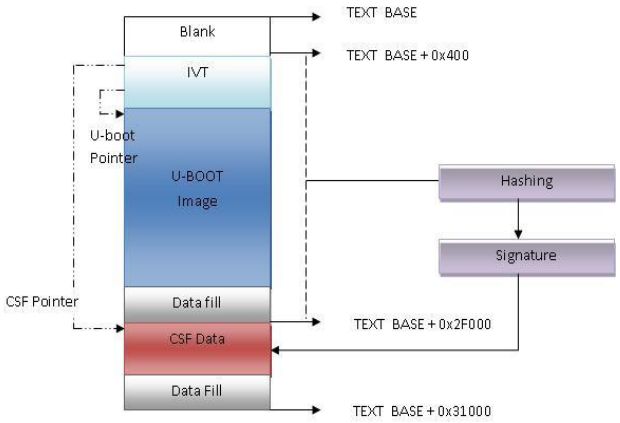

Fig 2. Secure U-Boot Image

7.1 Secure U-boot creation

The steps for creation of secure U-boot are

i. Initially, U-boot repository source code has to be extracted from Freescale.
ii. To support cross compilation on to ARM platform, corresponding variables have to be exported.
iii. Next, base address should be fixed for loading the image into the memory.
iv. U-boot.lds file is changed for adding HAB related enhancements.
v. One more file is added to U-boot main code for enabling hab_status command.
vi. This command will study HAB events and perform secure boot debugging.

7.2 Installation of CST tool

The Code Signing Tool (CST) of version CST 2.0 is installed. CST contains executables for signing the image, generation of certificates and for key generation like fuse values.

7.3 Image Signing

The unsigned U-boot images are placed in a folder along with the U-boot.bin for signing. Keys are selected for CSF signing. A script file habimagegene.sh is generated and executed for producing the padded and signed U-boot file.

8. Conclusion

This paper is handling the security of the entire system with several secure features like tamper detection, high assurance boot and encrypted file system against the memory attacks. From the above preliminary discussion of the features, it is clear that the only way for us to implement a high end secure system would be to make use of processor internal features and this is precisely what the system proposed by us is leveraging on.

9. Future Work

As part of this work, we have secured the device from the hardware enclosure to the boot loader to the OS image as well as the root file system completely. The next layer that is required to be done is the applications on the Android layer that needs to be protected and

made secure. For this, the security framework provided by Android has to be studied and adopted to make the Application SW layer also to be very secure. The next step is to secure the execution environment and the critical programs need to execute securely.

References

[1] Freescale IMX 6 Dual/6 Quad ARM Cortex- A9 core processor Reference Manual http://cache.freescale.com/files/32bit/doc/ref_manual/IMX6DQRM.pdf?fpsp=1&WT_TYPE=Reference%20Manuals&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation&fileExt=.pdf

[2] T.Muller, A. Dewald, and F. C. Freiling. AESSE: a cold-boot resistant implementation of AES. In Proc. of the 3rd European Workshop on System Security (EUROSEC), 2010.

[3] T. Muller, A. Dewald, and F. Freiling. TRESOR runs encryption securely outside RAM. In Proc. of the 20th USENIX Security Symposium, 2011.

[4] NIST. Pub. 197 – advanced encryption standard (AES). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.

[5] P. A. Peterson. Cryptkeeper: Improving security with encrypted RAM. In Proc. of IEEE International Conference on Technologies for Homeland Security, 2010.

[6] X. Chen, R. P. Dick, and A. Choudhary. Operating system controlled processor-memory bus encryption. In Proceedings of the conference on Design, automation and test in Europe, 2008.

[7] M. D. Corner and B. D. Noble. Zero-interaction authentication. In Proc. of the 8th Annual International conference on Mobile computing and networking (Mobicom), 2002.

[8] M. D. Corner and B. D. Noble. Protecting applications with transient authentication. In Proc. of the 1st International Conference on Mobile systems, applications and services (MobiSys), 2003.

[9] Y. Tang, P. Ames, S. Bhamidipati, A. Bijlani, R. Geambasu, and N. Sarda. CleanOS: Limiting mobile data exposure with idle eviction. In Proc. of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2012.

[10] P. Simmons. Security through amnesia: A software-based solution to the cold boot attack on disk encryption. In Proc. of the 27th Annual Computer Security Applications Conference (ACSAC), 2011.

[11] R. Geambas¸u, J. P. John, S. D. Gribble, T. Kohno, and H. M. Levy.Keypad: An Auditing File System for Theft-prone Devices. In Proc.of the European Conference on Computer Systems (EuroSys), 2011.

[12] Patrick Colp, Jiawen Zhang, James Gleeson, Sahil Suneja, Eyal de Lara, Himanshu Raj, Stefan Saroiu, Alec Wolman: Protecting Data on Smartphones and Tablets from Memory Attacks.In Proc. of the conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'15), 2015.

[13] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. K. Ports. Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems. In Proc. of 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Seattle, WA, 2008.

[14] J. Criswell, N. Dautenhahn, and V. Adve. Virtual ghost: Protecting applications from hostile operating systems. In Proc. of 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2014.

[15] O. H. A. Dunn, S. Kim, M. Lee, and E. Witchel Inktag: Secure applications on an untrusted operating system. In Proc. of 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2013.

[16] High Assurance Boot on iMX6, by IIT, Madras, available in Freescale Support community portal

DOI: 10.18535/ijecs/v5i6.33