Performance evaluation of cryptographic operations on a SAMR21-XPRO board

Mathias Tausig, Silvia Schmidt

Motivation

Ongoing research project concerned with secure firmware upgrade for IoT devices.
Position paper with our current state of development has been presented to the *Internet of Things Software Update Workshop (IoTSU)* held by the *Internet Architecture Board(IAB)* in Dublin in June 2016:
Secure Firmware Update Over the Air in the Internet of Things Focusing on Flexibility and Feasibility - Proposal for a Design
by Silvia Schmidt, Mathias Tausig, Matthias Hudler, Georg Simhandl

Motivation

> The Internet of Things (IoT) is becoming more and more a part of our everyday life

Motivation

- The Internet of Things (IoT) is becoming more and more a part of our everyday life
- Reports of incidents are becoming more frequent

Motivation

- The Internet of Things (IoT) is becoming more and more a part of our everyday life
- Reports of incidents are becoming more frequent
- People are afraid

Motivation

- The Internet of Things (IoT) is becoming more and more a part of our everyday life
- Reports of incidents are becoming more frequent
- People are afraid
- If we want IoT to be a success, something needs to be done

Problems

Time-to-market is often too short, to test device sufficiently, thus upgrading a device’s firmware becomes necessary to fix erroneous behaviour.

Problems with Firmware Upgrades

- An update gone wrong can brick the device
- On-site upgrade is not practical and will thus not be done
- Remote update can be an attack vector to gain control of the device
- A remote update can leak sensible information

Many devices have **many capabilities** to help remedy these problems

Our Solution

A proposal for a secure Firmware upgrade over the air (FOTA) which is as generic and lightweight as possible, able to be used across many devices with very little implementation overhead. We aim to be easy on the device’s memory and on the hardware cost.

We do not

- use a TPM
- use an HSM or Secure Element
- use memory protection

Our Solution

We do

Our Solution

We do

- tackle the classic problems of IT-Security: authenticity, confidentiality and integrity

Our Solution

We do

- tackle the classic problems of IT-Security: authenticity, confidentiality and integrity
- offer a safety approach to prevent device bricking

Our Solution

We do

- tackle the classic problems of IT-Security: authenticity, confidentiality and integrity
- offer a safety approach to prevent device bricking
- use an *Atmel SAMR21-XPRO with Cortex-M0+ processor* (256Kb flash, 32Kb RAM) as a reference device

Our Solution

We do

- tackle the classic problems of IT-Security: authenticity, confidentiality and integrity
- offer a safety approach to prevent device bricking
- use an *Atmel SAMR21-XPRO with Cortex-M0+ processor* (256Kb flash, 32Kb RAM) as a reference device

We do not

Our Solution

We do

- tackle the classic problems of IT-Security: authenticity, confidentiality and integrity
- offer a safety approach to prevent device bricking
- use an *Atmel SAMR21-XPRO with Cortex-M0+ processor* (256Kb flash, 32Kb RAM) as a reference device

We do not

- protect you from attackers who can physically tamper the device

Our Solution

Create a lightweight bootloader which handles the whole update process and all necessary security functions

Our Solution

Create a lightweight bootloader which handles the whole update process and all necessary security functions

- Applying an electronic signature or a MAC on the update package to guarantee its integrity and authenticity

Our Solution

Create a lightweight bootloader which handles the whole update process and all necessary security functions

- Applying an electronic signature or a MAC on the update package to guarantee its integrity and authenticity
- Encrypt the update package to ensure confidentiality

Our Solution

Create a lightweight bootloader which handles the whole update process and all necessary security functions

- Applying an electronic signature or a MAC on the update package to guarantee its integrity and authenticity
- Encrypt the update package to ensure confidentiality
- Achieve fault tolerance through a dedicated backup area

Our Solution

Create a lightweight bootloader which handles the whole update process and all necessary security functions

- Applying an electronic signature or a MAC on the update package to guarantee its integrity and authenticity
- Encrypt the update package to ensure confidentiality
- Achieve fault tolerance through a dedicated backup area
- Make the approach configurable and thus more adaptable to different scenarios and devices

Our Solution

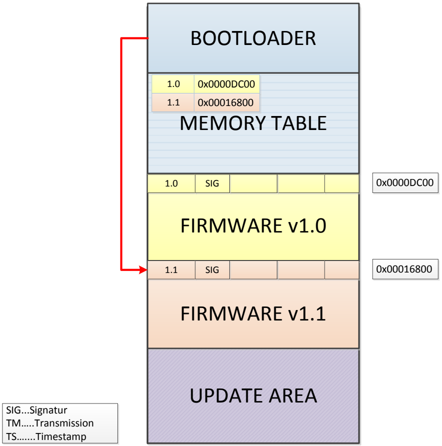
Create a lightweight bootloader which handles the whole update process and all necessary security functions

- Applying an electronic signature or a MAC on the update package to guarantee its integrity and authenticity
- Encrypt the update package to ensure confidentiality (Optional)
- Achieve fault tolerance through a dedicated backup area (Optional)
- Make the approach configurable and thus more adaptable to different scenarios and devices

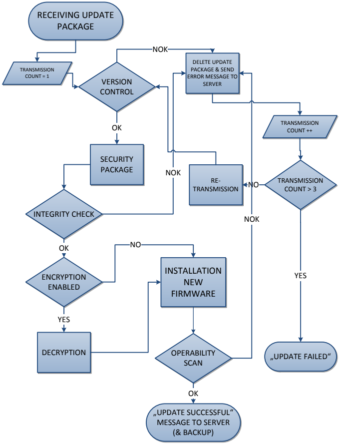
The Bootloader

- Preinstalled upon device manufacturing
- Comes preinstalled with public and/or secret key(s)
- Manages a memory table holding all keys and metadata of the installed firmware image(s)
- Makes the decision if an update package is to be installed
- Makes the decision which firmware is to be booted

Device memory overview



Update process



Protocols used

- ECDSA for the electronic signatures (well tested, more lightweight than RSA)
- AES for encryption (fast, well implemented on small devices, often even hardware support)
- HMAC or CMAC for package authentication, if public key cryptography would use too much (flash) memory
- LWM2M (CoAP based), 6LoWPAN and TFTP for wireless communication

Open Questions

- Key rollover
- Management of the symmetric keys

Open Questions

- Key rollover
- Management of the symmetric keys

Suggestions very welcome!

Goal

A full specification and working reference implementation by the end of the year

Evaluations

In order to define the actual specifications for our update process, a lot of cryptographic primitives had to be evaluated, to see which algorithms are usable on such low end hardware, and which are not.
Those evaluations were done using RIOT-OS.

Evaluations

We have measured runtime, memory consumption and ROM requirements for the following:

- ECDSA key generation, signature creation, signature verification (using *micro-ecc*)
- Ed25519 signature creation and verification
- Hash calculations using SHA-1 and SHA-2
- Symmetric cipher functions (AES and Twofish) in various operation modes (CBC, CCM)
- Hashing the plain text of encrypted data

Results

The actual numbers are in the paper.

Results

The actual numbers are in the paper.

- The *micro-ecc* implementations of ECDSA is very lightweight (especially when only verification is needed) and perfectly usable for our purpose

Results

The actual numbers are in the paper.

- The *micro-ecc* implementations of ECDSA is very lightweight (especially when only verification is needed) and perfectly usable for our purpose
- Ed25519 is supposed to be faster than ECDSA, but the available implementations are not

Results

The actual numbers are in the paper.

- The *micro-ecc* implementations of ECDSA is very lightweight (especially when only verification is needed) and perfectly usable for our purpose
- Ed25519 is supposed to be faster than ECDSA, but the available implementations are not

Signature verification with comparable bit-length: 500 ms versus 3916 ms.

Results

The actual numbers are in the paper.

- Verifying a signature of the plain text of encrypted data is doable even with that limited amount of RAM.

Results

The actual numbers are in the paper.

- Verifying a signature of the plain text of encrypted data is doable even with that limited amount of RAM.

Calculating the SHA-256 hash of the plaintext of a large amount (128 kB) of encrypted data takes 1831 ms and uses 3120 byte RAM.

Contributions

Some contributions to the open source world have been created in the course:

Contributions

Some contributions to the open source world have been created in the course:

- The *micro-ecc* package of RIOT-OS has been updated

Contributions

Some contributions to the open source world have been created in the course:

- The *micro-ecc* package of RIOT-OS has been updated
- Implementation of stronger curves for *micro-ecc* (to be released)

Contributions

Some contributions to the open source world have been created in the course:

- The *micro-ecc* package of RIOT-OS has been updated
- Implementation of stronger curves for *micro-ecc* (to be released)
- Streamlined the Hash Function interface of RIOT-OS

Contributions

Some contributions to the open source world have been created in the course:

- The *micro-ecc* package of RIOT-OS has been updated
- Implementation of stronger curves for *micro-ecc* (to be released)
- Streamlined the Hash Function interface of RIOT-OS
- Implemented the HWRNG in RIOT-OS for our evaluation board

Contributions

Some contributions to the open source world have been created in the course:

- The *micro-ecc* package of RIOT-OS has been updated
- Implementation of stronger curves for *micro-ecc* (to be released)
- Streamlined the Hash Function interface of RIOT-OS
- Implemented the HWRNG in RIOT-OS for our evaluation board
- Created a lightweight ASN.1 parser implementation for embedded devices

Acknowledgements

This paper has been written as part of the research project **SecureIoTy**

Academic partner
FH CAMPUS WIEN UNIVERSITY OF APPLIED SCIENCES

Industrial partners
adaptivia intelligent sensor networks
EMBEDDED DEVICE SOLUTIONS EMBEDIT

Funding
vienna business agency
A service offered by the City of Vienna

Contact: Georg Simhandl (georg.simhandl@adaptivia.com)

We are looking for more industrial partners. If you are interested, please let us know.

THANK YOU