Solving IoT Security Challenges with OAuth 2.0

Hannes Tschofenig

Content

- Design Patterns
- Constraints
- ACE
  - What's new on top of OAuth?
  - Ongoing work in ACE
- Call for help

Design Patterns

1. Device-to-Device Communication
2. Device-to-Cloud
3. Backend Data Sharing
4. Gateway Communication
5. IP-based Device-to-Device

(More details available at RFC 7452 or at the IoT Overview whitepaper published by the Internet Society (ISOC) at http://www.internetsociety.org/doc/iot-overview.)

Device-to-Device Design Pattern

Device-to-Device Examples

Hearing Aid

Suunto Ambit 3

StickNFind

Beacons

Parrot

Cadence Sensor

Security Considerations

- These use cases are today based on 1-to-1 relationship between a peripheral and a master (smart phone, sports watch).
- This relationship is established using pairing.
- Some BLE use cases use no security at all (e.g., beacons).

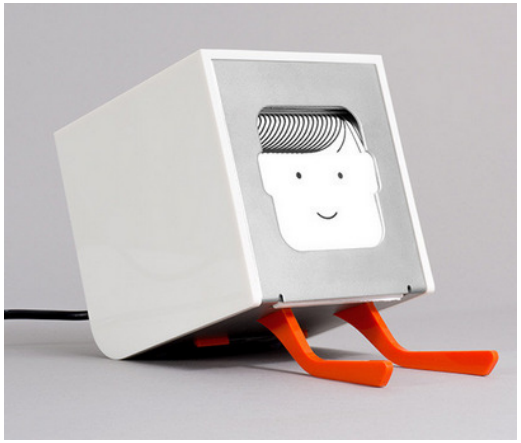
Security Considerations, cont.

But what if …

- Access by multiple users is demanded?
- Permissions dynamically change?
- Fine-grained authorization decisions are needed?
- Access policies need to be managed centrally?
- Access rights can be delegated?
- Existing authentication infrastructures has to be re-used?

Device-to-Cloud Design Pattern

Device-to-Cloud Examples



LittlePrinter

Withings Scale

Dropcam

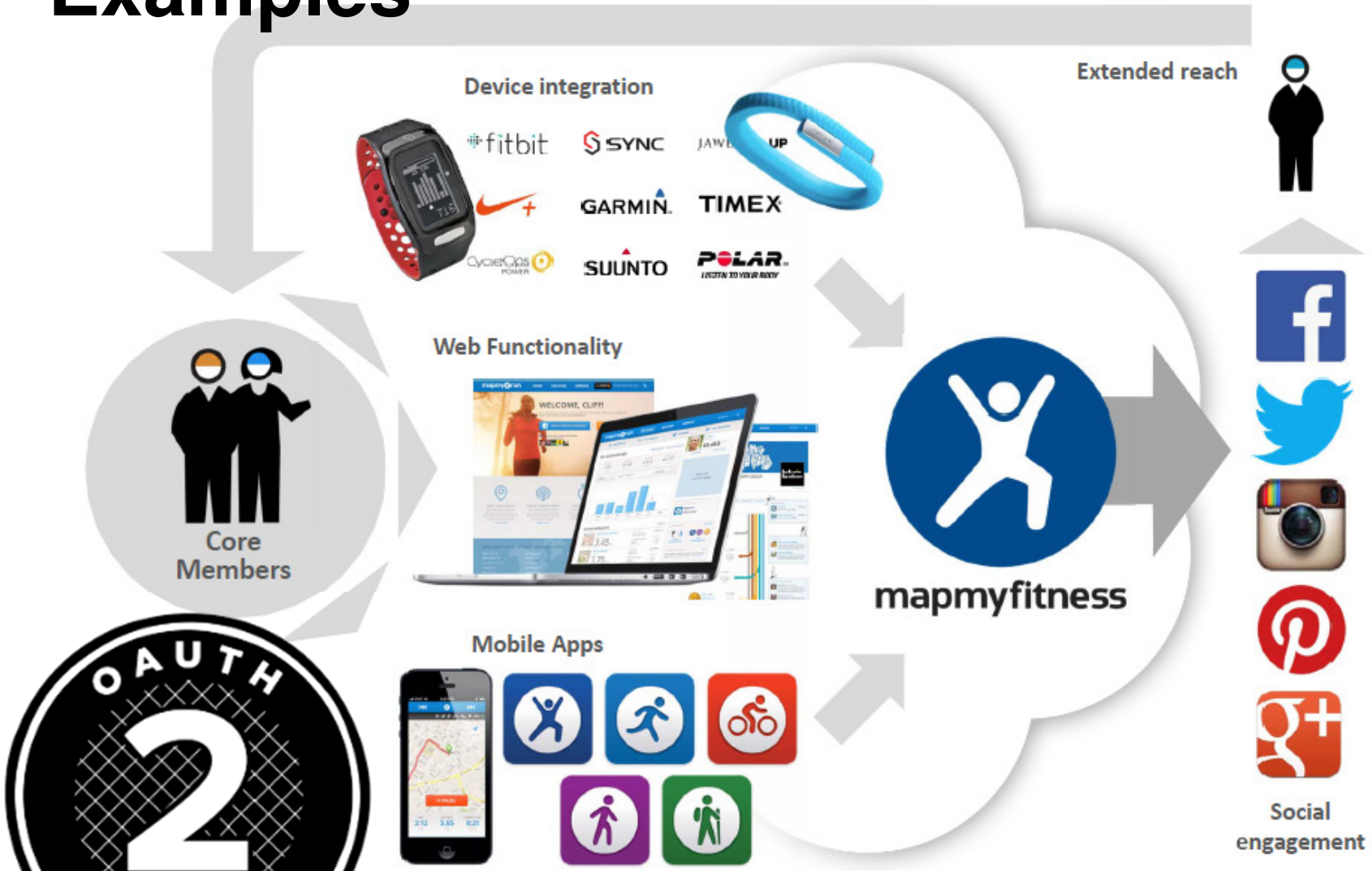
Tractive

Security Considerations

- Credentials needed for
  - Network access (e.g., WiFi credentials)
  - End-to-end security for cloud access.
- The **OAuth 2.0 Device Flow (draft-ietf-oauth-device-flow>** is relevant to this scenario.
  - But often manufacturer provided secrets are used to only talk to pre-configured server.
  - For some radio technologies the two credentials are combined and devices can only talk to an operator-provided server. E.g., low power WANs.

Backend Data Sharing Design Pattern

Backend Data Sharing Examples



MapMyFitness

https://www.mapmyapi.com/docs/OAuth_2_Intro

Gateway Design Pattern

Smart Phone as Gateway Examples

Zepp Golf Sensor

Oral-B Toothbrush

Fitbit

Garmin Forerunner 920XT

Security Considerations

- Classical app security challenges.
  - Rarely end-to-end security between the IoT device and the cloud service.
  - Device needs to be associated with the user. Typically requires some form of identity management.
- The **OAuth 2.0 for Native Apps** (**draft-ietf-oauth-native-apps>** is relevant for this scenario.

Independent Gateway Examples

Philips

NXP Janet-IP

SmartThings

Ne
st

Revolv Smart Home Gateway

Security Considerations

- Typically used for bridging different link layer security technologies but may also gateway different IoT frameworks.
- Particularly relevant for mesh networks like the IEEE 802.15.4. The specific access network authentication technologies are relevant, such as Thread.
- End-to-end security possible!
- Allows communication to cloud infrastructure as well as communication between IoT devices themselves.

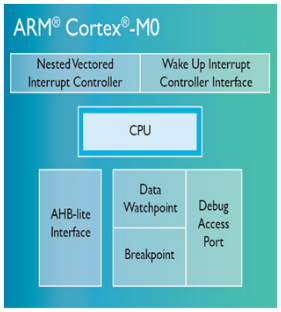
IP-based Device-to-Device Design Pattern

IP-based Device-to-Device

- IP network local-only communication
- UPnP style of communication
- Security:
  - Communication assumed to be local in the network.
  - Today, no authentication and authorization based on network access.

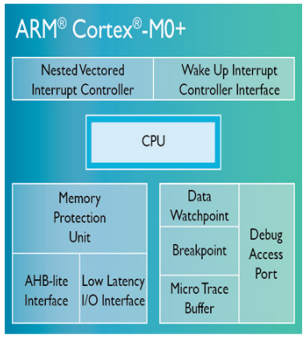
IoT Device Constraints

Cortex-M Processors



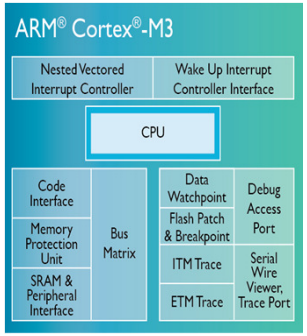
Lowest cost
Low power

Example: Touchscreen Controller



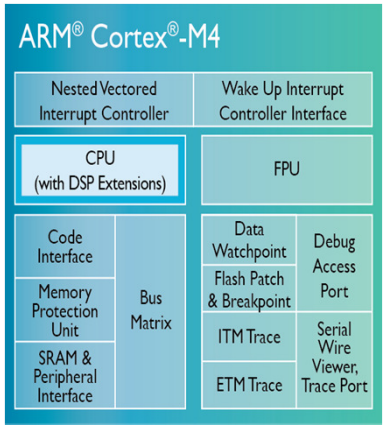
Lowest power
Outstanding energy efficiency

Example: Sensor node
Bluetooth Smart



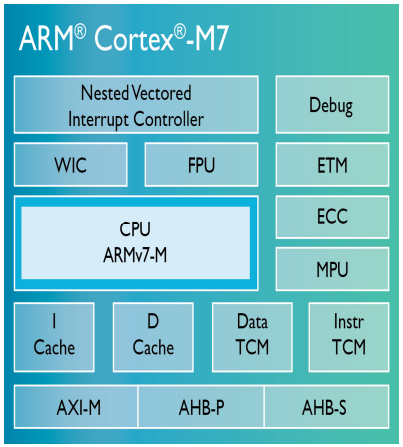
Performance & efficiency
Feature rich connectivity

Example: Weables,
Activity trackers, Wifi receiver



Digital Signal Control (DSC)/
Processor with DSP
Accelerated SIMD
Floating point (FP)

Example: Sensor fusion, motor control



Maximum Performance
Flexible Memory Cache
Single & Double Precision FP

Examples: Automotive, High-end audio set

Processors use the 32-bit RISC architecture:
http://www.arm.com/products/processors/cortex-m/index.php

Example: STM32F215RET6

- 32bit CORTEX M3 with120 Mhz
- Flash: 512 KB
- RAM: 128KB
- Features:
  - Interfaces: CAN, I2C, SPI, UART, USART, USB, Ethernet
  - Camera interface
  - Random number generator, hardware accelleration (AES-128, AES-192, AES-256, Triple DES, MD5, SHA1, HMAC)
  - Real-Time Clock
  - A/D & D/A Converters
  - Temperature sensor
  - Serial wire JTAG debug port
- Datasheet can be found at http://www.farnell.com/datasheets/1874880.pdf.

Farnell

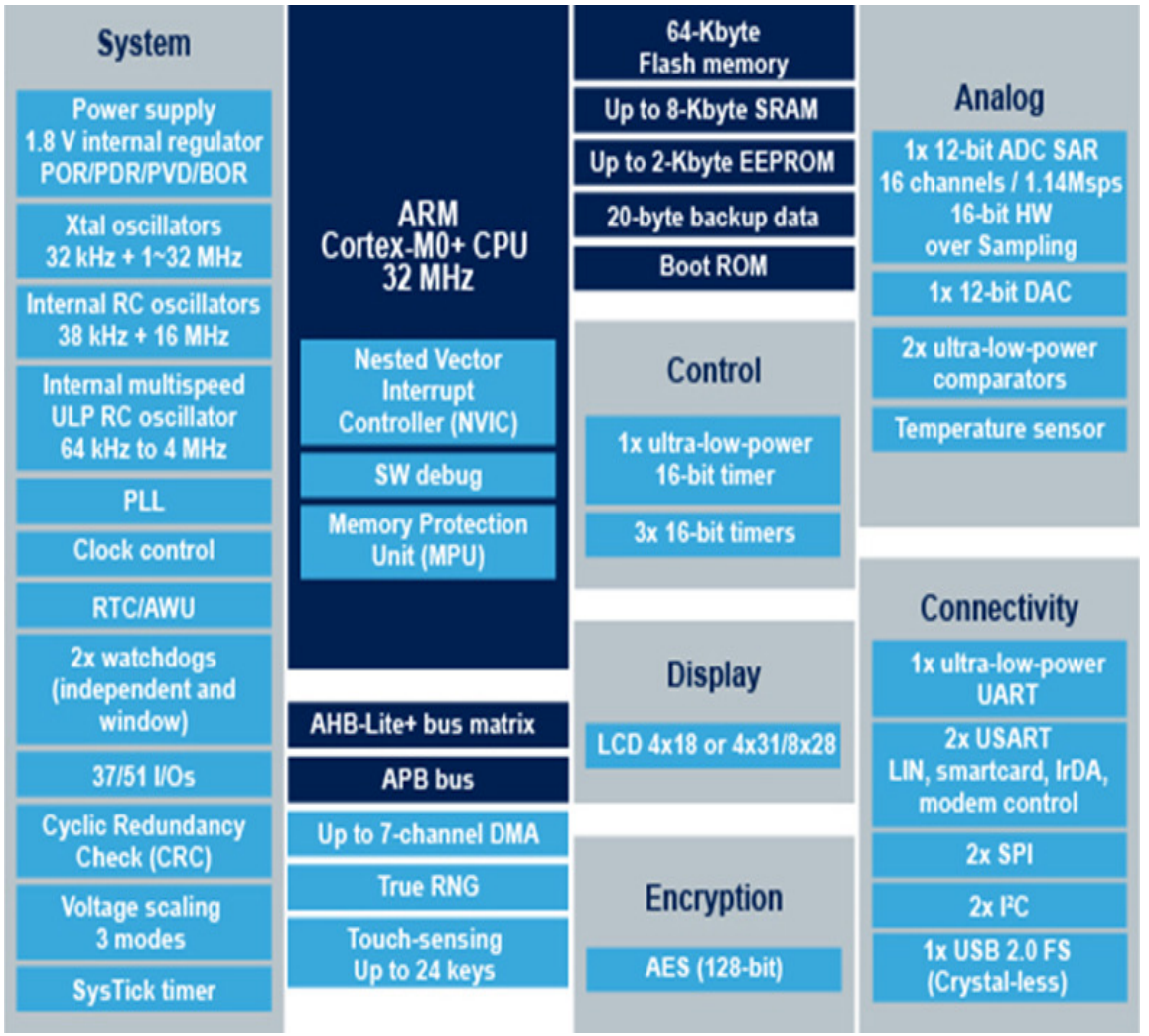
Volume	Price
1 - 9	€ 16,49
10 - 99	€ 13,61
100 - 249	€ 10,15
250 - 499	€ 8,99
500 - 999	€ 8,09
1000 - 1999	€ 7,35
2000+	€ 6,62

Example: STM32L063C8T6

- 32bit CORTEX M0+ with max 32 Mhz
- Flash: 64 KB
- RAM: 8 KB

Mouser

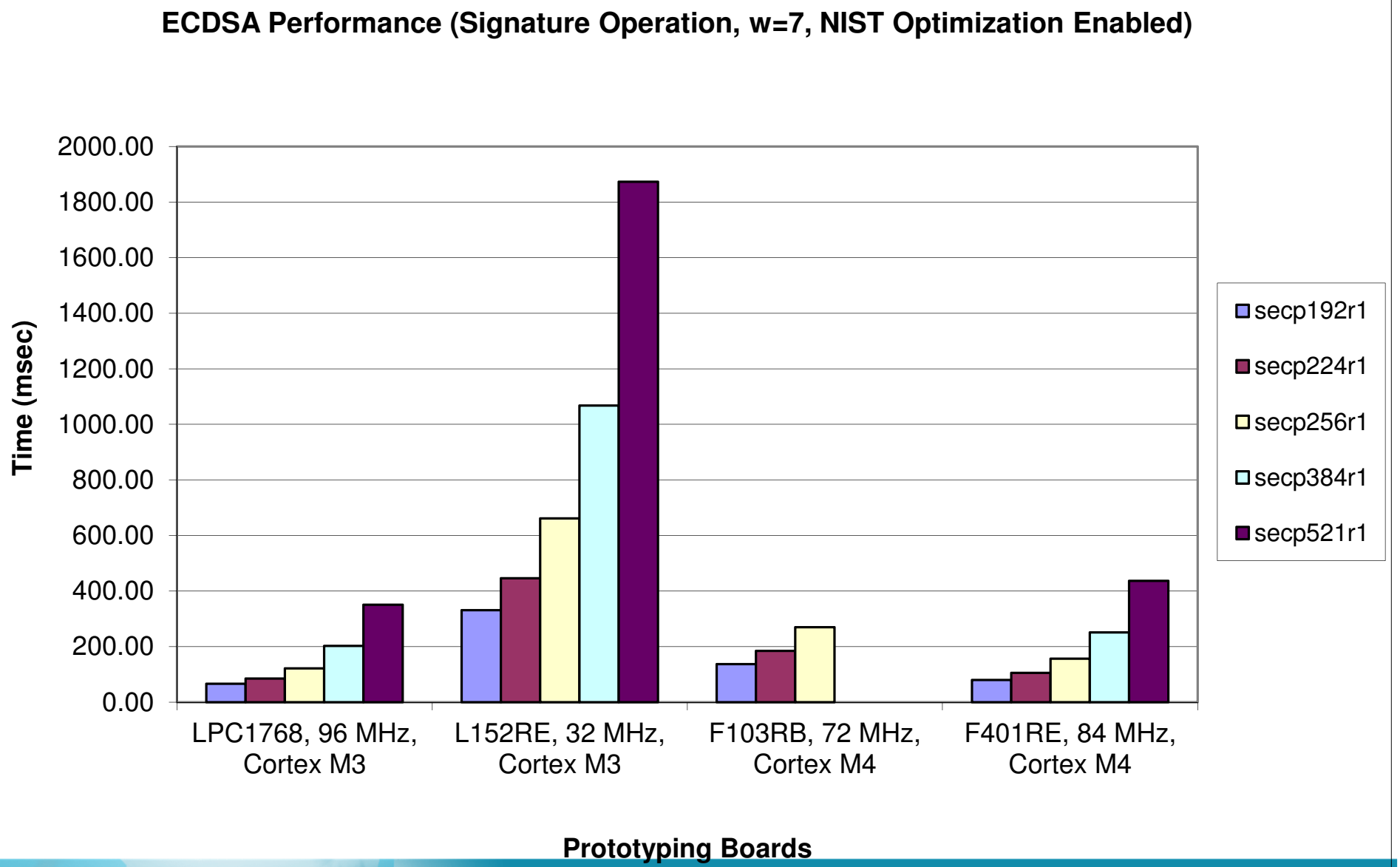
Volume	Price
1000	€ 2,25
2500	€ 2,14
5000	€ 2,06



Background Notes about Optimizations

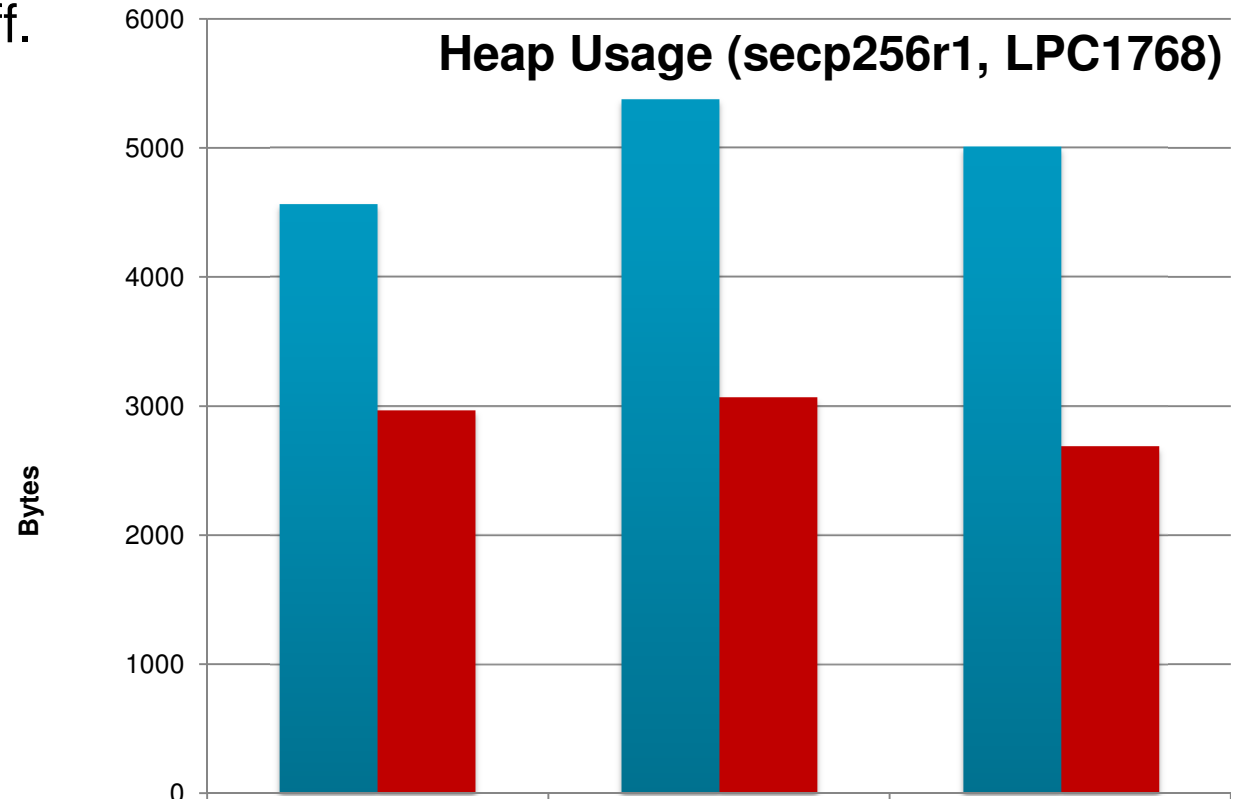
- NIST Optimization
  - Utilizes special structure of NIST chosen curves.
  - Appendix 1 of http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf
  - Longer version in FIPS PUB 186-4: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
  - Configuration parameter: ECP_NIST_OPTIM
- Fixed Point Optimization:
  - Pre-computes points
  - Described in https://eprint.iacr.org/2004/342.pdf
  - Configuration parameter: ECP_FIXED_POINT_OPTIM
- Window:
  - Technique for more efficient exponentation
  - Sliding window technique described in https://en.wikipedia.org/wiki/Exponentiation_by_squaring
  - Configuration parameter: ECP_WINDOW_SIZE (min=2, max=7).

Performance Comparison: Prototyping Boards



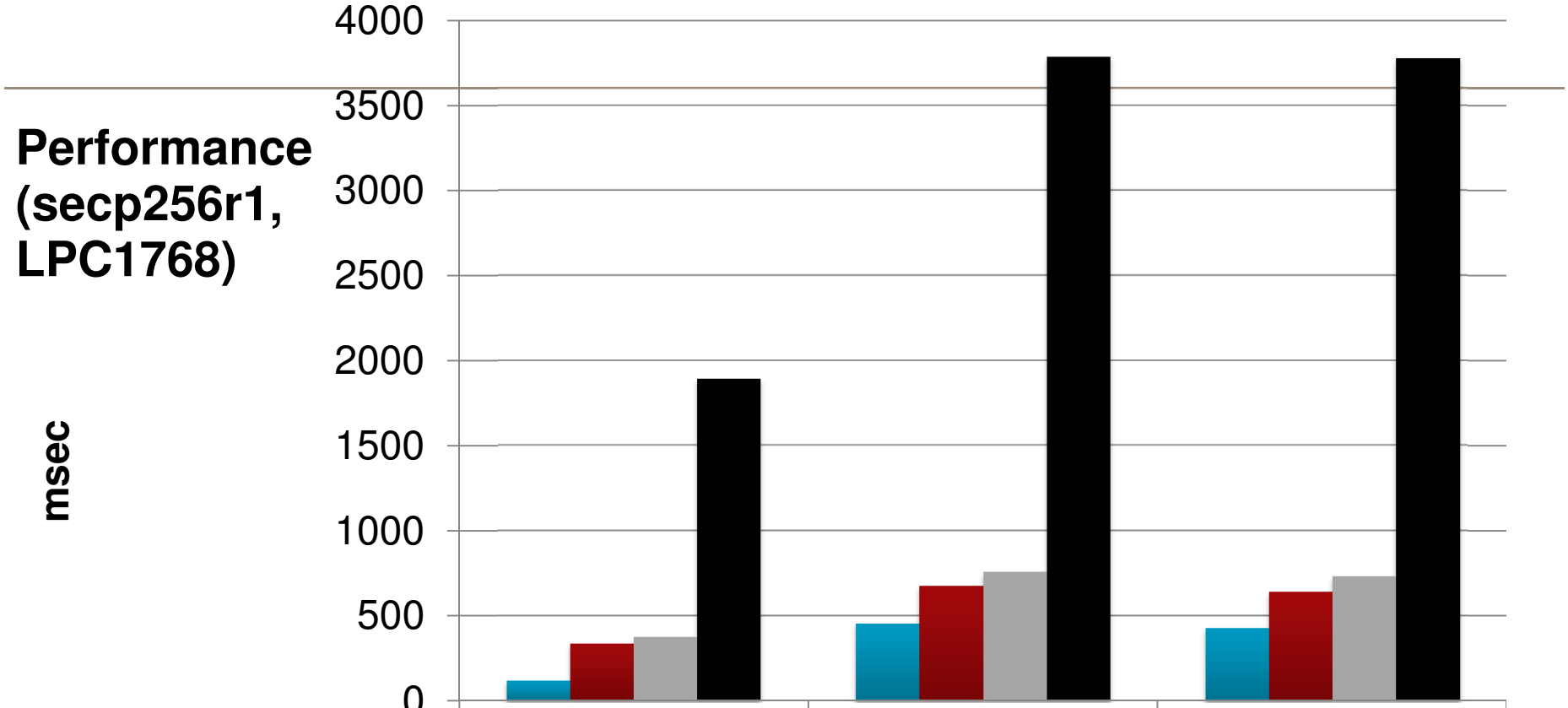
RAM Utilization

- To enable certain optimizations sufficient RAM is needed. A tradeoff decision between RAM and speed.
- Optimizations pays off.
- This slide shows heap usage (NIST optimization enabled).



	ECDSA-Sign	ECDSA-Verify	ECDHE
W=6, FP	4568	5380	5012
W=2, No FP	2972	3072	2692

Performance (secp256r1, LPC1768)



	ECDSA-Sign	ECDSA-Verify	ECDHE
w=6, FP, NIST	122	458	431
w=6, no FP, NIST	340	677	644
w=2, no FP, NIST	378	759	734
w=2, no FP, no NIST	1893	3788	3781

Using ~50 % more RAM increases the performance by a factor 8 or more.

Code Size

Position paper for the Smart Object Security Workshop 2013: http://www.lix.polytechnique.fr/hipercom/SmartObjectSecurity/

Implementation did not use mbed TLS/PolarSSL.

TABLE II

BINARY CODE SIZE FOR TLS-SPECIFIC CODE

Library Name	Code Size	Description
x509	2,776 bytes	The x509 related code (*x509.c*) provides functions to parse certificates, to copy them into the program internal data structures and to perform certificate related processing functions, like certificate verification.
ASN1 Parser	5,512 bytes	The ASN1 library (*asn1.c*) contains the necessary code to parse ASN1 data.
Generic TLS Library	15,928 bytes	This library (*tls1.c*) is separated from the TLS client specific code (*tls1_clnt.c*) to offer those functions that are common with the client and the server-side implementation. This includes code for the master secret generation, certificate validation and identity verification, computing the finished message, ciphersuite related functions, encrypting and decrypting data, sending and receiving TLS messages (e.g., finish message, alert messages, certificate message, session resumption).
TLS Client Library	4,584 bytes	The TLS client-specific code (*tls1_clnt.c*) includes functions that are only executed by the client based on the supported ciphersuites, such as establishing the connection with the TLS server, sending the ClientHello handshake message, parsing the ServerHello handshake message, processing the ServerHelloDone message, sending the ClientKeyExchange message, processing the CertificateRequest message.
OS Wrapper Functions	2,776 bytes	The functions defined in *os_port.c* aim to make development easier (e.g., for failure handling with memory allocation and various header definitions) but are not absolutely necessary.
OpenSSL Wrapper Functions	931 bytes	The OpenSSL API calls are familiar to many programmers and therefore these wrapper functions are provided to simplify application development. This library (*openssl.c*) is also not absolutely necessary.
Certificate Processing Functions	4,456 bytes	These functions defined in *loader.c* provide the ability to load certificates from files (or to use a during compile time), to parse them, and populate corresponding data structures.

Parts omitted by raw public key implementation

Size of TLS / DTLS 1.3 code?

Code size of DTLS profiles?

More details from the NIST Lightweight Cryptography Workshop 2015:
http://www.nist.gov/itl/csd/ct/lwc_workshop2015.cfm

ACE

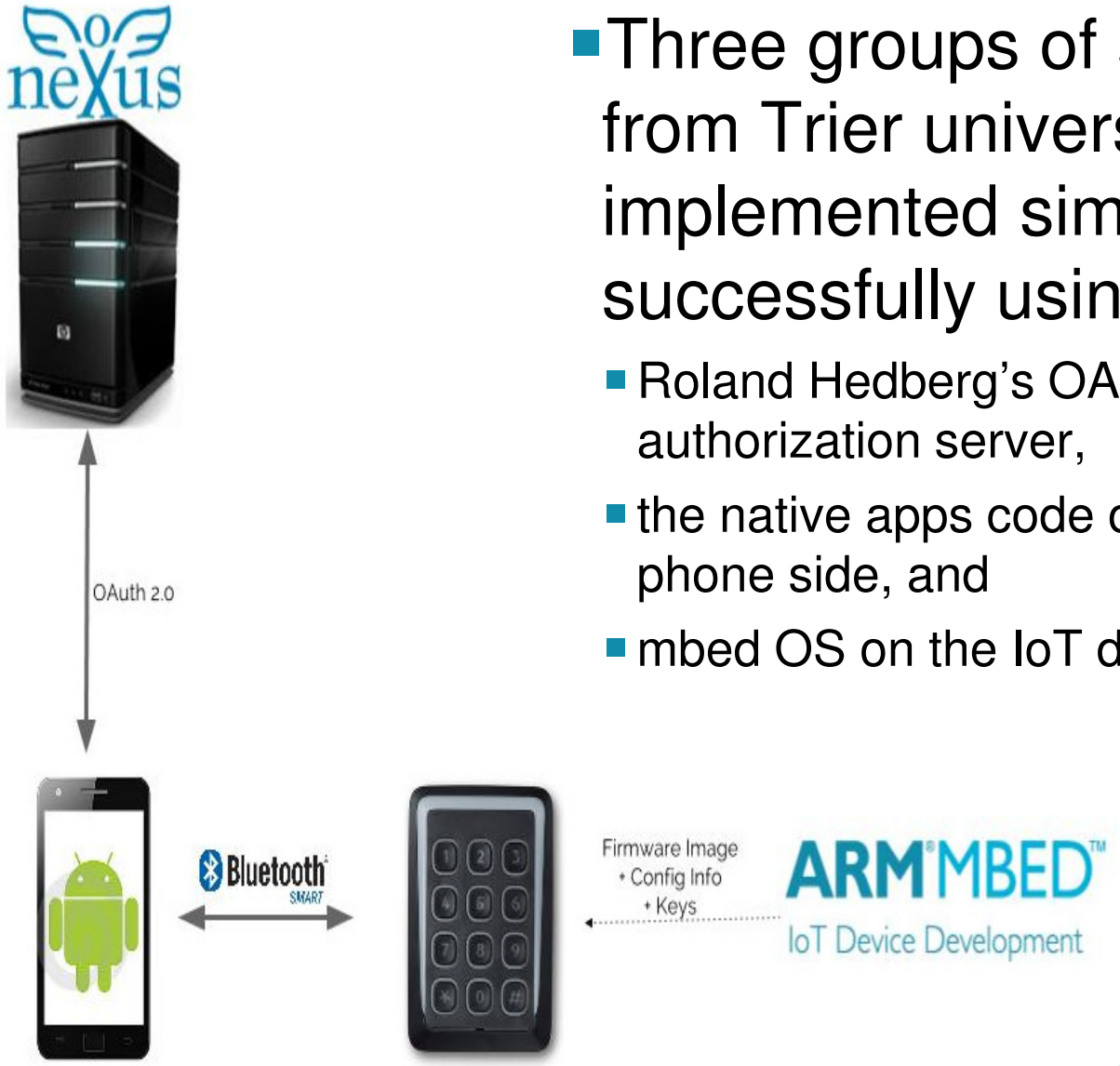
- ACE adds optimizations and profiles OAuth.

OAuth	ACE
JSON	CBOR
JOSE	COSE
JWT	CWT
HTTP	CoAP, MQTT, HTTP/2, …
TLS	DTLS
Bearer Tokens	PoP Tokens
User is often owner	User is often not the owner

PoP Tokens:

- **Symmetric**
- **Asymmetric**
- **Raw Public Key**

Demo Setup



- Three groups of students from Trier university implemented similar setup successfully using
  - Roland Hedberg's OAuth authorization server,
  - the native apps code on the mobile phone side, and
  - mbed OS on the IoT device.

Ongoing Work

- draft-ietf-ace-oauth-authz provides the framework (see open issues at https://github.com/LudwigSeitz/ace-oauth/)
- Group communication security (draft-somaraju-ace-multicast-01 and draft-hardjono-ace-fluffy-03)
- Privacy extensions (draft-cuellar-ace-pat-priv-enhanced-authz-tokens-03)
- Application layer security on top of CoAP (draft-selander-ace-cose-ecdhe-02 and draft-selander-ace-object-security-05).

Help Needed

- Implementation ongoing but more help appreciated (e.g., COSE, CWT, PoP tokens). Examples:
  - https://github.com/erwah/acetest
  - https://github.com/Gunzter/COSE-C
- Performance analysis
- Security analysis
- Use cases and demos