Smart Object Security Workshop Report

Jari Arkko

Ericsson

Hannes Tschofenig

NSN

Smart Object Security Workshop

http://www.lix.polytechnique.fr/hipercom/SmartObjectSecurity/
http://www.tschofenig.priv.at/sos-papers/PositionPapers.htm

- Held last Friday, March 23rd
- Hosted by Ecole Polytechnique & Thomas Clausen
- Organizers: Hannes Tschofenig, Jari Arkko, Carsten Bormann, Cullen Jennings, Zach Shelby, Peter Friess, Antonia Skarmeta, Thomas Clausen
- Participation by submission of a position paper
- 36 papers received

EUROPEAN COMMISSION
ÉCOLE POLYTECHNIQUE ParisTech
Laboratoire d'Informatique

Workshop Goals

- We had a gut feeling that we might have problems with securing smart object networks
- Bring together implementation experience, application requirements, and researchers and protocol designers
- What deployment experience is there? What credential types are most common? What implementation techniques make it possible to use Internet security technology in these devices? What are the challenges?

Requirements and Use Cases

Paul Chilton: Security challenges in the lighting use case

Rudolf van der Berg: Open interfaces, identifier spaces, and economic challenges

Implementation experience

Carsten Bormann: Light-weight COAP & DTLS implementations

Hannes Tschofenig: TLS and Raw Public Keys Implementation

Mohit Sethi: Public Key Crypto Implementation Experience

Authorization and Role-based Access Control

Richard Barnes: Beyond COMSEC

Jan Janak: On Access Control

Provisioning

Johannes Gilger: Secure pairing

Cullen Jennings: A deployment model

Summary

All slides at http://www.tschofenig.priv.at/wp/?p=874

Potential Conclusions

- There are serious attacks, this is not just a matter of kids from neighbor messing up your home automation
- A big challenge is setting up security when devices have very limited user interfaces and the installation is done by, e.g., normal people in their homes
- Different applications have very different requirements, e.g., individual users vs. 1 million device users
- There are examples of using standard Internet security protocols and algorithms in small devices; it is not clear if new protocol or algorithm work is needed
- The participants saw many challenges in setting up authorization and performing enrollment & pairing
- Here in LWIG we will focus mostly on the implementation experience, see the SAAG presentation for the other issues

Implementation Challenges

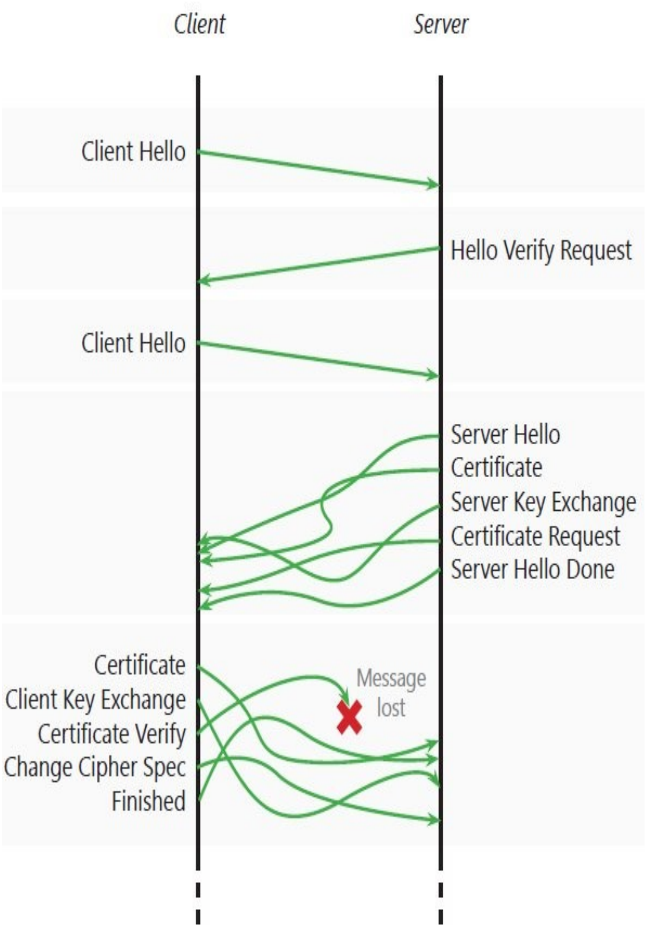
- The participants felt that existing algorithms are usable even for the smart objects
- The participants also felt that existing protocols are probably usable
  - Perhaps with some small extensions or changes in some cases
  - Enrollment & pairing is a big question mark
- The participants have done a lot of implementation work on TLS, DTLS, PANA, EAP, JOSE, and crypto algorithms
- But more work is needed

Implementation Challenges 2

- It is important to focus on the system – including all protocols, authorization, enrollment, configuration, management
- Implementation size and speed just for the pure crypto or protocol may often be misleading
- If optimization is needed, on what?
  - Speed of operations
  - Memory usage (RAM or ROM)
  - Power usage
  - Number of messages
  - Number of bits sent over the wireless interface
  - Time spent while waiting for packets to be received
- Metrics related to communication efficiency are probably more important than, say, ROM usage

DTLS on CoAP (Carsten et al)

- Presented one way to use DTLS with CoAP, along with numbers about the implementation size
- Generated a discussion on what is the right way to use DTLS with CoAP
- See Klaus' presentation here in LWIG for further information



Optimizing DTLS Implementions (Carsten et al.)

Code Size	Description
1429 Bytes	SHA-256
992 Bytes	CCM
9812 Bytes	DTLS state machine

TABLE I

CODE FOOTPRINT OF MINIMAL DTLS IMPLEMENTATION

Optimizing DTLS Implementions (Hannes)

- There is no free lunch
- Lower footprint means fewer functions or more assumptions
  - Example: if you strip your system down to pre-shared secrets and symmetric crypto, you may have the smallest footprint, but can the system be deployed?
- Decide what you really need, leave other things out

Optimizing DTLS Implementions (Hannes)

TABLE I

BINARY CODE SIZE FOR CRYPTOGRAPHY SUPPORT FUNCTIONS

Library	Code Size
MD5	4,856 bytes
SHA1	2,432 bytes
HMAC	2,928 bytes
RSA	3,984 bytes
Big Integer Implementation	8,328 bytes
AES	7,096 bytes
RC4	1,496 bytes
Random Number Generator	4,840 bytes

Optimizing DTLS Implementions (Hannes)

TABLE II
BINARY CODE SIZE FOR TLS-SPECIFIC CODE

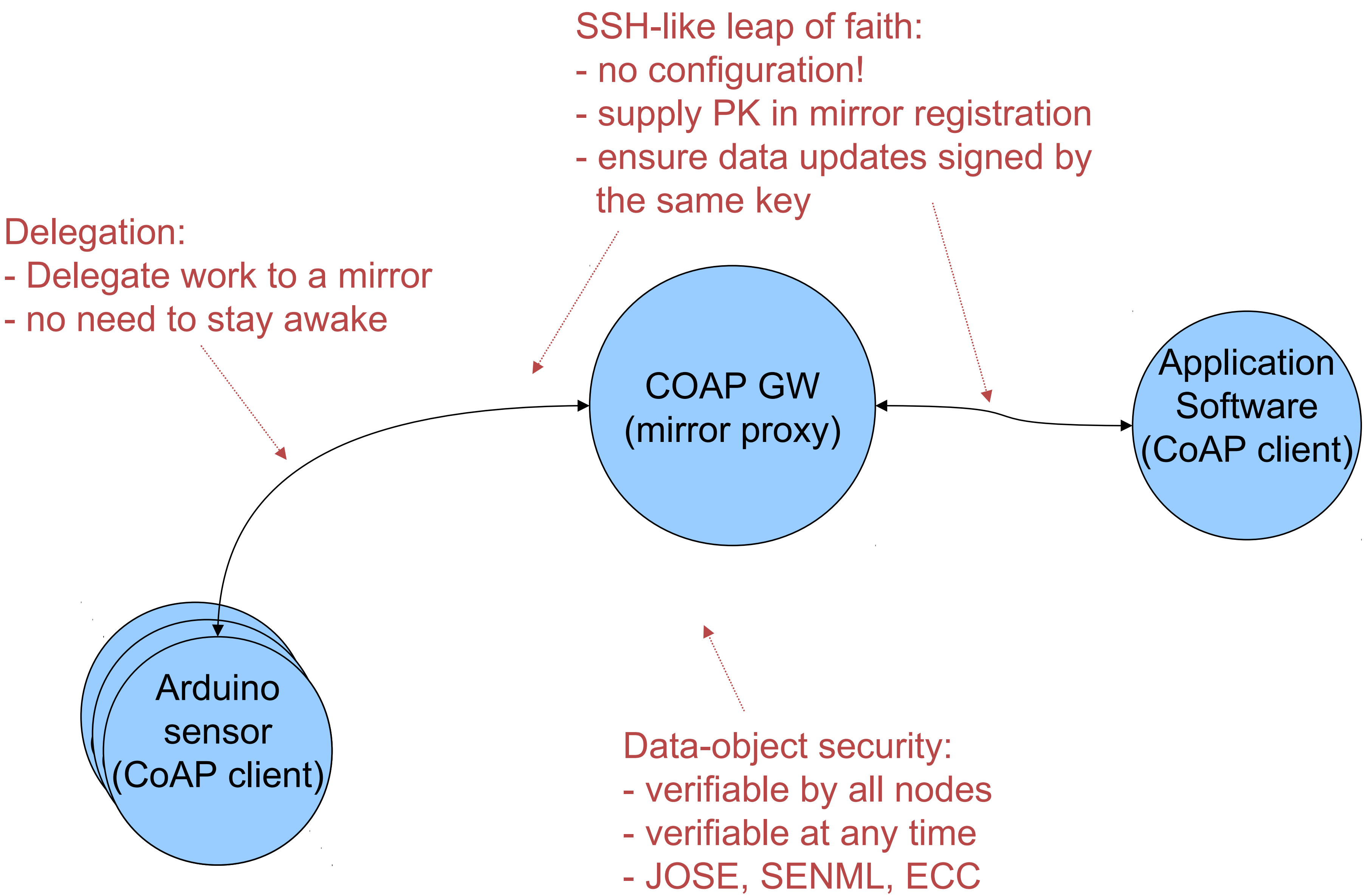
Library Name	Code Size	Description
x509	2,776 bytes	The x509 related code (*x509.c*) provides functions to parse certificates, to copy them into the program internal data structures and to perform certificate related processing functions, like certificate verification.
ASN1 Parser	5,512 bytes	The ASN1 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 default key as a static data structure embedded during compile time), to parse them, and populate corresponding data structures.

ECC and RSA on Arduino

Library	ROM
AvrCryptolib	3.6 KB
Wiselib	16.0 KB
TinyECC	18.0 KB
Relic	29.0 KB

Algorithm	Library	RAM	Time
RSA-512	AvrCryptolib	320 B	25.0 s
RSA-1024	AvrCryptolib	640 B	199.0 s
ECC 128r1	TinyECC	776 B	1.8 s
ECC 192k1	TinyECC	1008 B	3.4 s
NIST K163	Relic	2804 B	0.3 s ← ~ RSA 1024!
NIST K233	Relic	3675 B	1.8 s ← ~ RSA 2048!

Example Application



Possible IETF Work Items

LWIG:

- Documentation on making power-efficient and small implementations of DTLS/TLS, CoAP security, JOSE, crypto algorithms
- Without changing the protocols (just like all the other work in LWIG)

Elsewhere:

- Explain how to use DTLS with CoAP (CORE)
- Imprinting & enrollment protocols over the Internet?