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
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 Implementations
(Carsten et al.)

<table>
<thead>
<tr>
<th>Code Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1429 Bytes</td>
<td>SHA-256</td>
</tr>
<tr>
<td>992 Bytes</td>
<td>CCM</td>
</tr>
<tr>
<td>9812 Bytes</td>
<td>DTLS state machine</td>
</tr>
</tbody>
</table>

**TABLE I**

**Code footprint of minimal DTLS implementation**
Optimizing DTLS Implementations (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 Implementations (Hannes)

**TABLE I**

<table>
<thead>
<tr>
<th>Library</th>
<th>Code Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>4,856 bytes</td>
</tr>
<tr>
<td>SHA1</td>
<td>2,432 bytes</td>
</tr>
<tr>
<td>HMAC</td>
<td>2,928 bytes</td>
</tr>
<tr>
<td>RSA</td>
<td>3,984 bytes</td>
</tr>
<tr>
<td>Big Integer Implementation</td>
<td>8,328 bytes</td>
</tr>
<tr>
<td>AES</td>
<td>7,096 bytes</td>
</tr>
<tr>
<td>RC4</td>
<td>1,496 bytes</td>
</tr>
<tr>
<td>Random Number Generator</td>
<td>4,840 bytes</td>
</tr>
</tbody>
</table>
## Optimizing DTLS Implementations (Hannes)

### TABLE II

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Code Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x509</td>
<td>2,776 bytes</td>
<td>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.</td>
</tr>
<tr>
<td>ASN1 Parser</td>
<td>5,512 bytes</td>
<td>The ASN1 library (asn1.c) contains the necessary code to parse ASN1 data.</td>
</tr>
<tr>
<td>Generic TLS Library</td>
<td>15,928 bytes</td>
<td>This library (tls1.c) is separated from the TLS client specific code (tls1.cnt.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).</td>
</tr>
<tr>
<td>TLS Client Library</td>
<td>4,584 bytes</td>
<td>The TLS client-specific code (tls1.cnt.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.</td>
</tr>
<tr>
<td>OS Wrapper Functions</td>
<td>2,776 bytes</td>
<td>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.</td>
</tr>
<tr>
<td>OpenSSL Wrapper Functions</td>
<td>931 bytes</td>
<td>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.</td>
</tr>
<tr>
<td>Certificate Processing</td>
<td>4,456 bytes</td>
<td>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.</td>
</tr>
</tbody>
</table>
ECC and RSA on Arduino

<table>
<thead>
<tr>
<th>Library</th>
<th>ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvrCryptoLib</td>
<td>3.6 KB</td>
</tr>
<tr>
<td>Wiselib</td>
<td>16.0 KB</td>
</tr>
<tr>
<td>TinyECC</td>
<td>18.0 KB</td>
</tr>
<tr>
<td>Relic</td>
<td>29.0 KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Library</th>
<th>RAM</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-512</td>
<td>AvrCryptoLib</td>
<td>320 B</td>
<td>25.0 s</td>
</tr>
<tr>
<td>RSA-1024</td>
<td>AvrCryptoLib</td>
<td>640 B</td>
<td>199.0 s</td>
</tr>
<tr>
<td>ECC 128r1</td>
<td>TinyECC</td>
<td>776 B</td>
<td>1.8 s</td>
</tr>
<tr>
<td>ECC 192k1</td>
<td>TinyECC</td>
<td>1008 B</td>
<td>3.4 s</td>
</tr>
<tr>
<td>NIST K163</td>
<td>Relic</td>
<td>2804 B</td>
<td>0.3 s</td>
</tr>
<tr>
<td>NIST K233</td>
<td>Relic</td>
<td>3675 B</td>
<td>1.8 s</td>
</tr>
</tbody>
</table>

- ~ RSA 1024!
- ~ RSA 2048!
Example Application

Delegation:
- Delegate work to a mirror
- no need to stay awake

SSH-like leap of faith:
- no configuration!
- supply PK in mirror registration
- ensure data updates signed by the same key

Data-object security:
- verifiable by all nodes
- verifiable at any time
- JOSE, SENML, ECC
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?