# Simple Keys for Simple Smart Objects

Olaf Bergmann, Stefanie Gerdes, Carsten Bormann
Universität Bremen TZI
Email: {bergmann|gerdes|cabo}@tzi.org

## I. Introduction

During recent years, a working infrastructure for the *Internet of Things* (IoT, [1]) has evolved. This eco-system of networked devices is based on inexpensive but powerful hardware: microcontroller chips with low-power RF transmitters for license-free frequency bands, sufficient memory, and enough I/O ports to attach sensors and actuators. Moreover, various technical bodies have cooperated to foster open standards for *machine-to-machine* (M2M) communication between devices. Besides the basic communication infrastructure, foundations were laid to make objects *smart*, i. e. to create self-describing services that are offered by identifiable entities and use common data formats.

Like every node in a network, interconnected Smart Objects are susceptible to various threats, accentuated here by the special characteristics of wireless sensor networks (WSNs). As the nodes communicate wirelessly, conversations can easily be eavesdropped on, and fake messages can be injected into the network. Moreover, in many scenarios nodes are installed in places where physical access is easy. Thus, nodes may be damaged or captured. Therefore, specific attention must be given to protecting Smart Objects from attacks against data confidentiality, integrity, and service availability.

However, common security measures are not easily applicable to Smart Objects due to the constrained nature of the nodes. The memory of many devices, RAM as well as Flash memory (code space), is restricted. To gain a rough but usable terminology, Bormann clusters the large variety of devices into two groups of interest to this discussion [2]: Class 1 devices have a data (RAM) size of about 10 KiB and a code (Flash) size of about 100 KiB, whereas class 2 devices possess a data size of about 50 KiB and a code size of about 250 KiB.

The computational power of devices in most cases is limited by CPU clock rates of less than 25 MHz. Operating system support is often limited, e. g. no process scheduling is provided, so only one process at a time can run on each device. The achievable goodput within the network depends on the state of the wireless medium but has a low upper bound such as 250 kbit/s, the bit rate IEEE 802.15.4 chose in order to save energy. Most importantly, Smart Objects are often installed where they cannot easily be supplied with power and therefore run on primary batteries.

Because of these limitations, protection of Smart Objects against attacks is not easy to accomplish. Encryption mechanisms that have been used in other Internet-enabled environments to protect sensitive data often severely stress the abilities of the small devices. For instance, public-key cryptography requires a good amount of processing power and memory. (The way it has been used on the Web, it also needs a working certificate infrastructure to validate the peer's identity; it is not clear how such infrastructures will evolve for the requirements of the Smart Object space.)

Smart Object security will not come about by blindly copying the approaches that so far have worked in the "Big Things Internet". It is, however, also not necessary to reinvent the entire area of security for Smart Objects. The objective of our work is to obtain credible Smart Object security at a reasonable cost, which includes cost of design and deployment and therefore profits from re-using as many proven components of Internet technology as reasonably possible.

If the devices are simple, there is an even stronger requirement for simplicity on the usability side. The integrity of the life-savings of an individual are not going to be in danger when their neighbor manages to switch one of their lamps on and off. Still, some level of security is advised, and this is more likely to actually be deployed and used correctly if it can be made to work in simple steps. The most important attack scenario to be prevented in these environments is one of low-effort remote takeover of a large mass of Smart Objects.

## II. Background

In implementing these concepts, we can make use of the Constrained Application Protocol, CoAP [3]. The usage of CoAP in a real-world constrained node application has been described by Kuladinithi et al. [4]. Arkko et al. [5] present a very tiny implementation of CoAP with the aim to be as energy-efficient as possible.

The CoAP protocol will likely already have been deployed to support the actual application, so re-using it as the protocol to establish and retrieve security-related state will save code and simplify the overall architecture. CoAP's underlying architectural style, REST [6], fits well to many of the interactions required in key management.

As defined in the CoAP security model, we also assume that Datagram Transport Layer Security, DTLS [7] will be deployed and form the basis for the secure operation of the applications. (Alternatively IPsec/IKEv2 could be employed, which will be the subject of separate investigations.)

This paper will focus on using this protocol landscape to actually obtain our security objectives. It will also discuss how DTLS, which was developed with more capable devices in mind, can be used on constrained Smart Objects. The most vexing problem for making Smart Object security is key

management, which both has to live with the limits of the constrained devices and networks and has to enable security workflows that are optimized in a much more radical fashion for simplicity and ease of use than those in the Big Things Internet.

When examining the security processes and the different actors and roles in these processes, two distinguishable phases are almost always apparent:

- bootstrapping, and
- operational use.

The term *bootstrapping* in our work is derived from its use since about 2005 in the 6LoWPAN working group of the IETF. O'Flynn [8] and Sarikaya et al. [9] define it as the process of configuring a node to enable it to participate in normal network operation. But maybe more strikingly, in general usage the process of bootstrapping means setting up a complicated software environment in several steps, starting from something very primitive (such as a diode-matrix ROM bootstrap in early computing). Transferring this notion to security bootstrapping, and combining it with O'Flynn's definition, it might stand for the setup of robustly secure node configuration using some primitive initial keying materials and security procedures.

Various protocols that could be used for secure bootstrapping are listed by Sarikaya et al. [9]. Garcia-Morchon et al. [10] also analyze typical threats to wireless sensor networks and discuss approaches to secure bootstrapping in general. An overview of applicable security technologies is provided by Arkko et al. [11] together with a proposal for generating device identifiers for constrained nodes.

All of these publications address only certain aspects of the bootstrapping process and do not provide a concise description of the required CoAP vocabulary to use. We have decided to combine these building blocks to form a complete bootstrapping sequence.

For now, we are focusing on a situation without a strong integration of manufacturing, device delivery logistics, and on-site installation management. Instead, our scenario is based on an individual buying some nodes at a supermarket and then integrating them into their family's home network. We are not assuming this process is aided by global CAs/certificate chain; likewise, the DNS system will not be able to contribute much here. On the other hand, the scenario provides some simplification as we only need mutual trust between devices, which we can derive from basic *mother-duckling* relationships.

The duckling model we adapted for imprinting security credentials has been described by Stajano and Anderson [12], [13], and Bormann has described a method to initiate discovery in a CoAP network [14]. A more general overview over security issues in wireless sensor networks can be found in [15].

## III. Bootstrapping Node Security

For inexpensive Class 1 nodes that offer very limited user interfaces such as a couple of LEDs and maybe push buttons, distribution of credentials and validation of remote peers' identities is one of the biggest challenges for good security. Although in some areas there will be trusted platforms with tamper-proof unique identities, many application scenarios in our opinion will rely on other ways to provision cryptographic identities onto networked devices.

In [16], the authors present a protocol to bootstrap device configuration that enables a new network node to enter an existing security domain. To do so, they make use of the imprinting mechanism described in [12]. A new node that wants to participate in network communication (a *duckling*) thus must be imprinted with keying material before it will be granted access to the network. Once imprinting has succeeded, sufficient keying material is available to establish a secure channel with the *mother* node to provide additional configuration.

The bootstrapping protocol is split into three distinct phases each of which is intended to improve overall security of the system. It is crucial for this approach that a third party (e. g. a user who owns both the mother and the duckling) is able to authenticate both peers. To do so, the parties to authenticate must give audible or visible feedback when the keying material is imprinted. In this phase, the communication is specifically prone to man-in-the-middle attacks. Due to the lack of credentials at this step, it cannot be secured by cryptographic means. Instead, a short window of initial vulnerability is deemed acceptable.

This vulnerability could be mitigated by securing the communication physically, e. g. by a Faraday cage that shields mother and duckling from their environment. More likely, placing the nodes close and limiting the RF power and receiver sensitivity to a minimum might reduce the attack cross-section enough to thwart wide-spread attacking.

The imprinting phase is usually initiated by a user action on the duckling node, e. g. by pushing a button or by inserting batteries. The duckling then searches for network components that can act as mother node. With CoAP, this can be done by `POST` requests that are sent to a specific multicast group address or to addresses found in the ducklings IPv6 neighbor cache. As mother and duckling are anticipated to be on the same network link (for wireless communication, they should be within range of their RF senders), few `POST` requests will suffice to discover the mother duck.

Triggered by this `POST` request, the mother imprints a shared secret onto the duckling and establishes a secure channel as soon as it has received a positive response on the imprinting request. Additional configuration information including e. g. the network prefix and the address of the default gateway can then be provided.

With this three-step protocol (*discover*, *imprint*, *configure*) new nodes can be brought into an existing network easily without prior provisioning of keying material. The bootstrapping protocol itself is implemented as a CoAP application to avoid the need for another protocol stack in the code space of the constrained nodes.

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

TABLE I
CODE FOOTPRINT OF MINIMAL DTLS IMPLEMENTATION

## IV. A LIGHTWEIGHT DTLS IMPLEMENTATION

The CoAP specification [3] offers three operation modes for DTLS in constrained networks: *PreSharedKey* uses symmetric key cryptography and requires only little processing overhead. *RawPublicKeys* and *Certificates* use asymmetric ciphers and thus provide advanced security features. As there are many production-quality implementations for less-constrained devices (regarding CPU power and memory), these operation modes can be seen as mature and well understood.

Where resource usage is a concern, security parameters should be tuned to enable lightweight implementations without sacrificing the overall level of protection that can be achieved. Of course, a trade-off exists between efficiency and the resulting security level.

As a proof of concept and to feed the ongoing discussion of which parameters to adjust for the use of CoAP in constrained networks, we have implemented the bootstrapping protocol proposed in Section III for constrained nodes of Class 1. The hardware is based on ST Microelectronics `STM32W108` with 128 KiB flash memory and 8 KiB RAM, featuring ARM Cortex M3 CPU architecture. It is equipped with an IEEE 802.15.4 RF transmitter and includes hardware for AES-128 encryption. The Contiki operating system [17] provides support for several boards of the STM32 chip family as platform `mbxxx`.[1] The CoAP protocol is handled by the open-source library `libcoap`[2].

Our DTLS implementation is based on the open-source library tinyDTLS[3], ported to Contiki. We have replaced the existing cipher suite of tinyDTLS by our implementation of `TLS_PSK_WITH_AES_128_CCM_8` defined in [18]. Providing Authenticated Encryption with Associated Data (AEAD, [19]), this cipher suite combines payload encryption and message authentication. The base cryptographic primitives required thus are only AES-128 encryption and SHA-256 for the pseudo random function (PRF) of DTLS 1.2 [7]. Limiting the key exchange options to pre-shared keys only, no certificate exchange during DTLS handshake is required.

The resulting code size for these primitives and the DTLS handshake protocol is shown in Table I.

The DTLS state machine comprises the cipher suite implementation as well as the DTLS record and handshake protocol for both client and server functionality. For efficiency, we have decided not to implement DTLS message fragmentation and have limited the alert protocol to a minimum.

[1]See http://www.contiki-os.org/ for more information on the Contiki OS.
[2]See http://libcoap.sourceforge.net.
[3]See http://tinydtls.sourceforge.net.

These numbers can certainly be optimized, but already indicate that a basic DTLS implementation is feasible with less than 10 KiB of code plus 1.4 KiB for the hash underlying the PRF. Depending on the actual cipher suites selected, only little extra code is needed. For example, the overhead for CCM is less than 1 KiB when AES encryption is in hardware, resulting in approximately 12 KiB for the entire DTLS support.

The modest code size in this case is helped a lot by the use of AEAD in combination with hardware AES-128 encryption. Although improved error handling might add some code to the implementation of the DTLS state machine, our results demonstrate that DTLS is viable for communication security in constrained nodes and networks. However, our optimizations lose compliance with [7]. Future standardization should define a minimal DTLS profile for constrained nodes, simplifying error handling, removing the need to support fragmentation at the DTLS layer, and limiting mandatory-to-implement cipher suites to AEAD.

## V. CONCLUSIONS

We have presented an approach for Smart Object security based on security bootstrapping without pre-provisioned credentials and a basic implementation of DTLS. We are in the process of gaining experience with this approach in a lighting control scenario. In our implementation, we plan to optimize tinyDTLS further, to improve its robustness, and to derive some results on which are the most promising elements of the protocol that may be mandatory to implement in the standard but can be left out when needing a constrained implementation.

For work targeting a more managed deployment scenario, we are also interested in the implementation requirements for making asymmetric cryptography available to CoAP/tinyDTLS nodes: this has become more accessible by the recent work on using raw public keys instead of fully-built X.509 certificates in the DTLS handshake [20].

Finally, our prototype should give us experience in developing suitable user interfaces for inexpensive *mother* devices. With the ubiquity of relatively powerful smartphones, these are becoming the home automation remote control of choice and can be used to present relatively complex configuration and security setup operations in a simple, familiar user interface.

## REFERENCES


[1] K. Ashton. (2009, Jun. 22) That 'Internet of Things' thing. RFID Journal. [Online]. Available: http://www.rfidjournal.com/article/view/4986

[2] C. Bormann, "Guidance for Light-Weight Implementations of the Internet Protocol Suite," Internet-Draft (work in progress), Jan. 2012. [Online]. Available: http://tools.ietf.org/html/draft-bormann-lwig-guidance

[3] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained Application Protocol (CoAP)," Internet-Draft (work in progress), Nov. 2012. [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-coap

[4] K. Kuladinithi, O. Bergmann, T. Pötsch, M. Becker, and C. Görg, "Implementation of CoAP and its Application in Transport Logistics," in *Proc. of 'Extending the Internet to Low power and Lossy Networks' (IP+SN 2011)*, Chicago, 11. Apr. 2011. [Online]. Available: http://hinrg.cs.jhu.edu/joomla/images/stories/coap-ipsn.pdf

[5] J. Arkko, H. Rissanen, S. Loreto, Z. Turanyi, and O. Novo, "Implementing Tiny COAP Sensors," Internet-Draft (work in progress), Jul. 2011. [Online]. Available: http://tools.ietf.org/id/draft-arkko-core-sleepy-sensors

[6] R. Fielding, "Representational State Transfer (REST)," *Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine*, 2000.

[7] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347 (Proposed Standard), Internet Engineering Task Force, Jan. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6347.txt

[8] C. O'Flynn, "Initial Configuration of Resource-Constrained Devices," Internet-Draft (work in progress), Jan. 27, 2010. [Online]. Available: http://tools.ietf.org/html/draft-oflynn-6lowapp-bootstrapping

[9] B. Sarikaya, Y. Ohba, R. Moskowitz, Z. Cao, and R. Cragie, "Security Bootstrapping of Resource-Constrained Devices," Internet-Draft (work in progress), Jun. 2011. [Online]. Available: http://tools.ietf.org/html/draft-sarikaya-core-sbootstrapping

[10] O. Garcia-Morchon, S. Keoh, R. Hummen, and R. Struik, "Security Considerations in the IP-based Internet of Things," Internet-Draft (work in progress), Jul. 2011. [Online]. Available: http://tools.ietf.org/html/draft-garcia-core-security

[11] J. Arkko and A. Keranen, "CoAP Security Architecture," Internet-Draft (work in progress), Jul. 2011. [Online]. Available: http://tools.ietf.org/html/draft-arkko-core-security-arch

[12] F. Stajano and R. Anderson, "The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks," in *Security Protocols*, ser. Lecture Notes in Computer Science, B. Christianson, B. Crispo, J. Malcolm, and M. Roe, Eds. Springer Berlin / Heidelberg, 2000, vol. 1796, pp. 172–182. [Online]. Available: http://dx.doi.org/10.1007/10720107_24

[13] ——, "The Resurrecting Duckling: security issues for ubiquitous computing ," *Computer*, vol. 35, no. 4, pp. 22–26, Apr. 2002.

[14] C. Bormann, "CoRE Simple Server Discovery," Internet-Draft (work in progress), Mar. 7, 2011. [Online]. Available: http://tools.ietf.org/html/draft-bormann-core-simple-server-discovery

[15] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *Commun. ACM*, vol. 47, pp. 53–57, June 2004. [Online]. Available: http://doi.acm.org/10.1145/990680.990707

[16] O. Bergmann, S. Gerdes, S. Schäfer, F. Junge, and C. Bormann, "Secure Bootstrapping of Nodes in a CoAP Network," in *WCNC 2012 Workshop on Internet of Things Enabling Technologies, Embracing Machine-to-Machine Communications and Beyond*, Apr. 2012, to be published.

[17] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki – A Lightweight and Flexible Operating System for Tiny Networked Sensors," *Annual IEEE Conference on Local Computer Networks*, pp. 455–462, 2004.

[18] D. McGrew and D. Bailey, "AES-CCM Cipher Suites for TLS," Internet-Draft (work in progress), Feb. 2012. [Online]. Available: http://tools.ietf.org/html/draft-mcgrew-tls-aes-ccm

[19] D. McGrew, "An Interface and Algorithms for Authenticated Encryption," Internet Engineering Task Force, Jan. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc4627.txt

[20] P. Wouters, J. Gilmore, S. Weiler, T. Kivinen, and H. Tschofenig, "TLS Out-of-Band Public Key Validation," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-oob-pubkey-01, Jan. 2012, work in progress. [Online]. Available: http://tools.ietf.org/html/draft-ietf-tls-oob-pubkey