Author(s): Mohit Sethi, Ari Keränen and Jari Arkko

Title: End-to-end Security for Sleepy Smart Object Networks

Year: 2012

Version: Post print

Please cite the original version:

Mohit Sethi, Ari Keränen and Jari Arkko. End-to-end Security for Sleepy Smart Object Networks. In Proceedings of the 37th IEEE Conference on Local Computer Networks Workshops (LCN Work- shops), Clearwater, FL, pp. 964-972, ISBN 978-1-4673-2130-3, October 2012. DOI: 10.1109/LCNW.2012.6424089

End-to-end Security for Sleepy Smart Object Networks

Mohit Sethi*†‡, Jari Arkko*, Ari Keränen*
*NomadicLab, Ericsson Research
†Aalto University, Helsinki
‡Royal Institute of Technology, KTH, Stockholm
mohit@kth.se, jari.arkko@ericsson.com, ari.keranen@ericsson.com

***Abstract*—Internet of Things (IoT) refers to an inter-connected world where physical devices are seamlessly integrated into the Internet. The emergence of technologies such as Zigbee, Bluetooth low energy, and embedded sensors has transformed simple physical devices into smart objects that can understand and react to their environment. Such smart objects form the building blocks for the Internet of Things.**

Although the need for security is widely accepted, there is no clear consensus on how IP-based Internet security protocols can be applied to resource-constrained smart object networks. In this paper, we develop a new secure and energy-efficient communication model for the Constrained Application Protocol (CoAP), a light-weight communication protocol designed for smart object networks. This architecture and the communication model ensures data integrity and authenticity over a multi-hop network topology. It provides a mirroring mechanism that uses a proxy to serve data on behalf of sleeping smart objects, thereby allowing them to act as always-online web servers. A working prototype implementation of the architecture is also developed.

The security features in the architecture presented in this paper are based on using strong public-key cryptography. Contrary to popular belief, our performance evaluation shows that asymmetric public-key cryptography can be implemented on small 8-bit microcontrollers without modifying the underlying cryptographic algorithms using public libraries.

***Index Terms*—IoT, smart objects, security, CoAP, asymmetric cryptography, integrity, authenticity.**

I. Introduction

The term *Internet of Things (IoT)* was first coined by the MIT Auto-ID center [1] which had envisioned a world where every physical object is tagged with an RFID tag having a globally unique identifier. This would allow for tracking of objects in real-time and querying of data about them over the Internet. However, since then, the meaning of the Internet of Things has expanded and now encompasses a wide variety of technologies, objects and protocols. With the progress made in near field communication and sensor technology, the physical objects no longer act as unresponsive nodes but rather understand and react to the environment they reside in. Such objects, referred to as *smart objects*, form the building blocks of the Internet of Things.

Security is an important consideration in all modern communication systems. Since a wide variety of actors are involved in the manufacturing, installation and actual use of smart objects, the security challenges associated with a network of such objects is more perplexing than the Internet. Smart objects need to be protected against a wide variety of attacks during their lifecycle. For example, during the provisioning phase, an adversary may be able eavesdrop and obtain keying materials, security parameters, or initial settings if they are exchanged in the clear over a wireless medium. It can be non-trivial to perform device authentication since smart objects usually do not have a priori knowledge of each other and cannot always differentiate malicious network nodes from innocent neighbors via completely automated mechanisms.

Another factor that plays an important role while designing security solutions for smart objects is the resource-constrained nature of these devices. The devices not only have a small amount of memory and computational power but also a minimalistic energy supply available to them. Therefore, in many circumstances, the devices need to sleep for long periods in order to save energy and can wake up only for short periods to report sensor data. Such smart objects cannot afford to stay online for long durations to be polled data or support computationally intensive security protocols. We believe that extremely resource-constrained "sleepy" smart objects would form a large part of the deployment space and need appropriate treatment in terms of security and delegation mechanisms. Our work has the following contributions:

- Designing a standards-compliant security architecture for smart objects with an energy-efficient communication model.
- Developing a prototype based on the new architecture.
- Implementing and evaluating the performance of asymmetric public-key cryptography on 8-bit architectures.

The rest of the paper is organized as follows. In section II, we present our work on public-key cryptography on smart objects. The background information required to understand our proposed architecture is provided in section III. The entire architecture and its functioning is elucidated in section IV. In section V, we describe the proof-of-concept prototype developed for this architecture. Finally, section VI provides a summary of the paper and examines the potential future work in this area.

II. Asymmetric Cryptography in IoT

There are several proposals that attempt to secure smart object networks with non-public symmetric-key based authentication and key distribution mechanisms [2], [3], [4].

The underlying assumption being that public-key cryptography is too resource intensive for implementation on constrained devices. In symmetric-key based approaches, if an individual key is used for every node in a network of n nodes, then each node is required to store $n-1$ keys. Although this provides strong resilience against individual node compromise, it also leads to scalability issues making it undesirable for large networks. Conversely, when a single symmetric key is used for all the nodes in the network, the memory requirement for each node is reduced, but the resilience of the network also decreases.

In response to these problems, many probabilistic key distribution schemes for symmetric-key algorithms have been proposed [5], [6]. These schemes either need a pre-distribution of keys with complex configuration during provisioning, or large amount of network traffic that results in higher energy consumption. In general, symmetric-key schemes do not offer the flexibility of not having pre-shared keys that is provided with public-key asymmetric cryptography. Moreover, there are several studies that contradict the aforementioned assumption of feasibility of public-key cryptography on smart objects. We discuss some of the previous work related to public-key cryptography on smart objects in the next sub-section.

A. Previous Experiments with Asymmetric Cryptography

Gura et al. [7] carry out a performance comparison of RSA and ECC based public-key cryptographic schemes on 8-bit microcontrollers. They include several well known mathematical optimizations for improving the performance. The modular multiplication algorithm for RSA and ECC used a hybrid approach comprising of row-wise and column-wise schoolbook multiplication to increase the performance efficiency while keeping the SRAM consumption small. The authors were able to achieve 1024-bit RSA private key operation with exponent $e = 2^{16} + 1$ in 0.43 seconds and 160-bit elliptic curve point multiplication in 0.81 seconds on an 8-bit microcontroller with a clock speed of 8 MHz.

Blaß and Zitterbart [8] implement elliptic curve public-key cryptography on an 8-bit ATmega128 microcontroller. By using several optimizations they were able to achieve ECDSA based signature generation in 6.88 seconds. Besides the relatively slow performance, it is demonstrated that the 112-bit ECDLP can be solved in 3.5 months using 200 PlayStation 3 game consoles [9], making 113-bit elliptic curves vulnerable to attacks as well. In general, it is suggested that applications use 128-bit curves or higher to ensure sufficient security.

The work done by Uhsadel et al. [10] accomplishes a standards-compliant 160-bit secp160r1 curve based signature generation operation in 2881 cycles (0.39 seconds) on an 8-bit 8 MHz microcontroller. This is faster than the work done by Gura et al. [7] discussed earlier. The authors claim that the work provides the fastest known implementation of 160-bit elliptic curve point multiplication (in 2007).

Toheed and Razi [11] evaluate the feasibility of asymmetric public-key cryptography on the Contiki Operating System[1]. Their thesis provides a comprehensive performance comparison of two libraries, namely *Libtomcrypt* [12] and *Relic* [13] on the MSP430F1612 microcontroller [14] and on the COOJA simulator [15]. The authors evaluate ECC based signature generation and verification performance in terms of execution time, SRAM/ROM usage and energy consumption.

Gupta et al. [16] not only perform public-key cryptography on 8-bit platforms but also implement an entire web stack with a fully functional Secure Sockets Layer (SSL) [17] suite based on ECC. The implementation discussed can perform an entire SSL handshake on 8-bit 8 MHz microcontroller with 4 kB of SRAM in one second and can communicate 1 kB of application data over SSL in 0.4 seconds. This allows the sensors to be monitored and controlled remotely over the web without compromising on end-to-end security. The authors claim that the implementation provides the world's smallest secure web server (in 2005).

There have been several other works that have measured the energy efficiency of asymmetric cryptography on small platforms [18], [19]. With all the works presented thus far, there is a strong argument against the assumption that public-key cryptography is too resource intensive for execution on small platforms without changing the underlying cryptographic algorithms. However, most of the research work that was presented, did not have code in the public domain. Therefore, we set out to find code that is easily available online for use on 8-bit platforms and evaluate its performance.

B. Available Cryptographic Libraries

We provide a brief description of the libraries we found were suitable for such platforms and that were publicly available:

- AvrCryptolib [20]: Provides a variety of symmetric-key algorithms such as DES, Triple DES, AES, and RSA as an asymmetric public-key algorithm. The library only performs modular exponentiation and does not provide standardized padding such as PKCS #1 version 1.5 [21]. Parts of RSA algorithm were written in AVR 8-bit assembly language to reduce the execution time. The library also provides an option to store the keys in flash and access them directly from there, thus, saving the amount of SRAM consumed.
- Relic [13]: Written entirely in C, it provides a highly flexible and customizable implementation of a large variety of cryptographic algorithms. This not only includes RSA and ECC, but also pairing based asymmetric cryptography. Relic provides an option to build and include only the desired components for the required platform. While building the library, it is possible to select a variety of mathematical optimizations that can be combined to obtain the desired performance. Relic can be customized to use different bit-length words thus making it easy to compile for different architectures.
- TinyECC [22]: Designed for using elliptic curve cryptography on smart objects. It is written in the nesC programming language and is designed for specific use on TinyOS. However, the library can be ported to standard

[1]The Contiki OS, http://www.contiki-os.org/

TABLE I
RSA PRIVATE-KEY MODULAR EXPONENTIATION PERFORMANCE WITH AVRCYRPTOLIB

RSA Key Length (bits)	Execution Time (ms): Keys in SRAM	Memory footprint (bytes): Keys in SRAM	Execution Time (ms): Keys in ROM	Memory footprint (bytes): Keys in ROM
64	64	40	69	32
128	434	80	460	64
256	3516	80	3818	64
512	25,076	320	27,348	256
1,024	199,688	640	218,367	512
2,048	1,587,567	1,280	1,740,258	1,024

C99 by rewriting parts of the code. This allows for the library to be used on platforms that do not have TinyOS running on them. The library includes a wide variety of mathematical optimizations such as sliding window and Barrett reduction. While TinyECC only performs curves over prime fields, Relic performs curves over prime as well as binary fields.

- Wiselib [23]: Wiselib is a generic library written for sensor networks containing a wide variety of algorithms. It includes algorithms for routing, cryptography, localization, topology management. The library was designed to easily interface it with operating systems such as iSense. However, since the library is written entirely in C++ with a template based model similar to Boost/CGAL, it can be used on any platform directly without using any of the operating system interfaces provided. Similar to TinyECC, it implements curves over prime fields only. Wiselib does not implement many of the well known theoretical performance enhancement features.

C. Performance Analysis

For experimenting with public-key cryptography on resource-constrained platforms, we chose Arduino Uno board[2]. Arduino Uno has an ATmega328P (an 8-bit microcontroller with a clock rate of 16 MHz), 2 kB of SRAM, and 32 kB of flash memory. We specifically chose an 8-bit platform to demonstrate that our security architecture can be implemented even on very constrained platforms.

We have summarized the initial results of raw RSA private-key modular exponentiation performance using AvrCryptolib in Table I. The keys were generated separately with the value of public exponent as 3 and were hard coded into the program. The performance was faster for smaller key lengths as was expected. With longer keys the execution time increased exponentially. We performed two different sets of experiments for each key length. In the first case, the keys were loaded into the SRAM from the flash memeory before they were used by any of the functions. In the second case, the keys were addressed and used directly from the flash memory. As was expected, the second case used less SRAM but led to slightly longer execution time. The SRAM values indicate RSA processing only and do not reflect the SRAM consumption of the entire program. The execution times were calculated as mean of five consecutive experiments.

It is worth noting that the implementation performs basic modular exponentiation without mathematical optimizations used by Gura et al. [7]. With more SRAM, we believe that 1024/2048-bit RSA operations can be performed in much less time as has been shown elsewhere.

In Figure 1 we present the results from the performance comparison of ECDSA signature generation for all the libraries surveyed. Although we tested the libraries with various different curve parameters providing different levels of security, we only present the performance of curves with 80-bit symmetric-key equivalent security. The libraries were tested with National Institute of Standards and Technology (NIST) and Standards for Efficient Cryptography Group (SECG) standardized curve parameters provided in the libraries. All the libraries provide a SHA-1 hash algorithm, which was used in our performance analysis. While performing experiments on the Relic library, we were unable to use the Arduino Uno board because of memory constraints. However, since the library looked promising from previous experiments [11] we chose to test it on Arduino Mega[3] which has similar microcontroller as the Uno but has more SRAM (8 kB) and ROM (128 kB).

In order to measure the SRAM consumption for each of the curves we used the Avrora simulator [24]. Since all the libraries and our code use only stack based allocation, the stack trace produced by the simulator gives an accurate value for the memory footprint. The execution times were calculated on Arduino boards using the on-board ATmega internal oscillator which provides an accuracy of four microseconds. We use the rand() function provided in the Arduino library for creating random numbers. We performed all the experiments with a common seed (300) to the pseudo-random number generator so that the results are reproducible.

While Wiselib used the least amount of SRAM among all the libraries, its execution times for signature generation were the highest. In comparison, TinyECC was able to perform signature generation for the same curve parameters in times ranging from 2.2 to 2.5 seconds with 50 bytes of extra SRAM. TinyECC provides several optimizations which include Barrett reduction [25], hybrid multiplication and squaring [7], curve specific optimizations [7], projective coordinate system [26], sliding window for scalar multiplication [26], and Shamir trick [26]. We however only use the projective coordinate system, sliding window optimization for scalar multiplication and SECG curve specific optimizations for our experiments.

[2]Arduino Uno, http://arduino.cc/en/Main/arduinoBoardUno

[3]Arduino Mega, http://arduino.cc/en/Main/ArduinoBoardMega

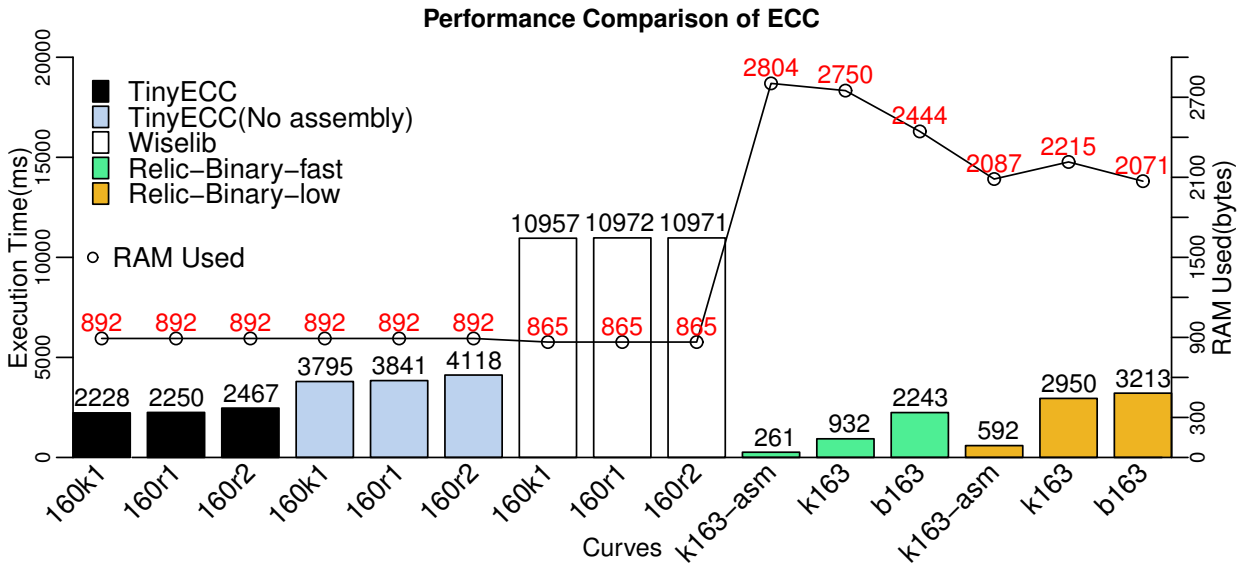

Fig. 1. ECC Performance

While we did not include Shamir trick as it helps only during verification, hybrid multiplication and squaring could not be ported to Arduino Uno as they were coded in incompatible assembly. Liu and Ning [22] demonstrate that Barrett reduction is only efficient when used with hybrid multiplication. Since we could not port hybrid multiplication, we also chose to exclude Barrett reduction from our code. Finally, we used assembly code only for simultaneous multiplication with addition and simultaneous multiplication with subtraction which were compatible with Arduino platforms.

Both the libraries implemented three different SECG curves over prime fields (secp160k1, secp160r1 and secp160r2 depicted as 160k1, 160r1 and 160r2 respectively) and their execution times and SRAM consumption are depicted in Figure 1. We experimented TinyECC with and without assembly optimizations. It was interesting to observe here that the assembly version consumed the same amount of SRAM but saved about 1.5 seconds of execution time for each of the three curves tested. It is evident that using assembly optimizations for the target platform can result in improved performance.

For Relic we tested curves with equivalent security over binary fields only. Though Relic implements curves over prime fields as well, previous work [27], [28] shows that binary fields are more suited for Atmel platforms used on Arduino boards. We tested two different configurations of the library for curves over binary fields: one which resulted in the least execution time and another which consumed the least amount of memory[4]. We tested two standard NIST curves over binary fields (NIST-K163 and NIST-B163 shown as k163 and b163 respectively). While experimenting with Koblitz curves, we tested a special case where the binary arithmetic was implemented in assembly (shown as k163-asm). As was expected, the assembly versions performed significantly faster than the non-assembly versions.

We intentionally did not focus any of the experiments towards the flash memory consumption of the libraries as it is available at low cost and is generally not the limiting factor on constrained devices. However, as a rough estimate Wiselib consumed about 17 kB of ROM, TinyECC consumed about 20 kB of ROM, while relic consumed 27-40 kB of ROM depending on the configuration. The TinyECC values differ from those in [22] as we no longer use the original nesC code and re-wrote the library in standard C99.

From these results it is clear that software implementation of public-key cryptography on smart objects is not only possible, but can be efficient with publicly available code. There is scope further performance enhancement and we believe, with increase in computational power and cost reduction, it will be even more trivial to achieve basic public-key cryptography on such devices. We did not measure signature verification times since our architecture described later, does not require verification on such devices.

III. Design

In order to understand the proposed security architecture, we first introduce a few basic concepts:

- CoAP: Constrained Application Protocol (CoAP) [29] is a RESTful application layer protocol designed for resource constrained devices in M2M and smart object networks. It is currently being developed by the Constrained RESTful Environments (CoRE) working group at the Internet Engineering Task force (IETF). CoAP provides a request/response interaction model similar to HTTP while keeping in mind the specific requirements of constrained device networks, such as support for multicasting, asynchronous messaging and low packet parsing overhead. CoAP uses a short fixed sized header and compact options along with a small payload. It

[4]Relic-Configurations, https://github.com/ms88/relic-configurations

runs on top of the unreliable User Datagram Transport (UDP) protocol with optional reliability. Using a binary base header, it supports two message types, request and response. A request can be a confirmable or non-confirmable depending on whether an acknowledgement is required from the recipient. A CoAP request consists of the Request Method for the resource being requested, an identifier, a payload and an Internet media type (if any) with optional meta-data about the request. The basic Request Methods supported in CoAP are GET, POST, PUT and DELETE. These methods can easily be mapped to HTTP and have the same safe (retrieval only) and idempotent (multiple invocations have same result) properties as HTTP. A CoAP response is identified by the Response Code in the CoAP header and is similar to the Status Code field of the HTTP header. It indicates the result of an attempt to execute the received request.

- Link Format: M2M and smart object networks are envisoined to work without human interaction. In such a scenario, discovery of resources hosted by a constrained server/smart object is important. Similar to Web Discovery and Web Linking [30] defined for HTTP, the CoRE working group is developing an equivalent CoRE discovery and CoRE link format [31] for constrained devices to support resource discovery and web linking. The resource discovery mechanism provides a set of Universal Resource Identifiers (URIs) or links that represent the resources hosted on the constrained server along with any additional attributes and link relations between the resources. The link format is carried as payload data and is assigned its own internet media type "application/link-format". A well known URI "/.well-known/core" is defined as the default entry point for requesting a list of resources hosted by the constrained server.
- Resource Directory: In many M2M networks, smart objects are often dispersed and have intermittent reachability either because of network outages or because they sleep during their operation cycle to save energy. In such scenarios, direct discovery of resources hosted on the constrained server might not be possible. To overcome this barrier, the idea of a Resource Directory (RD) [32] is being developed. A Resource Directory is an entity that hosts the descriptions of resources which are located on other dispersed constrained nodes. End Points (EPs) or constrained servers proactively discover, register and maintain their resources with the RD. The RD provides interfaces for registering, updating and removing resources to the EPs. It is also possible for the RD to proactively discover resources from the EPs and add or validate these entries.
- Mirror Proxy: A Mirror Proxy (MP) [33] is an entity that allows sleeping nodes to participate in RESTful architecture despite the fact that they do not behave as traditional web servers and sleep for long periods during their operation cycle. The MP is responsible for caching and serving data on behalf of sleeping constrained devices also referred to as Sleeping End Points (SEPs). The Mirror Proxy is designed such that it would host the resources of other SEP in its own resource tree on their behalf. The MP is supposed to update its /.well-know/core resource to add the resources of the registered SEP. The MP also registers the resources of the SEP as separate resources in the RD.
- JSON Web Singatures and Web Key: JavaScript Object Notation (JSON) Web Signatures (JWS) [34] is a representational format that uses JSON data structures for depicting content that has been secured with Hash based Message Authentication codes (HMACs) or digital signature schemes such as ECDSA. The representation format is independant of the content and can be used with any arbitrary data. A JWS representation consists of three parts: the JWS header, the JWS payload, and the JWS signature. The JWS header describes the HMAC or signature algorithm used. The payload consists of the content that needs to be secured and the signature is obtained by applying the cryptographic or HMAC algorithm over the header and payload. Similar to JWS, JSON Web Key (JWK) [35] is used to represent and communicate public keys as JSON objects. The JWK representation of a public key consists of JSON-object members that describe the public key such as the asymmetric cryptographic algorithm used.

IV. Security Architecture

While designing the architecture, one of the primary goals at the outset was to ensure that smart objects can sleep for long durations during the operational phase to save energy. However we did not want this requirement to be a hindrance for a client that wishes to obtain the most recent data update sent from the smart object. Thus our intention was to satisfy two contradicting goals where smart objects can serve updates to requesting clients at any time despite the fact that they sleep for long durations during their operational phase. In order to achieve these contradicting goals, we use Mirror Proxies [33] in our architecture to delegate the task of serving data from smart objects to proxies as shown in Figure 2.

A Mirror Proxy is assumed to have sufficient computational power and energy supply to remain online and serve data collected from several SEPs or sleeping smart objects. The SEPs no longer operate as servers that serve to client requests for data and rather act as clients of the MP themselves. The SEPs register their resources with the MP. When the MP receives a registration request, it adds the resources of the SEPs into its own resource tree as sub-resources and updates its /.well-known/core resource to reflect the additional resources. Once the registration is successfully acknowledged by the MP, the SEPs can sleep and wake-up at pre-determined intervals to update the cached content that is continuously served by the MP. The communication network between the SEPs and the MP runs on CoAP over UDP.

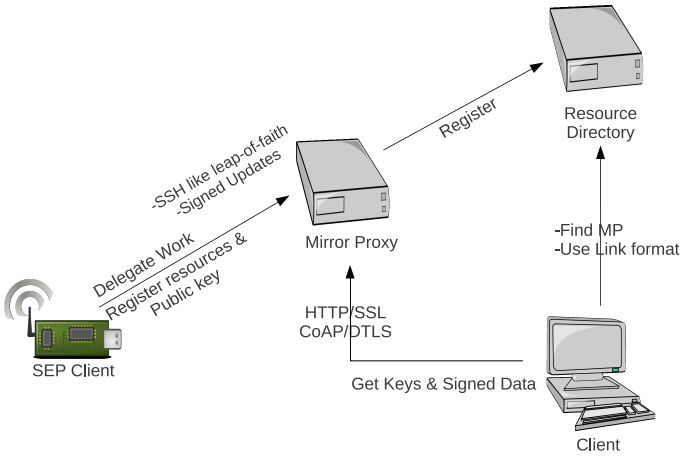

Fig. 2. Architecture

While SEPs can update the cached content with either Confirmable or Non-Confirmable CoAP messages, it may be desirable to use Non-Confirmable messages in certain scenarios to allow the SEPs to sleep without waiting for acknowledgements. This might be desirable when, for example, the transmission medium is relatively reliable or when the MP is interested in the mean or average value from a number of reporting SEPs and an occasional loss of updates is tolerable.

In order ensure security, our architecture requires each constrained device to have a public-private key pair. The architecture is independent of the public-key algorithm chosen. However, our results in Section II-C and results from previous work [7] suggest that ECC is not only more efficient than RSA but also results in smaller signatures, thereby reducing the network traffic. The key pair can be pre-generated and configured into the constrained device at the time of manufacture or can be created on the fly during the operational phase.

As seen in Figure 2, when a smart object sends a registration request to the MP, it also adds its public key to the registration message. The MP is responsible for storing the public key of each of the SEP that it serves. The SEPs wake-up at predetermined intervals and use the corresponding private key to sign all subsequent updates sent to the MP. The public keys and the signed content are sent in the standard JSON-based JWK [35] and JWS [34] formats respectively. This system essentially provides a SSH-like leap of faith where, after an uncompromised initial connection, the data integrity and authenticity is ensured. Such a system does not require any pre-configuration and allows the integrity and authenticity of updates to be verified by any node in the network at any point of time even when the SEP is asleep. This allows incremental deployment, where new SEPs can be added to the network at any point of time and also allows SEPs and MPs from different manufacturers to inter-operate. It also enables the MP to maintain several administrative domains with different access policies depending on the SEP or the client it is serving.

In some deployments, the network between the SEPs and the MP can be assumed to be secure and sending messages in plaintext along with signature is acceptable. However if required, data confidentiality can be assured if the MP also owns a public-private key pair and performs a Diffie-Hellman exchange with the SEPs at the time of registration to establish a shared secret. This shared secret would then be used to encrypt and decrypt the signed updates. A Mirror Proxy also needs to verify the updates sent from SEPs to prevent itself from caching malicious data. Alternatively, it may also choose to cache the data only for a limited maximum amount of time or store only a maximum number of signed data updates to prevent a compromised SEP from overwhelming it.

When a smart object registers with the MP, the MP registers the new resources separately in the Resource Directory (RD). In order to obtain data updates for a resource, a client contacts the RD to obtain the location of the SEP hosting this resource. On receiving a request from a client, the RD responds with the location of the resource requested. Although the location returned by the RD points to the MP, the client is unaware of this fact and believes it to be the location of the SEP itself. Thus, the MP serves data from the SEPs to the clients in a transparent manner. A client can save the location returned and bypass requesting the RD when it wants to obtain the next data update for the resource.

A client first obtains the public key of the SEP (which resides on the MP) and then retrieves the signed data updates (transparently cached by the MP). It can now correctly verify the authenticity and the integrity of data objects signed by the SEP. This communication between a client and the MP, as shown in Figure 2, occurs over HTTP or CoAP and should be protected with SSL [17] or DTLS [36] respectively. Therefore, this architecture securely communicates data-object integrity and authenticity end to end from the SEP to the client over a multi-hop network topology. Using SSL or DTLS ensures that signed updates are protected from a malicious eavesdroppers and man-in-the-middle modifications. Since the SEPs sleep for long durations and are never directly contacted with client requests for data, they are also inherently protected against denial-of-sleep attacks [37].

It is also important to ensure message freshness to protect the architecture against replay attacks. Including sequence numbers in signed messages can provide an effective method of replay protection. The MP should verify the sequence number of each incoming message and accept it only if the sequence number is greater than the highest previously seen sequence number. The MP drops any packet with a sequence number that has already been received or if the received sequence number is greater than the highest previously seen sequence number by an amount larger than the preset threshold.

Sequence numbers can wrap-around at their maximum value and, therefore, it is essential to ensure that sequence numbers are sufficiently long. However, including long sequence numbers in packets can increase the network traffic originating from the SEP and can thus decrease its energy efficiency. To overcome the problem of long sequence numbers, we can use a scheme similar to that of Huang [38], where the sender and

receiver maintain and sign long sequence numbers of equal bit-lengths but they transmit only the least significant bits.

Although sequence numbers protect the system from replay attacks, the MP has no mechanism to determine the time at which updates were created by the SEP. Moreover, if sequence numbers are the only freshness indicator used, a malicious eavesdropper can induce inordinate delays to the communication of signed updates by buffering messages. Depending on the hardware used by the SEPs, they may have access to accurate hardware clocks which can be used to include timestamps in the signed updates. These timestamps are included in addition to sequence numbers. The clock time in the SEPs can be set by the manufacturer or the current time can be communicated by the MP during the registration phase. However, these approaches require the SEP to either rely on the long-term accuracy of the clock set by the manufacturer or to trust the MP, thereby increasing the potential vulnerability of the system. The SEPs could obtain the current time from NTP, but this may consume additional energy and give rise to security issues discussed by Mills [39]. The SEPs could also have access to a GSM network or the Global Positioning System (GPS), and they can be used obtain the current time.

V. Implementation

To verify the feasibility of our architecture we developed a proof-of-concept prototype. In our prototype, the SEP was implemented using the Arduino Ethernet shield over an Arduino Mega board as shown in Figure 3.

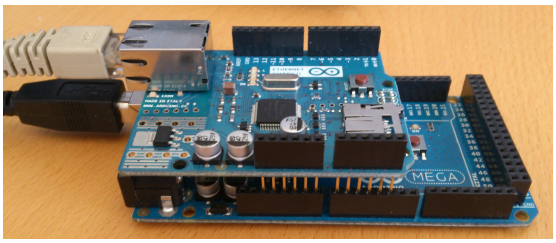

Fig. 3. Arduino SEP

Our implementation uses the standard C99 programming language on the Arduino Mega board. In this prototype, the Mirror Proxy (MP) and the Resource Directory (RD) reside on the same physical host. A 64-bit x86 linux machine serves as the MP and the RD, while a similar but physically different 64-bit x86 linux machine serves as the client that requests data from the SEP. We chose the Relic library version 0.3.1 for our sample prototype as it can be easily compiled for different bit-length processors. Therefore, we were able to use it on the 8-bit processor of the Arduino Mega, as well as on the 64-bit processor of the x86 client. We used ECDSA to sign and verify data updates with the standard NIST-K163 curve parameters (163-bit Kobltiz curve over binary field). While compiling Relic for our prototype, we used the fast configuration without any assembly optimizations.

We used the Ericsson Gateway [40] running on a x86 linux machine to serve as the MP and the RD. The gateway implements the CoAP base specification in the Java programming language and extends it to add support for Mirror Proxy and Resource Directory REST interfaces. We also developed a minimalistic CoAP C-library for the Arduino SEP and for the client requesting data updates for a resource. The library has small SRAM requirements and uses stack-based allocation only. It is inter-operable with the Java implementation of CoAP running on the gateway. The ECDSA signature generation and signature verification functionality were used with this library. The location of the MP was pre-configured into the SEP by hardcoding the IP address. We used an IPv4 network with public IP addresses obtained from a DHCP server.

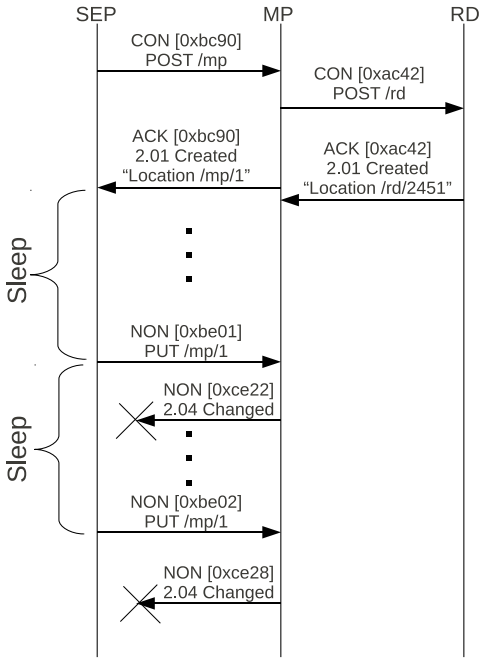

Fig. 4. Registering and Caching Updates

In our prototype, the SEP registers with the MP by sending a Confirmable CoAP POST message as shown in figure 4. This registration message includes a temperature resource in the CoRE link format along with the public key of the SEP in the JWK format. The MP adds this resource as a sub-resource in its own resource tree and also updates its ./well-known/core resource. The MP then sends a piggybacked CoAP ACK to the SEP to confirm the successful registration. The piggybacked CoAP ACK contains the location that would be used by the SEP to update the cache. The MP also stores the public key of the SEP received in the registration message.

Once the ACK from the MP confirming the registration is received by the SEP, it can go into the energy saving sleep mode. However, if the registration message or the ACK is lost, the SEP would re-transmit the registration message. The MP is responsible for registering the new resources with the RD and it sends a confirmable CoAP POST message to add these new resources as shown in Figure 4. In our prototype, we use hardcoded temperature values in the SenML [41] format as sample data along with simple sequence numbers to ensure

freshness. This SenML data is signed with ECDSA algorithm and sent to the MP in the JWS format with Non-Confirmable CoAP PUT messages. This allows the SEPs to return to the sleep mode without having to wait for any acknowledgements. Although the Non-Confimable CoAP PUTs do not result in an ACK from the MP, the MP still responds to the requested operation indicating a success or failure as shown in Figure 4. The SEP in this case remains in the sleep mode and is unaware of any packets sent to it. There are no CoAP options currently available for suppressing such responses.

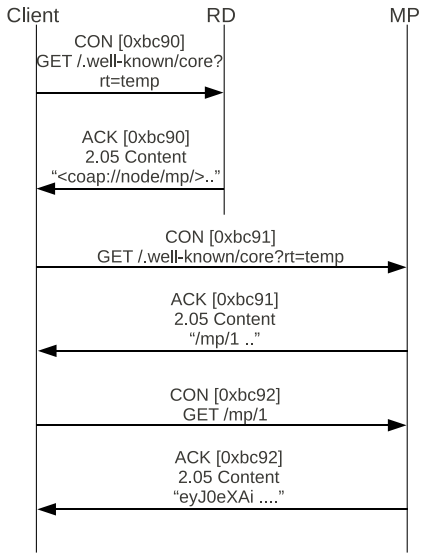

Fig. 5. Retrieving Data Updates

A client (the x86 linux machine) that wishes to obtain data updates from the SEP first contacts the RD as shown in Figure 5. We assume that the location of the RD is known to the client through DHCP or through pre-configuration. The client uses a Confirmable CoAP GET message for the /.well-known/core resource and specifies the resource type parameter to determine the location of the temperature resource. The RD responds with a CoAP ACK containing location of the resource piggybacked in the message. Although this location points to the MP, the client is not aware of this and believes it to be the location of the SEP itself (transparent caching).

The client then sends a second CoAP GET message to the location returned by the RD for the ./well-known/core resource. The MP returns a piggybacked CoAP ACK containing the location where the data is being cached along with the public key of the SEP from which the updates were received. The client stores the public key for this SEP and sends a third CoAP GET message as shown in Figure 5 to obtain the actual signed content in the JWS format. The client can use the public key received to verify all subsequent signed data updates. If the signature verifies correctly, the client can be assured of the integrity and authenticity of these data updates. We use an unprotected CoAP communication channel between the client and the MP. However if required, this communication could be secured with DTLS or by using HTTP over SSL.

Some important statistics of this prototype are listed in table II. Our straw man analysis of the performance of this prototype is preliminary. Our intention was to demonstrate the feasibility of the entire architecture with public-key cryptography on an 8-bit microcontroller. The stated values can be improved further by a considerable amount. For example, the flash memory and SRAM consumption is relatively high because some of the Arduino libraries were used out-of-the-box and there are several functions which can be removed. Similarly we used the fast version of the Relic library in the prototype instead of the low memory version.

TABLE II
PROTOTYPE PERFORMANCE

Flash memory consumption (for the entire prototype including Relic cryto + CoAP + Arduino UDP, Ethernet and DHCP Libraries)	51 kB
SRAM consumption (for the entire prototype including DHCP client + key generation + signing the hash of message + COAP + UDP + Ethernet)	4678 bytes
Execution time for creating the key pair + sending registration message + time spent waiting for acknowledgement	2030 ms
Execution time for signing the hash of message + sending update	987 ms
Signature overhead	42 bytes

To demonstrate the efficacy of this communication model we compare it with a scenario where the smart objects do not transition into the energy saving sleep mode and directly serve temperature data to clients. As an example, we assume that in our architecture, the smart objects wake up once every minute to report the signed temperature data to the caching MP. If we calculate the energy consumption using the formula $W = U*I*t$ (where U is the operating voltage, I is the current drawn and t is the execution time), and use the voltage and current values from the datasheets of the ATmega2560 (20mA-active mode and 5.4mA-sleep mode) and W5100 (183mA) chips used in the architecture, then in a one minute period, the Arduino SEP would consume 60.9 Joules of energy if it directly serves data and does not sleep. On the other hand, in our architecture it would only consume 2.6 Joules if it wakes up once a minute to update the MP with signed data. Therefore, a typical Li-ion battery that provides about 1800 milliamps per hour (mAh) at 5V would have a lifetime of 9 hours in the unsecured always-on scenario, whereas it would have a lifetime of about 8.5 days in the secured sleepy architecture presented. These lifetimes appear to be low because the Arduino SEP in the prototype uses Ethernet which is not energy efficient. The values presented only provide an estimate (ignoring the energy required to transition in and out of the sleep mode) and would vary depending on the hardware and MAC protocol used. Nonetheless, it is evident that our architecture can increase the life of smart objects by allowing them to sleep and can ensure security at the same time.

VI. CONCLUSION

In this work we demonstrate how public-key cryptography can be implemented in software on small resource constrained

devices with publicly available cryptographic libraries. Although performing a key exchange followed by the use of symmetric keys would be desirable in some cases, such a model works well only for DTLS and other transport layer solutions and works less well for data object security, particularly when the number of communicating entities is not exactly two.

Using these publicly available libraries, we develop a functional standards-compliant prototype of secure communication and mirroring mechanism with CoAP for sleepy smart objects. Our architecture and communication model not only allows intermediaries to act as caches without impacting the security, it also operates in the presence of traditional middleboxes such as protocol translators or NATs although we do not recommend their use in these environments. Our architecture is agnostic to the application layer communication protocol used, and can work seamlessly over HTTP instead of CoAP if desired. We use a leap-of-faith provisioning system as a starting point and it requires further investigation on how things can be paired securely. We intend to research how multicasting can be used in such networks, how to implement group communication and what kind of congestion control, if any, would be needed in such networks.

ACKNOWLEDGMENT

We would like to thank Tuomas Aura, Heidi-Maria Rissanen, Mats Näslund and Daoyuan Li for their continued support and help. Special gratitude to Diego F. Aranha for the immense help he provided with configuring and using the Relic library.

REFERENCES

[1] "MIT AUTO-ID labs," Accessed 02.04.2012. [Online]. Available: http://autoid.mit.edu/cs/

[2] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *Communications of the ACM*, vol. 47, no. 6, pp. 53–57, 2004.

[3] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler, "SPINS: Security protocols for sensor networks," *Wireless networks*, vol. 8, no. 5, pp. 521–534, 2002.

[4] C. Karlof, N. Sastry, and D. Wagner, "TinySec: a link layer security architecture for wireless sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 162–175.

[5] L. Eschenauer and V. Gligor, "A key-management scheme for distributed sensor networks," in *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 2002, pp. 41–47.

[6] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *Symposium on Security and Privacy Proceedings*. IEEE, 2003, pp. 197–213.

[7] N. Gura, A. Patel, A. Wander, H. Eberle, and S. Shantz, "Comparing elliptic curve cryptography and RSA on 8-bit cpus," *Cryptographic Hardware and Embedded Systems-CHES 2004*, pp. 925–943, 2004.

[8] E. Blaß and M. Zitterbart, "Towards acceptable public-key encryption in sensor networks," in *ACM 2nd International Workshop on Ubiquitous Computing*, 2005, pp. 88–93.

[9] J. Bos, M. Kaihara, T. Kleinjung, A. Lenstra, and P. Montgomery, "Solving a 112–bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction," *International Journal of Applied Cryptography*, vol. 2, no. 3, pp. 212–228, 2012.

[10] L. Uhsadel, A. Poschmann, and C. Paar, "Enabling full-size public-key algorithms on 8-bit sensor nodes," *Security and Privacy in Ad-hoc and Sensor Networks*, pp. 73–86, 2007.

[11] R. Hassan and T. Qamar, "Asymmetric-key cryptography for contiki," Master's thesis, Chalmers University of Technology, 2010.

[12] T. Denis, "Libtomcrypt," 2004, Accessed 04.04.2012. [Online]. Available: http://libtom.org

[13] D. F. Aranha and C. P. L. Gouvêa, "RELIC is an Efficient LIbrary for Cryptography," http://code.google.com/p/relic-toolkit/.

[14] M. Baar, E. Köppe, A. Liers, and J. Schiller, "Poster abstract: The scatterweb msb-430 platform for wireless sensor networks," in *Contiki Workshop*, 2007.

[15] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. Marrón, "COOJA/MSPSim: interoperability testing for wireless sensor networks," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. ICST, 2009, p. 27.

[16] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, and S. Chang Shantz, "Sizzle: A standards-based end-to-end security architecture for the embedded internet," *Pervasive and Mobile Computing*, vol. 1, no. 4, pp. 425–445, 2005.

[17] A. Freier, P. Karlton, and P. Kocher, "Rfc 6101: The secure sockets layer (SSL) protocol version 3.0," IETF, Tech. Rep., 2011.

[18] A. Wander, N. Gura, H. Eberle, V. Gupta, and S. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks," in *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*. IEEE, 2005, pp. 324–328.

[19] D. Carman, P. Kruus, and B. Matt, "Constraints and approaches for distributed sensor network security (final)," *DARPA Project report, (Cryptographic Technologies Group, NAI Labs)*, vol. 1, p. 1, 2000.

[20] E. Laan, "AVRCryptolib," Accessed 04.04.2012. [Online]. Available: http://www.emsign.nl/

[21] B. Kaliski, "RFC 2313: PKCS# 1: RSA encryption version 1.5," March 1998.

[22] A. Liu and P. Ning, "TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks," in *International Conference on Information Processing in Sensor Networks, IPSN*.

[23] T. Baumgartner, I. Chatzigiannakis, S. Fekete, C. Koninis, A. Kröller, and A. Pyrgelis, "Wiselib: A generic algorithm library for heterogeneous sensor networks," *Wireless Sensor Networks*, pp. 162–177, 2010.

[24] B. Titzer, D. Lee, and J. Palsberg, "Avrora: Scalable sensor network simulation with precise timing," in *Fourth International Symposium on Information Processing in Sensor Networks, IPSN*. IEEE, 2005, pp. 477–482.

[25] A. Menezes, P. Van Oorschot, and S. Vanstone, *Handbook of applied cryptography*. CRC, 1997.

[26] D. Hankerson, S. Vanstone, and A. Menezes, *Guide to elliptic curve cryptography*. Springer-Verlag New York Inc, 2004.

[27] A. Kargl, S. Pyka, and H. Seuschek, "Fast arithmetic on ATmega128 for elliptic curve cryptography," *preprint, available online at http://eprint.iacr.org/2008/442*, 2008.

[28] D. Aranha, R. Dahab, J. López, and L. Oliveira, "Efficient implementation of elliptic curve cryptography in wireless sensors," *Advances in Mathematics of Communications*, vol. 4, no. 2, pp. 169–187, 2010.

[29] Z. Shelby, K. Hartke, C. Bormann, and D. Sturek, "Protocol requirements for internet media guides," IETF, Internet Draft, March 2012.

[30] H. Schulzrinne, S. Casner, R. Frederic, and V. Jacobson, "RFC 5988: Web linking," October 2010, status: PROPOSED STANDARD.

[31] Z. Shelby, "CoRE link format," IETF, Internet Draft, January 2012.

[32] Z. Shelby and K. S., "CoRE resource directory," IETF, Internet Draft, November 2011.

[33] M. Vial, "CoRE mirror proxy," IETF, Internet Draft, November 2011.

[34] J. Bradley, N. Sakimura, and M. Jones, "JSON web signature (JWS)," IETF, Internet Draft, November 2011.

[35] M. Jones, "JSON web key (JWK)," IETF, Internet Draft, January 2012.

[36] E. Rescorla and N. Modadugu, "RFC 4347: Datagram transport layer security," 2006.

[37] F. Stajano and R. Anderson, "The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks," in *Security Protocols*. Springer, 2000, pp. 172–182.

[38] C. Huang, "Loft: Low-overhead freshness transmission in sensor networks," in *Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC'08. IEEE International Conference on*. IEEE, 2008, pp. 241–248.

[39] D. Mills, "RFC 1305: Network time protocol (version 3) specification, implementation and analysis," March 1992.

[40] J. Höller, "Internet of things comes alive through smart objects interoperability," April 2012, https://labs.ericsson.com/developer-community/blog/internet-things-comes-alive-smart-objects-interoperability.

[41] C. Jennings, J. Arkko, and Z. Shelby, "Media types for sensor markup language (SENML)," IETF, Internet Draft, January 2012.