Aalto University
School of Science
Degree Programme of Computer Science and Engineering

Mohit Sethi

Security in Smart Object Networks

Master's Thesis
Espoo, June 30, 2012

Supervisors: Professor Tuomas Aura, Aalto University, Finland
Professor Markus Hidell, Royal Institute of Technology, Sweden

Instructors: Ari Keränen, NomadicLab, Ericsson Research, Finland
Jari Arkko, NomadicLab, Ericsson Research, Finland

Aalto University
School of Science
Degree Programme of Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author:	Mohit Sethi		
Title:	Security in Smart Object Networks		
Date:	June 30, 2012	**Pages:**	73
Professorship:	Computer Science	**Code:**	T-110
Supervisors:	Professor Tuomas Aura Professor Markus Hidell		
Instructors:	Ari Keränen, M.Sc. (Tech.) Jari Arkko, Licentiate (Tech.)		

Internet of Things (IoT) refers to an inter-connected world where physical devices are seamlessly integrated into the Internet and become active participants of business, information and social processes. This involves the inter-connection of a large number of heterogeneous networked entities and networks. 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. The communication infrastructure for these objects is based on an extension of the Internet protocol stack.

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 thesis, 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. We contribute to the standardization of the generic communication architecture by adding security and delegation components for smart objects that sleep for large amounts of time during their operational phase. This architecture ensures data integrity and authenticity over a multi-hop network topology. It also 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 thesis 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 micro-controllers without modifying the underlying cryptographic algorithms.

Keywords:	IoT, smart objects, security, CoAP, asymmetric cryptography, integrity, authenticity, mirroring mechanism
Language:	English

Acknowledgements

I sincerely thank Professor Tuomas Aura at Aalto University for his constant feedback and for providing the funding to attend the IETF 83 meeting, where I presented the initial results from the thesis. I am also grateful to Professor Markus Hidell for supervising the thesis at Royal Institute of Technology.

I owe my gratitute to my instructors Ari Keränen and Jari Arkko at NomadicLab, Erisson Research for their regular guidance and advice during the course of my research work. I am indebted to my colleagues at NomadicLab for their continuous support.

Finally, I would like to thank my family and friends for their moral support and motivation.

Espoo, June 30, 2012

Mohit Sethi

Abbreviations and Acronyms

CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environments
DHCP	Dynamic Host Configuration Protocol
DLP	Discrete Logarithmic Problem
DNS	Domain Name System
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
EAP	Extensible Authentication Protocol
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithmic Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
EXI	Efficient XML Interchange
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HIP	Host Identity Protocol
HIP-BEX	HIP Base EXchange
HIP-DEX	HIP Diet EXchange
HTTP	Hyper-Text Transfer Protocol
IETF	Internet Engineering Task Force
IKEv2	Internet Key Exchange Protocol version 2
IoT	Internet of Things
IPSec	Internet Protocol Security
JSON	JavaScript Object Notation
JOSE	JavaScript Object Signing and Encryption
JWK	JSON Web Key
JWS	JSON Web Signature
LWIG	Light Weight Implementation Guidance
MAC	Media Access Control
MP	Mirror Proxy
M2M	Machine to Machine

NAT	Network Address Translation
nesC	Network Embedded Systems C
NIST	National Institute of Standards and Technology
NTP	Network Time Protocol
PAA	PANA Authentication Agent
PANA	Protocol for Carrying Authentication for Network Access
PGP	Pretty Good Privacy
RD	Resource Directory
REST	Representational State Transfer
RFID	Radio-Frequency Identification
RSA	Rivest Shamir Adelman Cryptographic Algorithm
RTT	Round Trip Time
SA	Security Association
SAAG	Security Area Advisory Group
SECG	Standards for Efficient Cryptography Group
SenML	Sensor Markup Language
SEP	Sleeping End-point
SIM	Subscriber Identity Module
SRAM	Static Random Access Memory
SSH	Secure Shell
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Universal Resource Identifier
URN	Universal Resource Name
UTF-8	Universal Character Set Transformation Format 8-bit
WLAN	Wireless Local Area Network
XML	Extensible Markup Language
6LoWPAN	IPv6 based Low-Power Personal Area Networks

Contents

List of Tables

List of Figures

Chapter 1

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 a radio-frequency identification (RFID) tag having a globally unique identifier. This would not only allow tracking of objects in real-time but also allow quering 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 emergence of technologies such as Bluetooth low energy [5], Zigbee [6] and embedded sensor technology, the physical objects no longer act as unresponsive nodes and have transformed into objects that 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. As the computational power of such smart objects increases and their physical size decreases, they are becoming more productive at cheaper costs. Thus, these smart objects are on the path to form a pervasive network around us. The importance of IoT in our future daily lives is further asserted from the fact that IoT is included by the US National Intelligence Council in the list of six "Disruptive Civil Technologies" with potential impacts on US national power [7]. It is not surprising then that the area is attracting the attention of researchers all over to solve issues that might hinder the seamless adoption of IoT in our everyday life.

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 are more perplexing than those in the current Internet. Moreover, these devices are extremely constrained in terms of computational power and memory, which makes it even more arduous to ensure strong security in these networks.

To better understand the acute nature of the security issues, consider the use-case where the lights in a house are automatically controlled by a sensor that detects the amount of natural light available. An attack in such a scenario might not just be a prank by a neighbor controlling the lights of the house but a large-scale coordinated attack that can potentially turn-off the lights of an entire city. Although the needs for securing the IoT is generally well understood and accepted, there is a lack of consensus on which Internet security protocols will be used in the context of IoT and how.

1.1 Problem Area

As illustrated by the previous example, smart objects need to be protected against a wide variety of attacks during their lifecycle. For example, during the provisioning of a smart object, 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.

Smart objects should be available at relatively low prices to support their large-scale deployment. Thus, including additional hardware features to support secure provisioning or communication may not be possible. Additionally, the smart objects are often deployed in small spaces and inaccessible areas, which limits the possibility of additional hardware or regular access for installing software updates and fixing security vulnerabilities.

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.

1.2 Research Goals and Methodology

Keeping in mind the above-stated problems, we aim to design a solution that not only ensures end-to-end data integrity and authenticity but also allows resource-constrained low-power sensors to delegate to a gateway or

proxy the task of serving data to their clients. 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. The existing Internet security protocols need to be applied in an adept manner on these small platforms. Thus, we define the following as the goals of this thesis:

- Designing a security architecture for smart object networks, which includes an energy-efficient communication model. The architecture will be based on the Constrained Application Protocol (CoAP) [132].
- Developing a prototype of the new architecture.
- Implementing and evaluating the performance of asymmetric public key cryptography on 8-bit architectures.
- Contributing to the current standardization of CoAP and JavaScript Object Notation (JSON) [39] signature representation and transfer.

The architecture presented in this thesis is entirely based on existing standards for smart object networks. We discuss several secure provisioning and message-freshness schemes that may used with this architecture. A performance analysis of the cryptographic algorithms used in this architecture is done on constrained devices. The purpose of our prototype implementation is to support the standardization of the new secure and energy-efficient communication model.

1.3 Structure of the thesis

The rest of the thesis is organized as follows. Chapter 2 starts by describing the lifecycle of a smart object along with the security threats that it faces at each stage of the lifecycle. It then goes on to discuss the CoAP protocol, resource discovery mechanisms in smart object networks and the fundamentals of public-key cryptography. It concludes with a brief background of standard signature and public-key representation formats. Chapter 3 begins by detailing previous research that has been done to implement public-key cryptography on constrained platforms. It then documents the libraries that are publicly available for use and finally evaluates the performance of these libraries on a 8-bit micro-controller. The entire proposed architecture is elucidated in Chapter 4 and the implemented prototype along with its functioning is explained in Chapter 5. Chapter 6 discusses some analytical perspectives and evaluations on the developed architecture. Finally, Chapter 7 provides a summary of the thesis and examines the potential future work items.

Chapter 2

Background

This chapter begins by describing the lifecycle of a smart object and the vulnerabilities that it may encounter during each phase of the lifecycle, followed by an introduction to the Constrained Application Protocol (CoAP). Thereafter, the current standardization work for resource representation and discovery in smart object networks is presented. Since our communication architecture utilizes public-key cryptography, a summary of common algorithms for asymmetric public-key cryptography is provided. Finally, the chapter ends with a description of the existing standards for communicating public keys and signed content.

2.1 Lifecycle of a Smart Object

The lifecycle of a smart object, depicted in Figure 2.1, begins when the object is manufactured. Smart objects are tailored to perform different tasks such as temperature measurement, pressure sensing, lighting automation and a variety of other operations depending on their application area. Thus it is unlikely that a single manufacturer would be responsible for producing all the objects that may be used together in a particular use-case. It is therefore required that objects from different manufacturers not only inter-operate, but also securely bootstrap with each other.

The object, after its manufacture, is installed and commissioned within a network by an installer. Depending the deployment scenario, the installer may be the manufacturer, the end user or some other third party. During this bootstrapping phase, the object identity and secret keys that would be used during the operational phase are provided to the object. This bootstrapping process may not be a discrete event and may extend over a period of time involving a number of parties.

Once the object has been bootstrapped into the network, it enters the operational phase and is under the control of the owner of the object. Depending on the lifetime of an object, there might be a maintenance phase where the software or firmware may be upgraded either on site or remotely. It may be required to re-bootstrap the smart object after maintenance if the required state information is lost once the maintenance is complete. The object continues to loop through this phase of operation and maintenance until it is finally decommissioned and removed from the network. This marks the end of the lifecycle of the smart object. However, this does not necessarily mean that the object has become defective or unusable. It is therefore possible that the removed device is re-commissioned in a different network under a different owner starting the lifecycle again.

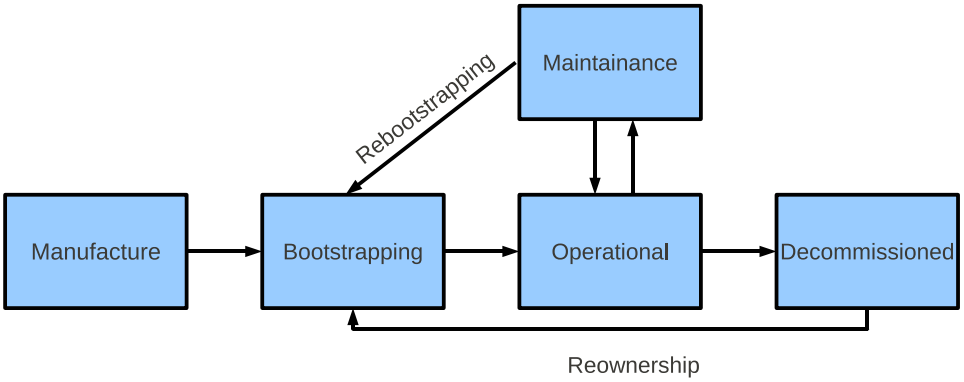


Figure 2.1: Lifecycle and Vulnerabilities for Smart Objects

During each phase of its lifecycle, a smart object is susceptible to a variety of security vulnerabilities. These security vulnerabilities include [63]:

1. Cloning during manufacture: At the time of manufacture, an untrusted manufacturer may clone the security keys of a smart object. Thereafter, the malicious manufacturer may sell the cloned security keys to a third party. Depending on its intentions, a manufacturer may also alter the software or firmware to implement a back-door. However, in this thesis, it is assumed that a manufacturer can be trusted and any security keys would not be duplicated for misuse or sale to third parties.

2. Eavesdropping: An adversary may be able eavesdrop during the provisioning of a smart object and obtain keying materials, security parameters, or initial settings if they are exchanged in clear using a wireless medium. The adversary could then use this information to recover the secret keys established, thereby compromising the authenticity and confidentiality of the channel. Additionally, an eavesdropper

may gain meaningful information from traffic analysis during the operational phase of a smart object. It is also possible for a malicious eavesdropper to cause a replay attack by recording and replaying packets if appropriate message freshness mechanisms are not used.

3. Man-in-the-middle attack: The provisioning stage is also susceptible to man-in-the-middle attacks. For example, if the keying material between communicating entities is exchanged in the clear and the security of the keying protocol depends on the assumption that no one is able to eavesdrop and actively modify the messages between the two communicating entities during the execution of this protocol (leap-of-faith based systems). 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 nodes from innocent neighbors in the network via completely automated mechanisms. Such an attack is also possible during the operational phase when a re-keying is performed.

4. Firmware Replacement: When a smart object is in the maintenance phase, the manufacturer may update the software or firmware to provide new functionality and features. An attacker may be able to exploit such an upgrade by updating the objects with malicious code, thereby influencing its operational behavior.

5. Extraction of security parameters: A smart object deployed in the ambient environment is typically physically unprotected and could be captured by an adversary. Such an adversary may then attempt to extract security information such as cryptographic keys which may be printed on the device or try and re-program the device to serve its needs.

6. Routing attack: As shown by Park et al. [118], routing information in smart object networks can be spoofed or replayed in order to create routing loops, extend or shorten paths and hinder the usual routing behavior of the network. Several other routing attacks that can occur in such networks are:

    - Sinkhole/Blackhole [116]: An attacker pretends to have a high-quality route to a destination allowing it to lure nearly all the traffic from a particular region in the network. The attacker can then perform any desired processing on the packets passing through it.

- Selective forwarding [85]: A compromised node may selectively forward or drop packets.
- Wormhole attack [74]: An attacker significantly impacts routing and network statistics by recording packets at one location and tunneling them to another location.
- Sybil attack [44]: A malicious entity or node presents multiple identities to other objects in the network, thereby subverting a reputation system.

However, our communication model uses a simple IP [124] network and no special emphasis is given to attacks caused by different routing protocols.

7. Privacy threat: An adversary may track the location or usage behavior of a smart object and subsequently sell this information to interested parties for marketing, targeted advertising or spying.

8. Denial-of-Service: Smart objects typically have a small amount of memory and limited computational power available to them, thus, making them vulnerable to resource exhaustion. A malicious entity can launch a DoS attack using several techniques such as jamming the network with flooding or by continuously sending valid service requests to smart objects in order to deplete their resources.

9. Implementation vulnerabilities: If the cryptographic implementation utilizes weak security parameters or incorrectly implements cryptographic algorithms, a malicious entity may be able to extract information from the messages or obtain a copy of the cryptographic keys, thus, making the entire system vulnerable.

Our security architecture focuses on data-object integrity to counter eavesdropping and man-in-middle attacks. We design the architecture based on public-key cryptography and carefully choose the security parameters to prevent brute force attacks. We also discuss some important implementation details necessary for ensuring security in smart object networks. Additionally, our architecture also protects constrained devices against denial-of-service attacks.

There are several different defense mechanisms that have been suggested for smart object networks. Kivinen [89] discusses an Internet Key Exchange Protocol (IKEv2) [86] based approach for mutual authentication in M2M networks. IKEv2 is a component of Internet Protocol Security (IPSec) [87] that

is used for performing mutual authentication and maintaining Security Associations (SAs) between two nodes. Kivinen argues that IKEv2 includes many optional features which are not required in a minimal implementation for use in constrained device networks. He proposes the removal of features such as Network Address Translation (NAT), support for multiple SAs, Cookies and several others. However, he fails to discuss numerous policy aspects which are necessary to implement an inter-operable IPsec in any environment.

Host Identity Protocol (HIP) [112] is an inter-networking architecture that allows end-hosts to authenticate each other and protect their data flows with public keys using the HIP Base EXchange (HIP-BEX) [112] procedure. Urien et al. [139] used a modified version of HIP-BEX to implement privacy preserving RFID tags for the Internet of Things (IoT). They propose the use of private identification protocols such as Randomized Hash Lookup [145], to preserve the identity of a tag communicating with a reader.

HIP Diet Exchange (HIP-DEX) [111], another variant of HIP-BEX, is designed for use in smart object networks. It utilizes very few cryptographic primitives along with static elliptic curve Diffie-Hellman key pairs. HIP-DEX forgoes perfect forward secrecy and use of digital signatures to obtain a minimalistic implementation suitable for constrained devices. Kuptsov et al. [92] used HIP-DEX to develop a standards compliant security protocol for medical sensor networks which provides authentication, data protection and an access control mechanism.

Jara et al. [78] present a security architecture for medical sensor networks. This architecture is based on IPv6 based Low-Power Personal Area Networks (6LoWPAN) [114] along with a new protocol for mobility support. They use Subscriber Identity Module (SIM) [31] cards for authentication and encryption of data in their architecture.

TinySec [84] develops a link-layer security architecture for wireless sensor networks. It argues that unlike the one-to-one traffic pattern observed on the Internet, wireless sensor networks predominantly use a many-to-one communication model where multiple sensors communicate their readings over a multi-hop network topology to a central base-station. In order to reduce the amount of traffic and conserve the energy consumed, these networks use in-network processing techniques such as aggregation and duplicate elimination [98, 99], and therefore end-to-end security mechanisms for authenticity, integrity and confidentiality cannot be used. The authors therefore use a link-layer security architecture which can detect unauthorized packets at the point of injection. The architecture supports authenticity and integrity with optional confidentiality of link-layer messages. It uses block chaining based encryption techniques and discusses several keying mechanisms.

Extensible Authentication Protocol (EAP) [9] is an authentication frame-

work that supports multiple authentication methods. EAP runs directly over the link layer and supports duplicate detection with retransmission but does not allow fragmentation of packets. Protocol for Carrying Authentication for Network Access (PANA) [59] is a network-layer protocol with which a node can authenticate itself to gain access to the network. PANA does not define a new authentication protocol and rather uses EAP over User Datagram Protocol (UDP) [123] for authentication. Colin [38] proposes the use of PANA for secure bootstrapping of resource constrained devices. He demonstrates how a 6LowPAN Border Router (PANA Authentication Agent (PAA)) can authenticate the identity of a joining constrained device (PANA Client). Once the constrained device has been successfully authenticated, the border router can also provide network and security parameters to the joining device.

Bergmann et al. [20] implement a Datagram Transport Layer Security (DTLS) [126] based communication model for smart object networks. This work uses tinyDTLS [19], an open-source minimal DTLS library and replaces the existing cipher suite with the AES-CCM suite [16] to provide payload encryption with message authentication. The authors also implement a secure bootstrapping mechanism without pre-provisioned credentials using *resurrecting-duckling* imprinting scheme [136]. This bootstrapping protocol involves three distinct phases: *discover* (the duckling node searches for network nodes that can act as mother node), *imprint* (the mother node imprints a shared secret establishing a secure channel once a positive response is received for the imprinting request) and *configure* (additional configuration information such as network prefix and default gateway are configured). In this model for bootstrapping, a small initial vulnerability window is acceptable and can be mitigated using techniques such as a Faraday Cage to protect the environment of the mother and duck nodes, though this may be inconvenient for the user.

From the variety of defense mechanisms discussed thus far, it is clear that the problem of security in smart object networks still remains open and there is no consensus on which of the Internet security protocols (or a combination of them) would eventually form a predominant part of the deployment space. Another important observation that can be derived is the fact that while designing secure systems, considering all the vulnerabilities during the entire lifecycle of the smart object is critical.

2.2 CoAP

Constrained Application Protocol (CoAP) [132] is an application-layer communication protocol designed for resource constrained devices in M2M and

smart object networks. It is based on the Representational State Transfer (REST) [56] architecture and is currently being developed by the Constrained RESTful Environments (CoRE) working group at the Internet Engineering Task Force (IETF).

CoAP provides a generic request/response interaction model similar to the Hyper-Text Transfer Protocol (HTTP) [55] while giving due consideration to the specific requirements of constrained device networks, such as support for multicasting, asynchronous messaging and low packet parsing overhead. The messaging model is similar to the client/server model of HTTP. However, in typical M2M deployments, entities behave as both clients and servers and are therefore referred to as end points. Unlike HTTP, messages in CoAP are exchanged asynchronously over the unreliable datagram-oriented transport such as UDP [123] with optional reliability.

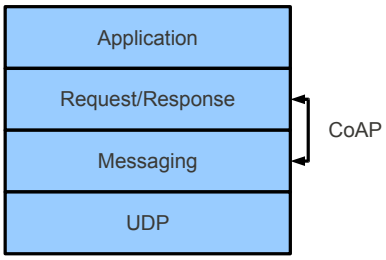


Figure 2.2: CoAP Abstraction

CoAP can be visualized with a two layer approach as depicted in Figure 2.2. The lower messaging layer is responsible for dealing with the asynchronous transport protocol interactions while the upper layer handles the request/response messaging using Method and Response codes.

CoAP uses a fixed-length binary header which may be followed by compact binary options in the Type-Length-Value (TLV) format and a payload (if any). The message format is shown in Figure 2.3 and the various header fields are defined as follows:

1. Version (Ver): 2-bit unsigned integer that specifies the CoAP version and must be set to 1 by applications. The other possible values are reserved for future use.

2. Type (T): 2-bit unsigned integer indicating the type of the message. The different types are: Confirmable, Non-Confirmable, Acknowledgement and Reset.

3. Option Count (OC): 4-bit unsigned integer stating the number of options following the header (can be from 0-14). When there are no

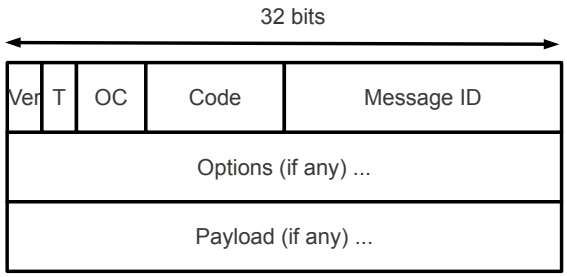


Figure 2.3: CoAP Message Format

options and the payload (if any) directly follows the header, it is set to 0. It is also possible to have an unlimited number of options by setting the option count to 15 and using the end-of-options marker to indicate the end of options.

4. Code: 8-bit unsigned integer which distinguishes a request message (1-31) from a response message (64-191). In case of a request message, the code field indicates the Request Method (such as GET/PUT/POST); and in case of a response message it indicates the Response Code (such as 2.01 Created/4.00 Bad Request).

5. Message ID: 16-bit unsigned integer used for detecting message duplicates or for matching Acknowledgement/Reset and Confirmable/Non-Confirmable messages.

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.

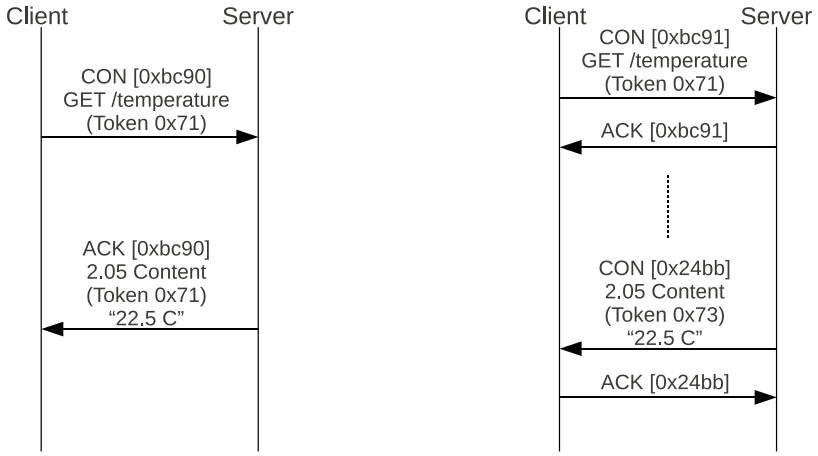
A CoAP response is identified by the Response Code in the 8-bit Code field of the CoAP header. It is similar to the Status Code field of the HTTP header and indicates the result of an attempt to execute the received request. The upper 3-bits of the Response Code identify the class (2 - Success, 4 - Client Error and 5 - Server Error) while the remaining bits identify the sub-category within the class.

The four different message types supported in CoAP are as follows:

1. Confirmable: Messages that require an acknowledgement. These messages can be a request or a response and cannot be empty. In case no packets are lost, they elicit only one return acknowledgement message.

2. Non-Confirmable: Messages that do not require an acknowledgement. These messages can be a request or a response and cannot be empty. Such messages are used when an application requires regular repeated transmissions and eventual delivery (or loss) is acceptable.

3. Acknowledgement: An acknowledgement message acknowledges the receipt of a confirmable message identified by its Message ID. It does not indicate the success or failure encountered in processing the encapsulated request. Depending on whether the request can be processed immediately or not, the response may be piggybacked in the acknowledgement or sent later separately as depicted in Figure 2.4.



(a) Response Piggbacked ACK (b) Empty ACK

Figure 2.4: Acknowledgements in CoAP

4. Reset: This message is sent in response to a confirmable or non-confirmable message indicating that it cannot be processed because of some missing context which might be caused when the node is rebooted and some state required to process the message is lost.

The CoAP base specification [132] provides a description of how DTLS can be used for securing CoAP. It proposes three different modes for using DTLS, namely: Presharedkey mode (where nodes have pre-provisioned keys for initiating a DTLS session with another node), RawPublicKey mode (where nodes have an asymmetric-key pair(s) but no certificates to verify the ownership) and Certificate mode (where public keys are signed in certificates by a certification authority). The specification also provides an alternative approach for securing CoAP with IPSec. It argues that many constrained

devices already have support for link layer encryption in hardware which can be used to make IPSec a viable option in such networks.

2.3 Link Format

M2M and smart object networks are envisioned to work without human interaction. In such a scenario, automated discovery of resources hosted on a constrained device is important. The CoRE working group at the IETF is developing the CoRE link format [131] for supporting resource discovery and web linking in constrained device networks. It is similar to the concept of Web Discovery and Web linking [117] defined for HTTP.

The resource discovery mechanism provides a set of Universal Resource Identifiers (URIs) or links [21] 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. The following example shows a typical request and response for resource discovery [131]:

```
REQ: GET /.well-known/core
RES: 2.0 "Content"
</sensors/temp>;rt="TemperatureC";if="sensor"
</sensors/light>;rt="LightLux";if="sensor"
```

In this example, the response indicates two resources hosted on the constrained device: a temperature sensor and a light intensity sensor. The *if* (*interface descriptor*) here indicates the generic REST methods that can be requested for this resource. For example, a sensor would typically support GET requests for obtaining the most recent measured value. The *rt* (*resource type*) attribute associates a semantic type with the resource. The resource type could be an application specific semantic type such as *IndoorTemperatureC* (indicating that the resource is an indoor temperature sensor reporting measurements in degree Celsius), a Universal Resource Name (URN) [103] or a Universal Resource Identifier (URI) [102].

2.4 SenML

While CoAP defines a standard communication protocol for M2M networks, a format for representing sensor measurements and parameters over CoAP

is required. Sensor Markup Language (SenML) [79] defines media types for representing simple sensor measurements and parameters. It has a minimalistic design so that constrained devices with limited computational capabilities can easily encode their measurements and at the same time servers can efficiently collect a large number of measurements. SenML is used to communicate dynamic data originating from the constrained device and static meta-data is communicated out-of-band using the CoRE Link Format. This reduces the message size and improves the decoding efficiency. For example, the CoRE Link format can be used to indicate that the resource is available in the SenML format.

SenML Representations can be defined with JavaScript Object Notation (JSON) [39], eXtensible Markup Language (XML) [30] or Efficient XML Interchange (EXI) [128], all of which share a data model similar to SenML. Our architecture presented in Chapter 4 uses the JSON syntax and an example of a SenML measurement in JSON syntax [79] is as follows:

```
{"e":[{ "n": "urn:dev:ow:10e2073a01080063", "v":43.5,
 "u":"degF" }]}
```

The array elements in the JSON representation are explained in Table 2.1.

SenML	**JSON**	**Data Type**
Measurements	e	Array: Sensor Measurements
Name	n	String: Name of the sensor
Units	u	String: Unit of measurement
Value	v	Floating point: Value of the entry
String Value	sv	String: String value of the entry
Boolean Value	sv	String: Boolean value of the entry
Value Sum	s	Floating Point: Sum of values over Time
Time	t	Number: Time when the value was recorded
Update Time	ut	Number: Maximum time before which sensor will update this value

Table 2.1: SenML Parameter entries

Thus the above example represents a temperature measurement from a sensor named *urn:dev:ow:10e2073a01080063* with a temperature of 43.5 degrees Fahrenheit.

2.5 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 operational phase to save energy. In such scenarios, direct discovery of resources hosted on the constrained server might not be possible. To overcome this barrier, a Resource Directory (RD) [133] can be used. As shown in Figure 2.5, the Resource Directory is an entity that hosts the de-

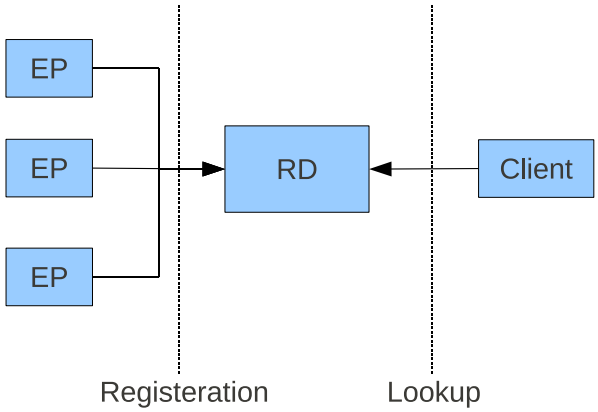


Figure 2.5: Resource Directory

scriptions of resources which are located on other nodes. These resource descriptions are specified as CoRE Link format URIs.

End points (EPs) or constrained servers proactively discover, register and maintain their resources through the interfaces provided by the RD. It is also possible for the RD to proactively discover resources from the EPs and add or validate these entries. A client can use the lookup interface of the RD to discover Web Links describing resources belonging to different EPs. An example of an EP named *node1* registering two resources with the RD using its registration interface is as follows [133]:

```
Req: POST coap://rd.example.com/rd?ep=node1
Etag: 0x3f
Payload:
</sensors/temp>;ct=41;rt="TemperatureC";if="sensor",
</sensors/light>;ct=41;rt="LightLux";if="sensor"
Res: 2.01 Created
Location: /rd/4521
```

In this example, the *ETag* option in the registration request is added to allow the RD to poll the EP and check if the current resource registrations still

exist. The response message from the RD confirms the creation of an entry /rd/4521. This entry is specified by the EP when refreshing or deleting its registrations with the RD.

2.6 Public-key Cryptography

Public-key cryptography relies on the use of two keys: a private key which is kept secret, and a corresponding public key which is disclosed to everyone. In public-key cryptography, a plaintext message is encrypted using the public key, and the encrypted ciphertext is decrypted using the corresponding private key. This establishes a secure communication channel between users having access to the public key and the owner of the corresponding private key. Public-key cryptography can also be used in signature schemes to sign messages. A digital signature of a message is created using the private key, and the authenticity of this signature can be verified by anyone having access to the corresponding public key. Since this cryptographic approach uses asymmetric keys, it is also referred to as asymmetric-key cryptography. Asymmetric-key cryptographic algorithms have been used in a wide range protocols such as Transport Layer Security (TLS) [41], Secure Shell (SSH) [149], Pretty Good Privacy (PGP) [48] and several others.

2.6.1 RSA

RSA is a asymmetric public-key cryptographic algorithm named after its authors Ron Rivest, Adi Shamir and Leonard Adleman who first described the algorithm in 1978. The security of RSA is based on the fact that factorizing a large prime number is complex. In RSA, the product of two large prime numbers, along with an auxiliary value, forms the public key and is disclosed to the public. The prime factors of this product are however kept a secret. This public key can now be used by anyone to encrypt a message, but only the owner of the corresponding private key with the knowledge of the prime factors can feasibly decode the message. The three steps of key generation, encryption and decryption in RSA are performed as follows:

1. Key Generation:
    - Two large prime numbers p and q are selected such that $p \neq q$.
    - Next, $n = pq$ is computed. For acceptable security, the integer n should be at least 1024 bits long.
    - Euler's totient function [93] ϕ is calculated as $\phi(n) = (p-1)(q-1)$.

- An integer e is chosen such that $1 < e < \phi$ and the greatest common divisor (GCD) of e and $\phi(n)$ is 1. This implies that e and $\phi(n)$ are co-primes. The pair n and e form the public key. Although smaller values of e are more efficient, they can be insecure [27]. The commonly used value of e is 65537 ($2^{16} + 1$).
- Finally, d is calculated as $d = e^{-1}\ mod(\phi(n))$ and the pair d and e forms the private key.

2. Encryption: Anyone who has a copy of the public key n, e can now send an encrypted message to the owner of the public-private key pair. If a message M is to be encrypted, it is first converted to an integer m such that $0 < m < n$ using a mutually agreed padding scheme. Next, the ciphertext is determined as $c = m^e (mod\ n)$.

3. Decryption: Once the owner of the public-private key pair receives the encrypted ciphertext, it determines the integer m according to the equation $m = c^d (mod\ n)$. From this integer, the original message M is determined using the reversible padding scheme agreed upon.

2.6.1.1 RSA Signatures

So far we have discussed how RSA can be used for encryption and decryption of messages. However, RSA can also be used for signing messages which can be verified by anyone who owns a copy of the public key of the signer. RSA digital signatures are typically not applied to the whole message but to a hash of the original message. In this signing scheme, the hash $h(m)$ of the message to be signed is raised to the power d *modulo* n and is sent as the signature along with the original message. The receiver then raises this signature to the power e *modulo* n and compares it to the actual hash of the message received. If the two values are consistent, then the authenticity and integrity of the message is confirmed to the receiver. Since this scheme requires the signer to send the original message along with the signature, it is also referred to as *Signature with Appendix*. While using the RSA signature scheme, it is important to pad the plaintext message with structured randomized data before signing, as defined in the PKCS #1 [82] standard. Failure to do so can results in security vulnerabilities shown in the attacks documented by Boneh [27].

2.6.2 Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) was introduced in 1985 separately and independently by Victor Miller [106] and Neal Koblitz [90]. It can be seen as an

elliptic curve variant of the older Discrete Logarithmic Problem (DLP) [104]. We describe the Elliptic Curve Discrete Logarithmic Problem (ECDLP) before discussing the signature scheme based on ECC.

An elliptic curve is represented by the equation:

$$y^2 = x^3 + ax + b$$

where a, b, x and y are real numbers. Several different elliptic curves can be obtained for different values of a and b. As an example, $a = -5$ and $b = 0.7$ results in an elliptic curve represented by the equation $y^2 = x^3 - 5x + 0.7$ which is also illustrated in Figure 2.6. If the curve equation $y^2 = x^3 + ax + b$

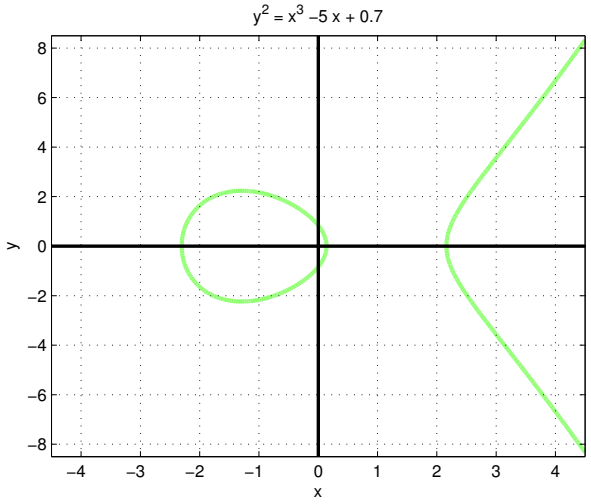


Figure 2.6: Example of an Elliptic Curve

satisfies the condition $4a^3 + 27b^2 \neq 0$ then it does not contain any repeated factors and it can be used to form a group. The points that lie on such a curve along with a special point O, referred to as the *point at infinity*, form an elliptic curve group defined over real numbers.

There are two operations defined on this group:

1. Point Addition: To add two distinct points P and Q on the curve (such that Q $\neq$ -P) a line passing through the two points is drawn. This line intersects the curve at exactly one more point -R and the reflection of this point on the x-axis gives R, which denotes the sum of P and Q. This is elucidated in Figure 2.7. Adding a point P to its negative -P results in the point at infinity O, and hence -P is the additive identity of P.

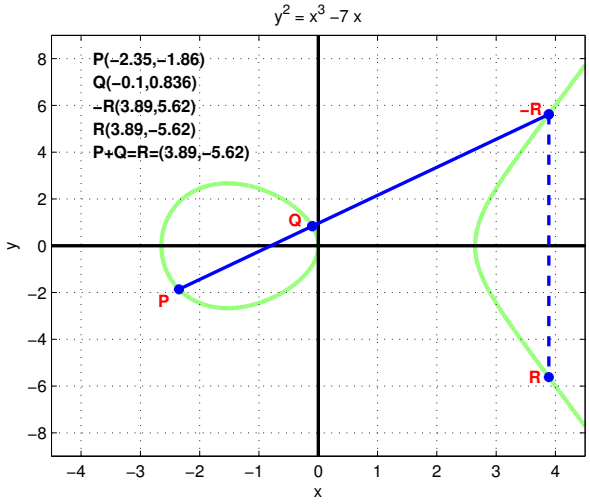


Figure 2.7: Point Addition

2. Point Doubling: To add a point P to itself, a tangent line is drawn at P. This tangent intersects the curve at exactly one other point -R. The reflection of this point on the x-axis gives R, which denotes the result of the doubling operation. This is also show in Figure 2.8. If however, P is on the x-axis, the tangent will always be vertical and therefore $2P = O$, the point at infinity.

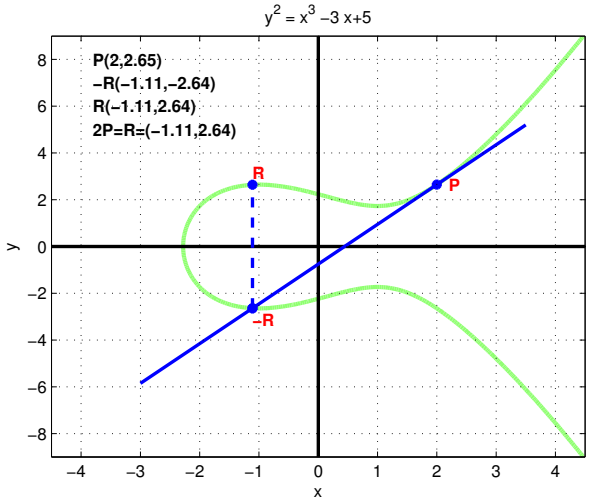


Figure 2.8: Point Doubling

Performing these operations of point addition and point doubling over real

numbers is slow and inaccurate due to rounding errors. Applications, however, require fast and precise mathematics and therefore, elliptic curve groups are defined over finite fields such as prime fields (F_p) and binary fields (F_{2^m})).

We have seen that a point P in an elliptic curve group can be doubled to obtain $2P$ and then can be added to itself to obtain $3P$. Determination of a point nP in this manner, with repeated doubling and addition operations, is known as *scalar multiplication* of P. The Elliptic Curve Discrete Logarithmic Problem (ECDLP) is defined as: given two points P and Q in an elliptic curve group, find a number k, such that $Pk = Q$; where k is referred to as the discrete logarithm of Q to the base P. Applications based on ECDLP are designed such that k is very large to make it infeasible for guessing k by repeated addition and doubling operations. The basis for security of elliptic curve crypto-systems is the computational complexity of solving ECDLP. Since this variant is significantly harder than the original DLP, the strength per key bit is greater in ECC systems than conventional DLP systems. Thus, smaller parameters are used in ECC for equivalent levels of security. This is extremely useful for constrained devices and smart object networks where processing power, bandwidth and power consumption are constrained.

2.6.2.1 ECDSA

Elliptic Curve Digital Signature Algorithm (ECDSA) is an elliptic curve analog of the Digital Signature Algorithm (DSA) [57], first proposed by Scott Vanstone [142]. The operations of key generation, signature generation and signature verification over prime field (F_p) is as follows [10]:

1. Key generation:
    - An Elliptic Curve E over F_p is selected such that the number of points in the field is divisible by a large prime number n.
    - A point $P \in F_p$ of the order of n is selected.
    - A random integer d in the range $[1, n-1]$ is selected and it forms the private key.
    - Next, $Q = dP$ is computed.
    - The set (E, P, n, Q) forms the public key.
2. Signature Generation
    - A random integer k in the range $[1, n-1]$ is chosen.
    - $(x_1, y_1) = kP$ is determined and then used to find r where $r = x_1 \ mod \ n$. If however, $r = 0$, then the previous step is repeated by choosing a new value for k.

- The hash of the message to be signed is computed as $h(m)$ and is used to determine s, where $s = k^{-1}(h(m) + dr)\ mod\ n$. If $s = 0$, then the process is started again from step 1 and a new value for k is chosen. This because if s is zero then $s^{-1}\ mod\ n$, which is needed during signature verification, ceases to exist.
- The pair of integers (r, s) forms the signature and is sent along with the original message.

3. Signature Verification

- The receiver verifies that r and s are integers in the range $[1, n-1]$.
- The hash of the message received $h(m)$ is computed along with w, where $w = s^{-1}\ mod\ n$.
- Next, u_1 and u_2 are calculated as, $u_1 = (h(m).w\ mod\ n)$ and $u_2 = (r.w\ mod\ n)$.
- Finally, $u_1P + u_2Q = (x_0, y_0)$ is computed to determine v, where $v = x_0\ mod\ n$.
- The signature is accepted as valid only if $v = r$.

ECDSA is also a *Signature with Appendix* scheme similar to RSA and requires the original message to be sent along with the signature. The key generation operation in both schemes require entropy to generate random numbers. However, unlike RSA, ECDSA not only requires randomness during key generation but also during each signature operation. Thus, it is essential to provide appropriate entropy for each signature operation in ECDSA.

2.7 Javascript Object Notation (JSON) Object Signing and Encryption

Javascript Object Notation (JSON) is a lightweight text representation format for structured data [39]. It is often used for transmitting serialized structured data over the network. The JSON Object Signing and Encryption (JOSE) working group at IETF is developing standards for interoperability of security features between protocols. The working group is developing a scheme for encoding public keys as JSON objects. This scheme is known as JSON Web Key (JWK). It is also developing a JSON representational format for depicting signed content called as JSON Web Signatures (JWS). We describe the JWS and JWK formats with examples in the following subsections.

2.7.1 JavaScript Object Notation (JSON) Web Key (JWK)

JSON Web Key (JWK) [80] is a data structure used for representing public keys as JSON objects. The JWK representation of a public key consists of JSON-object members that describe the characteristics of the key such as the public-key algorithm used. While some members are common to all the public keys independent of the cryptographic algorithm used, there are other members which are specific to the public-key cryptographic algorithm used. The common members are as follows:

- alg: Identifies the cryptographic algorithm family used with the key.
- use: An optional member which indicates whether the key is used for signing or for encryption.
- kid: A key identifier responsible for matching keys, and can be used to identify a key during a key rollover. The kid member is also optional and can be excluded from JWK.

All integers in this representation are base64url encoded. The following is an example of a JWK object representing an ECC key [80]:

```
 {"jwk":
  [
    {"alg":"EC",
     "crv":"P-256",
     "x":"MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
     "y":"4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
     "use":"enc",
     "kid":"1"},
  ]
}
```

The members *crv*, x and y are specific to ECC keys and indicate the elliptic curve used to create the key along with the x and y coordinates of the elliptic curve point representing the public key.

2.7.2 JavaScript Object Notation (JSON) Web Signatures (JWS)

The JSON Web Signature [81] 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 and RSA. This format is independent of the content and therefore can be used with any arbitrary data.

A JWS representation consists of three parts: the JWS header, the JWS Payload, and the JWS Signature. A 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 signature or the HMAC algorithm over the header and payload. The header, payload and the signature are encoded in base64url and concatenated with period characters. The following is an example of a JWS header [81] in the Universal Character Set Transformation Format 8-bit (UTF-8) [148] format:

```
{"typ":"JWT",
 "alg":"HS256"}
```

Here the *alg* parameter identifies the cryptographic algorithm used for securing the content of the payload while the *typ* parameter indicates the type of content being secured. An example JSON payload which can be secured with JWS is as follows [81]:

```
{"iss":"joe",
 "exp":1300819380,
 "http://example.com/is_root":true}
```

The HMAC or the cryptographic signature is applied over the header and the payload concatenated with a period character in UTF-8 representation. The result is encoded in base64url format and along with the base64url encoding of the header and payload forms the JWS representation as follows [81]:

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
.
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkz
ODAsDQogImh0dHA6Ly9leGFtcGxlLmNvbS9pc19y
b290Ijp0cnVlfQ
.
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

Chapter 3

Public-key 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 [84, 96, 119–121]. The underlying assumption in all of them is that public-key cryptography is far too resource and energy intensive for actual implementation and deployment on resource 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 which make this scheme undesirable for networks with a large number of nodes. In addition, perfect forward secrecy cannot be guaranteed once the key of a node is compromised. Conversely, if a single symmetric key is shared among all the nodes in the network, the memory requirement for each individual node is greatly reduced, but it also results in lower network resilience to key compromise.

In response to these problems, many probabilistic key distribution schemes [34, 46, 51] for symmetric cryptographic algorithms have been proposed. These schemes either need pre-distribution of keys, which requires a more complex configuration during the provisioning, or larger amount of network traffic, which results in higher energy consumption. In general, symmetric-key schemes do not offer the flexibility of not having pre-shared keys which is provided with public-key asymmetric cryptography. Moreover, there have been several studies that contradict the aforementioned assumption and we start the chapter by discussing some of the previous work that has been done in implementing public-key cryptography on small devices in Section 3.1. We then describe the publicly available code sources that can be used to implement RSA or ECC based asymmetric cryptography on 8-bit platforms in Section 3.2. Finally, in Section 3.3, we document the performance of these publicly available libraries.

3.1 Previous Experiments with Asymmetric Cryptography

Gura et al. [69] carry out a performance comparison of RSA and ECC based public-key cryptographic schemes on 8-bit micro-controllers. For maximizing the efficiency of the RSA algorithm, they include several well known mathematical optimizations such as the Chinese remainder theorem for fast verification [42], Montgomery multiplication [109] and an optimized squaring method that takes advantage of partial products [69]. The ECC implementation incorporates techniques for efficient elliptic curve operations which include a projective co-ordinate system for inversion [37], the Non-Adjacent Forms (NAF) method for recording a scalar quantity during point multiplication to reduce the number of point additions [110], and recommended curve specific optimizations for modular reduction [2, 4]. The modular multiplication algorithm for RSA and ECC uses 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 a 8-bit processor with a clock speed of 8 MHz.

Blaß and Zitterbart [24] implement elliptic curve public-key cryptography on an 8-bit ATmega128 micro-controller. Their work argues that 113-bit elliptic curve fields provide sufficient security for the current hardware (in 2005). By using optimizations such as pre-computation for faster point multiplication and loop unrolling 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 [29], making 113-bit elliptic curves vulnerable to attacks as well. In general, it is suggested that modern day applications use 128-bit curves or higher to ensure sufficient security.

The work done by Uhsadel et al. [138] accomplishes a standards-compliant 160-bit curve based signature generation operation in 2881 cycles (0.39 seconds) on an 8-bit 8 MHz micro-controller. This performs faster than the work by Gura et al. [69] discussed earlier. The entire prime field arithmetic is implemented in hardware assembly along with an improved modular multiplication scheme. The authors claim that the work provides the fastest known implementation of 160-bit elliptic curve point multiplication (in 2007).

Hassan and Qamar [72] in their thesis evaluate the feasibility of asymmet-

ric public-key cryptography on the Contiki Operating System[1]. The thesis provides a comprehensive performance comparison of two libraries, namely *Libtomcrypt* [40] and *Relic* [13] on the MSP430F1612 micro-controller [15] and on the COOJA simulator [50]. Both libraries implement a fairly large number of cryptographic algorithms and the authors choose to evaluate ECC based signature generation and verification performance in terms of execution time, Static Random Access Memory (SRAM) consumption, flash memory usage and energy consumption. Although neither MSP430F1612 nor COOJA are 8-bit platforms, unlike Libtomcrypt, Relic does provide support for 8-bit platforms.

Sizzle, a standards-based end-to-end security architecture for the embedded internet [68], not only performs public-key cryptography on 8-bit platforms but implements an entire web stack with a fully functional Secure Sockets Layer (SSL) [60] suite based on ECC. The implementation can perform an entire SSL handshake on 8-bit 8 MHz micro-controller with 4 kB of SRAM in 1 second and can send 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 [33, 144, 150]. With all the attempts 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 its code available online for download and use. Therefore, we set out to find code sources that are available online and can be easily ported for use on 8-bit platforms within a short duration of time. We were able to find four such libraries and evaluate their performance in a period of two working weeks.

3.2 Available Cryptographic Libraries

We provide a brief description of the libraries which are suitable for such platforms and are publicly available:

- AvrCryptolib [140] : This library provides a variety of symmetric-key cryptographic algorithms such as DES, Triple DES, AES and RSA as

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

an asymmetric public-key algorithm. We stripped down the library to use only the required RSA components for our performance analysis. AvrCryptolib only performs modular exponentiation and does not implement reversible padding schemes as suggested in standards such as PKCS #1 encryption algorithm version 2.1 [82]. It implements the modular exponentiation functions in AVR 8-bit assembly language with C-interfaces to reduce the execution times. The library also provides an option to store the keys in flash memory and allows direct access to them, thus saving the amount of SRAM consumed. This feature takes advantage of the fact that Arduino boards allow the programmer to directly address the flash memory to access constant data during execution.

- Relic-Toolkit [13]: This library is entirely written in the C language and provides a highly customizable implementation of a large variety of cryptographic algorithms. This not only includes RSA and ECC, but also pairing based asymmetric cryptography, Boneh-Lynn-Schacham short signatures [28], Boneh-Boyen short signatures [26] and many other algorithms. The library provides an option to build and include only the desired components for the specified platform. While building the library, it is possible to select a variety mathematical optimizations that can be combined to obtain optimal performance. Relic implements prime and binary field arithmetic along with preliminary support for ternary field arithmetic. It includes a multi-precision integer math module, which can be customized to use different bit-length words, thus making it easy to compile for a variety of platforms ranging from 8-bit AVRs to 64-bit x86 machines. There is very little documentation available but it appears to be a very promising library with a large number of algorithms and optimizations implemented.

- TinyECC [95]: TinyECC was designed for using elliptic curve based public-key cryptography on constrained devices. It is written in the Network Embedded Systems C (nesC) programming language [64] and is designed for use on TinyOS [94]. However, the library can be ported to standard C99 either with tool-chains or by manually rewriting parts of the code. This allows 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 [70] and Barrett reduction for verification [105]. It also has one of the smallest SRAM consumption among the set of elliptic curve libraries surveyed so far. However, while Relic implements curves over prime and binary fields,

TinyECC only implements curves over prime fields.

- Wiselib [17]: Wiselib is a generic library written for sensor networks containing a wide variety of algorithms. While the stable version of the library contains algorithms for routing only, the test version includes algorithms for cryptography, localization, topology management among others. The library was designed for smooth integration with operating systems such as iSense and Contiki. However, since the library is written entirely in C++ with a template based model similar to the Computational Geometry Algorithms Library (CGAL) [52], it can be used on any platform directly without using any of the operating system interfaces provided. It implements elliptic curves over prime fields only. In order to make the code platform independent, no assembly level optimizations were incorporated. Since efficiency was not an important goal for the authors of the library while designing, many well known theoretical performance enhancements were also not incorporated.

3.3 Performance Analysis

For implementing and experimenting with public-key cryptography in resource constrained environments, we chose the Arduino Uno board[2] as the test platform. Arduino Uno has a 8-bit ATmega328 micro-controller with a clock frequency of 16 MHz, 2 kB of SRAM, and 32 kB of flash memory. Although 32-bit platforms such as ARM Cortex-M0+[3] are available at roughly the same cost and consume approximately the same amount of energy, we intentionally choose an 8-bit platform to demonstrate that our security architecture can be implemented even on extremely constrained platforms.

In order to measure the SRAM consumption for our experiments, we use the Avrora simulator [137]. Since all the libraries and our code use only a stack based allocation scheme, the stack trace produced by the simulator gives an accurate value for the SRAM consumption. The execution times were calculated on Arduino boards using the on-board ATmega internal oscillator which provides an accuracy of four microseconds.

We have summarized the results of raw RSA private-key modular exponentiation operation using the AvrCryptolib library on Arduino Uno in Table 3.1. In order to perform a comprehensive benchmark of this library, we perform a number of experiments with different key lengths. We generated

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

[3]ARM Cortex-M0+: http://www.arm.com/about/newsroom/worlds-most-energy-efficient-processor-from-arm-targets-low-cost-mcu-sensor-and-control-markets.php

Key Length (bits)	**Execution Time (ms): Keys in SRAM**	**SRAM consumption (bytes): Keys in SRAM**	**Execution Time (ms): Keys in flash**	**SRAM consumption (bytes): Keys in flash memory**
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

Table 3.1: RSA private-key modular exponentiation performance

public-private key pairs of lengths ranging from 64 to 2048 bits separately before using them with the library. The keys were generated with the value of the public exponent e as three and were hard-coded into the program. We performed two different sets of experiments for each key size. In the first case, the keys were copied into the SRAM from the flash memory before being used by any of the functions. In the second case, the keys were addressed and used directly from the flash. The execution times were calculated from the mean of five experiments rounded off to the nearest millisecond. The SRAM consumption indicated in Table 3.1 only reflects the requirements for RSA private-key modular exponentiation operation and does not depict the SRAM consumption of the entire program.

It can be seen in Table 3.1 that the performance of raw RSA private-key modular exponentiation was faster for smaller keys and we were able to achieve 64-bit RSA private-key modular exponentiation in 64 ms. With longer keys, the execution time increased exponentially and, for key lengths of 1024 bits, the resulting execution time was about three minutes. We also observed that when the keys were used directly from the flash memory, the SRAM consumption was reduced as the keys were no longer copied to the SRAM, but the execution times were significantly longer as reading from flash memory is considerably slower than reading from the SRAM. Our results for 64-bit and 512-bit raw RSA private-key modular exponentiation execution times concur with the reported values for a similar 16 Mhz platform [140]. The code size (flash memory consumption) for the experiments approximated to about 2.6 kilo bytes (kB). We did not focus on reducing the amount of flash consumed as it is available at a nominal cost and is generally not the

limiting factor for such devices.

It is also worth noting that this implementation performs basic modular exponentiation and multiplication operations without any of the well-known mathematical optimizations such as Montgomery multiplication [109], optimized multiplication and squaring [69, 110] used by Gura et al. [69] which enhance the performance significantly by consuming marginally larger amounts of SRAM. If larger SRAM consumption is acceptable, we believe that 1024 and 2048-bit RSA operations can be performed with greater efficiency as has been previously documented [69, 144]. Nonetheless, if in some scenario a delay of some tens of minutes is acceptable, 2048-bit RSA is possible with 1 kB of SRAM. Finally, it is important to point out that experiments for raw RSA public-key modular exponentiation were not performed as our security architecture presented in the next chapter does not require it to be implemented on resource constrained devices.

Next, we evaluate the performance of the remaining three libraries that perform elliptic curve encryption, decryption and signature operations. Elliptic curve cryptography works on elliptic curve groups defined over elliptic curves. There are two standardization bodies, namely Standards for Efficient Cryptography Group (SECG) and National Institute of Standards and Technology (NIST), that recommend curve parameters which are proven to be secure and efficient. The curves used in our experiments are listed in Table 3.2.

Curve	**Strength**	**RSA**
SECG Curves		
secp128r1, secp128r2	64	704
secp160k1, secp160r1, secp160r2	80	1024
secp192k1, secp192r1	96	1536
NIST Curves		
nist k-163, nist b-163	80	1024
nist k-233, nist b-233	112	2048

Table 3.2: ECC Curves and their Security Strengths

In Table 3.2, the second column denotes the approximate number of bits of security the curve parameter offers, and the third column denotes the approximate size (in bits) of a RSA modulus with comparable strength [2, 4]. There are two categories of curves defined over binary and prime fields. The first category comprises of provably random curves which are generated with a particular seed and a hash function. For binary fields, the second category comprises of a set of anomalous curves where a and b in the curve equation

$y^2 + xy = x^3 + ax^2 + b$ are chosen such that $a, b \in \{0, 1\}$. These curves exhibit especially efficient performance and are referred to as Koblitz curves [91]. On the other hand, for prime fields, the term Koblitz is generalized to refer to curves that exhibit efficient endomorphism [61] and comprise the second category for prime fields.

We only evaluate the performance of ECDSA signature generation operation for the three libraries as our security architecture, presented in the next chapter, does not require signature verification on constrained devices. The default rand() function provided in Arduino for generating random numbers is used for generating the public-private key pair and each time a message is signed. We choose to seed the random number generator with a common seed of 300 for all the experiments so that they can be repeated to reproduce the results.

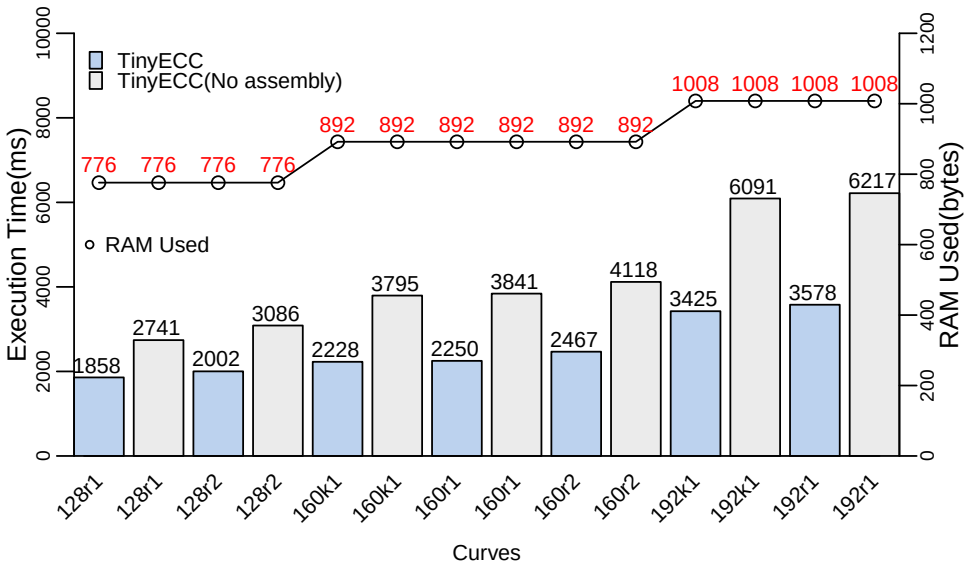


Figure 3.1: TinyECC Performance

Evaluation of TinyECC signature generation with ECDSA is illustrated in Figure 3.1. We had re-written the nesC code into native C99 for experimenting it on Arduino Uno. We evaluate a variety of standard SECG prime-field curves that are supported by TinyECC and provide different levels of security. The library implements several optimizations which include a projective co-ordinate system [37], Barrett Reduction [105], Shamir Trick for signature verification [70], sliding window for efficient scalar multiplication [70], hybrid multiplication and squaring [69], and curve specific optimizations [2, 4]. However in our implementation we only use the projective

coordinate system, sliding window optimization for scalar multiplication and SECG recommended curve specific optimizations. While Shamir trick was not implemented as it only increases the efficiency of signature verification, hybrid multiplication and squaring were omitted because they were written in assembly language that was incompatible with the Atmega328p microcontroller. Liu and Ning [95] 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.

Figure 3.1 shows two different test cases: one with all the assembly optimizations enabled, and the second with only standard C99 code. It is visible in the figure that assembly optimizations can increase the efficiency of signature generation and their effectiveness increases for larger prime fields. We also observe that curves on larger prime fields perform slower than those defined over smaller prime fields. While 128-bit curves such as 128r1 perform signature generation in 1858 milliseconds, 192-bit curves such as 192r1 requires 3578 milliseonds. Finally, Koblitz curves (shown as 160k1 and 192k1) prove to be more efficient than their pseudo-random counterparts (shown as 160r1,160r2 and 192r1) because they exhibit efficient endomorphism [61].

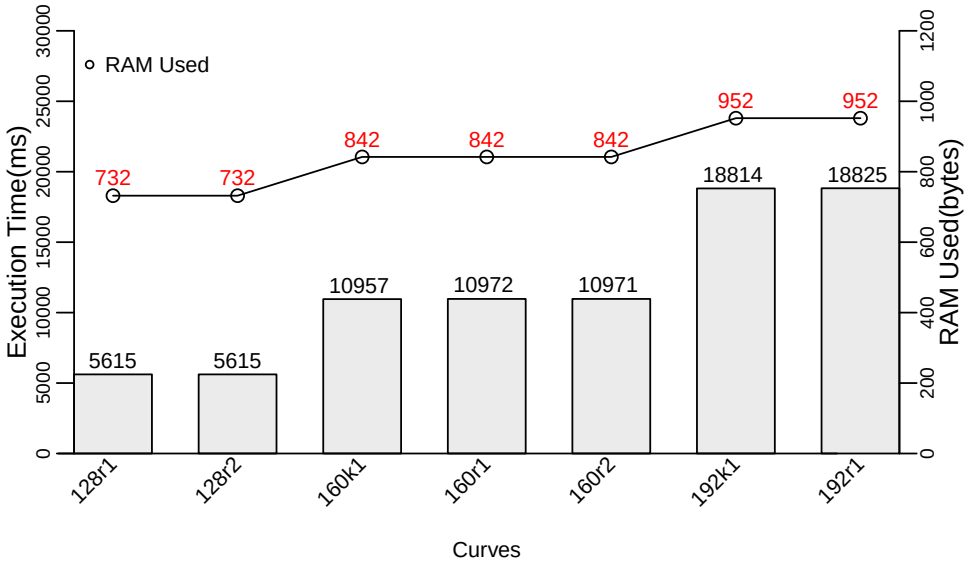


Figure 3.2: Wiselib Performance

Similar to TinyECC, Wiselib also implements curves over prime-fields only. The performance of Wiselib is depicted in Figure 3.2. It can be observed from the figure that the efficiency of Wiselib is significantly inferior to

TinyECC for all the curves. While TinyECC can execute 160-bit secp160k1 curve in 2228 milliseconds, wiselib takes 10957 milliseconds for the same curve. The authors of the library did not implement any of the well-known mathematical optimizations which leads to inferior performance. However, since Wiselib is entirely coded in C++, unlike TinyECC, it can easily be used on a wide variety of sensor platforms.

While experimenting with the Relic library, we were unable to use the Arduino Uno board because of SRAM constraints. We therefore chose to analyze its performance on Arduino Mega[4] which has a similar 8-bit microcontroller (ATmega2560) as the Uno but has more SRAM (8 kB) and flash (128 kB). Although Relic implements curves over binary as well as prime fields, previous work [11, 83] demonstrates that binary fields are more suited for Atmel platforms used on Arduino boards. Therefore, we only experiment with curves over binary fields during our evaluation. In order to thoroughly benchmark the library, we tested two different configurations of the library for curves over binary fields: one which resulted in the least execution time (Relic-fast) and another which consumed the least amount of memory (Relic-lowmem). The two configurations use different mathematical optimizations which are detailed in Appendix A.

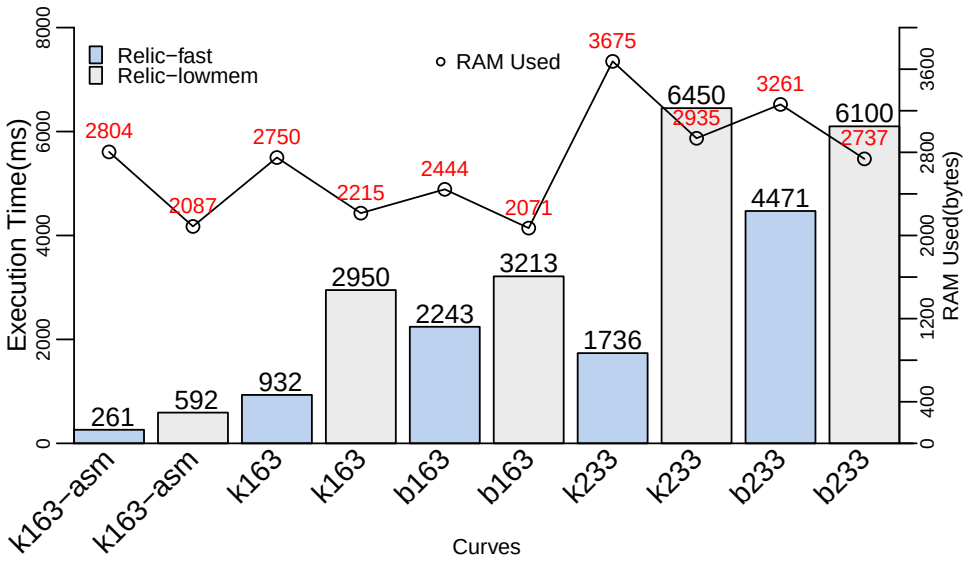


Figure 3.3: Relic Performance

We examine the performance of four standard NIST curves over binary

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

fields (NIST-K163, NIST-B163, NIST-K233 and NIST-B233 shown as k163, b163, k233 and b233 respectively). While experimenting with Koblitz curves over binary fields, we tested a special case where the binary arithmetic was implemented in assembly (shown as k163-asm). Figure 3.3 shows the execution times and SRAM consumption for all the curves. It can be observed that relic can provide 80-bit security with the curve labeled as k163-asm in just 261 milliseconds. In Figure 3.3, we observe that Koblitz curves perform much faster than their pseudo-random counterparts. However, for the curves k233 and b233, we encountered an anomalous behavior where the pseudo-random curve performed faster than the Koblitz equivalent. On further investigation, we discovered that this anomalous behavior was encountered because of an extremely slow implementation of the tau-NAF [134] method for scalar multiplication which affects Koblitz curves only.

Another important inference that can be derived from these graphs is the fact that the difference in efficiency of Koblitz curves defined over binary fields and their pseudo-random counterparts is greater than the difference between Koblitz and pseudo-random curves over prime fields. Our results for Relic do not concur with those of Aranha et al. [12] since the math backend used by them is not part of version 0.3.1 available online.

We did not focus any of the experiments towards the flash memory consumption of the libraries as flash memory is available at a low cost and is generally not the limiting factor on constrained devices. However, as a rough estimate Wiselib consumed about 17 kB of ROM and TinyECC about 20 kB of ROM while relic took 27-40 kB of ROM depending on the configuration. The results for TinyECC differ from those reported in [95] as we no longer use the original nesC code and re-wrote the entire library in standard C99.

The energy consumption for the three libraries is depicted in Table 3.3. The energy consumption was calculated using the formula:

$$W = U * I * t$$

where U is the operating voltage (5V), I is the current drawn (0.01A for Atmega328p and 0.02A for Atmega2560 [147]) and t is the execution time. This approach was also taken by the authors of TinyECC [95] to measure the energy consumption on different platforms.

From these results it is evident that software-only public-key cryptography is not only possible, but quite efficient on resource constrained devices with publicly available libraries. There is scope for further performance enhancement by using additional optimizations. With Relic, we were also able to achieve 112-bit (2048-bit RSA) security over binary fields in 1,736 seconds and 3,675 bytes of SRAM. As the computational capacity of these

constrained-devices increases and their cost reduces, achieving public-key cryptography on such devices would become simpler. In the next chapter we present our security architecture which is based on using public-key cryptographic signature schemes on constrained devices.

Library	**Curve**	**Energy Consumption (mJ)**
TinyECC	128r1	92.90
	128r2	100.10
	160k1	111.40
	160r1	112.50
	160r2	123.35
	192k1	171.25
	192r1	178.90
TinyECC (no assembly)	128r1	137.05
	128r2	154.30
	160k1	189.75
	160r1	192.05
	160r2	205.90
	192k1	304.55
	192r1	310.85
Wiselib	128r1	280.75
	128r2	280.75
	160k1	547.85
	160r1	548.60
	160r2	548.55
	192k1	940.70
	192r1	941.25
Relic-fast	k163-asm	26.10
	k163	93.20
	b163	224.30
	k233	173.60
	b233	447.10
Relic-lowmem	k163-asm	59.20
	k163	295.00
	b163	321.30
	k233	645.00
	b233	610.00

Table 3.3: Energy consumption for TinyECC, Wiselib and Relic

Chapter 4

Architecture

In this chapter we present our secure and energy-efficient communication architecture based on public-key cryptography along with provisioning schemes and message freshness mechanisms. 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 while they sleep and appear to be always online.

4.1 Mirror Proxy

In order to achieve these contradicting goals, we use Mirror Proxies [143] to delegate the task of serving data from smart objects to proxies. The concept of Mirror Proxy (MP) is illustrated in Figure 4.1. A Mirror Proxy is an entity responsible for caching and serving data to clients on behalf of sleeping constrained smart object servers also referred to as Sleeping Endpoints (SEPs). A Mirror Proxy is assumed to have sufficient computational power and energy supply to remain online and serve data collected from several SEPs. The smart objects no longer operate as servers that serve to client requests for data and rather act as clients of the MP themselves. A Sleeping End Point registers its resources with the MP. When the MP receives a registration request, it adds the resources of the SEP into its own resource tree as sub-resources. It also 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.

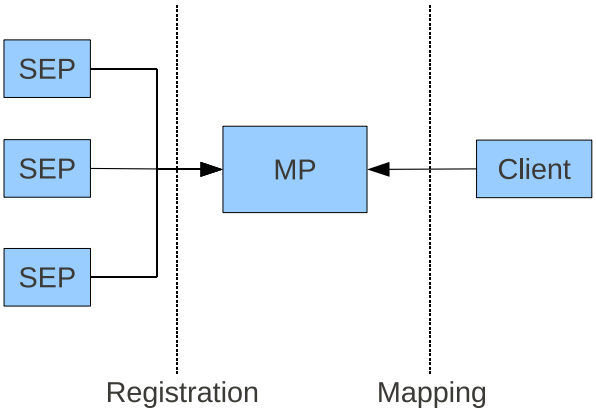


Figure 4.1: Mirror Proxy

A Mirror Proxy is similar to a caching reverse proxy except for the fact that it caches content from an origin client rather than an origin server. 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. Such a communication model might be acceptable 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 some data updates is tolerable.

A Sleeping End Point can determine the location of the MP using several different mechanisms. As an example, the IP address or the URL of the MP may be hard-coded into SEPs at the time of manufacture. However this approach is not only inflexible but using URLs would also require the SEPs to support the Domain Name System (DNS) [108]. Alternatively, the MP location may be configured with Dynamic Host Configuration Protocol (DHCP) [45] or by placing the MP on the border router in a 6LowPAN [114] network. Finally, if a SEP knows the location of the Resource Directory (RD), it can use the RD to contact the MP.

The location of the MP in the network topology can vary depending on the deployment scenario. While for some deployments a single central MP could serve all the SEPs, it can be advantageous to have several distributed MPs in certain scenarios. A number of distributed MPs in the immediate proximity of the SEPs not only ensures small Round Trip Times (RTT), but also allows the SEPs to avoid global connectivity. Nonetheless, relying on local connectivity between the SEPs and the MP can also lead to network fragility because of device mobility and radio signal propagation variations.

In our architecture, as depicted in Figure 4.2, we deploy a Mirror Proxy

to allow sleeping nodes to serve data through caches maintained in the MP. Although we use a single centralized MP in our architecture, multiple distributed MPs can be added with no additional complexity. The entire communication network between the SEPs and the MP relies on using CoAP over UDP. A sleeping end point can also indicate the duration for which the MP should cache the updates, and the MP can choose to uphold or ignore this request.

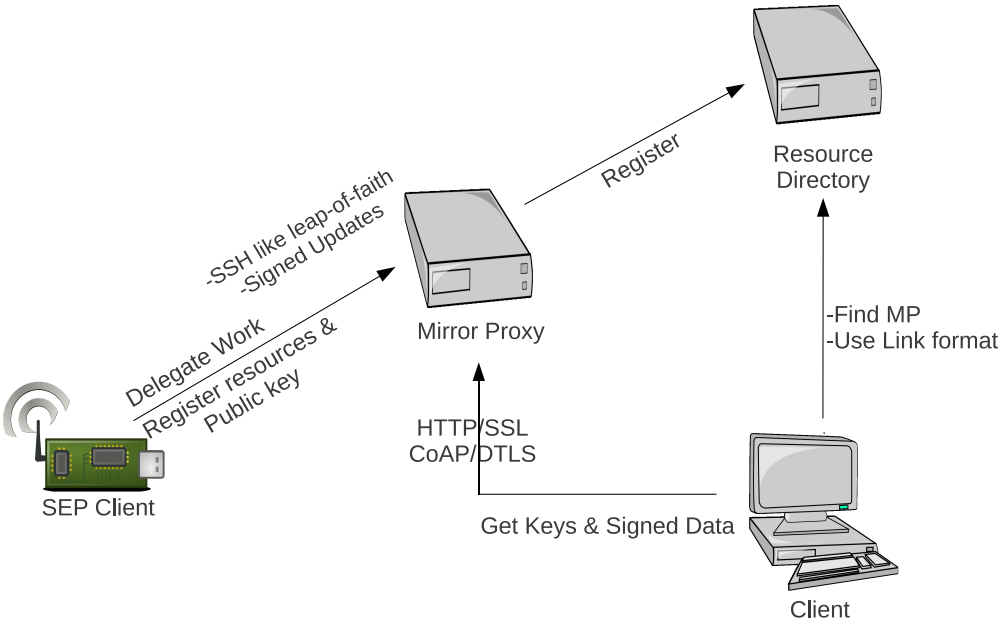


Figure 4.2: System Architecture

4.2 Secure Communication

In order ensure security in this communication model, our architecture requires each constrained device to have a public-private key pair. The architecture is independent of the asymmetric cryptographic algorithm chosen for generating the key pair. However, our results in Chapter 3 and results from previous work [69] suggest that ECC based cryptography 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 4.2, when a smart object sends a registration request to the MP, it also adds its public key in the registration message. A Mir-

ror Proxy is responsible for storing the public key of each of the SEP that it serves. The SEPs wake-up pre-determined intervals and use the corresponding private key to sign all subsequent data updates sent to the MP. The public keys and the signed content are sent in the standard JSON-based JWK [80] and JWS [81] formats respectively.

This system essentially provides a SSH-like leap of faith system 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. However, leap-of-faith is vulnerable to active man-in-the-middle attacks and, to counter these, we suggest some secure provisioning methods in Section 4.5. 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.

4.3 Retrieving Data Updates

When a smart object registers with the MP, the MP adds the resources to its own resource tree and updates its ./well-know/core to reflect the same. The MP also registers the new resources as 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.

The client first obtains the public key of a Sleeping End Point and then retrieves the signed data updates. 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 4.2, occurs over HTTP [55] or CoAP [132] and is protected with SSL [60] or DTLS [126] respectively. The MP has sufficient computational power and energy resources to securely serve this data. 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 [60] or DTLS [126] 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 some denial-of-service attacks.

4.4 Freshness

A replay attack occurs when a malicious entity records a packet (which may be encrypted) and replays it at a later time. In our architecture, if implemented as described thus far, messages along with their signatures sent from the SEPs to the MP can be recorded and replayed by an eavesdropper. The MP has no mechanism to distinguish previously received packets from those that are retransmitted by the sender or replayed by an eavesdropper. Therefore, it is essential for the SEPs to ensure that data updates include a freshness indicator. However, ensuring freshness on constrained devices can be non-trivial because of several reasons which include:

- Communication is mostly unidirectional to save energy.
- Internal clocks might not be accurate and may be reset several times during the operational phase of the SEP.
- Network time synchronization protocols such as Network Time Protocol (NTP) [107] are resource intensive and therefore may be undesirable in many smart object networks.

There are several different methods that can be used in our architecture for replay protection. The selection of the appropriate choice depends on the actual deployment scenario.

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 it 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 [75], where the sender and receiver maintain and sign long sequence numbers of equal bit-lengths but they transmit only the least significant bits.

It is important for the SEP to write the sequence number into the permanent flash memory after each increment and before it is included in the message to be transmitted. This ensures that the SEP can obtain the last sequence number it had intended to send in case of a reset or a power failure. However, the SEP and the MP can still end up in a discordant state where the sequence number received by the MP exceeds the expected sequence number by an amount greater than the preset threshold. This may happen because of a prolonged network outage or if the MP experiences a power failure for some reason. Therefore it is essential for SEPs that normally send Non-Confirmable data updates to send some Confirmable updates and re-synchronize with the MP if a reset message is received. The SEPs re-synchronize by sending a new registration message with the current sequence number.

Although sequence numbers protect the system from replay attacks, a 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. It may be important in certain smart object networks for SEPs to send data updates which include timestamps to allow the MP to determine the time when the update was created. For example, when the MP is collecting temperature data, it may be necessary to know when exactly the temperature measurement was made by the SEP. A simple solution to this problem is for the MP to assume that the data object was created when it receives the update. In a relatively reliable network with low RTT, it can be acceptable to make such an assumption. However most networks are susceptible to packet loss and hostile attacks making this assumption unsustainable.

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 also obtain the current time from NTP, but this may consume additional energy and give rise to security issues discussed by Mills [107]. The SEPs could also have access to a GSM [113] network or the Global Positioning System (GPS) [135], and they can be used obtain the current time. Finally, if the SEPs need to co-ordinate their sleep cycles, or if the MP computes an average or mean of updates collected from multiple SEPs, it is important for the network nodes to synchronize the time among them. This can be done by using existing synchronization schemes [49, 62, 141].

4.5 Provisioning

We have discussed previously that the leap-of-faith registration mechanism proposed in our architecture is vulnerable to active man-in-the-middle attacks and it is also possible to overwhelm the MP by sending registration messages from several malicious SEPs. Therefore it is important to authenticate the SEPs in the network to a Mirror Proxy. However, securely pairing smart objects to other network entities is difficult because of several reasons which include:

- Many smart objects lack even the most basic user interface.
- Provisioning should not be resource intensive as otherwise a substantial amount of battery may be drained even before the smart object enters the operational phase.
- Cost of providing additional auxiliary interfaces such as a RFID tags only for provisioning may not be feasible economically.

There are several techniques for secure pairing and device authentication that have been developed over the years. Gollakota et al. [66] have developed a secure in-band pairing mechanism for Wireless Local Area Network (WLAN) [3] devices and it remains to be seen if a similar approach can be developed for other smart object networking technologies such as Zigbee [6] and Bluetooth low energy [5]. Work by Cheneau et al. [35] uses Cryptographically Generated Addresses (CGAs) [14] for secure bootstrapping of devices in constrained networks. Proximity-based pairing schemes [77, 125] can also be used in smart object networks if the radio signal propagation is guaranteed to be confined within a particular area. There are several out-of-band pairing mechanisms [65, 67, 127] that can also be used in smart object networks.

While these pairing mechanisms are easy to implement, they generally tend to be slow and may sometimes require additional auxiliary user interfaces.

In order to securely authenticate the SEP to the MP, we describe two low-cost and efficient out-of-band provisioning mechanisms that do not require any auxiliary user-interfaces on constrained devices.

The first provisioning mechanism is aimed at domestic deployments and assumes that the MP has a suitable display for indicating a list of securely authenticated SEPs. As an example, the MP could be a mobile phone or home computer allowing the owner to view all the paired SEPs. It also requires the hash of the public key owned by the SEP to be included in a human-readable format. Farrell et al. [53] describe several methods that can be used to represent hashes in a human-readable format.

In this provisioning scheme, when a user turns on a previously unpaired SEP, it sends a HELLO request message containing its public key to register with a Mirror Proxy. If the location of the MP is already known through one of the several techniques discussed in Section 4.1, then the register message is unicasted. If however the location is unknown, then the registration message is broadcasted over the network. A Mirror Proxy that receives a request to register displays the hash of the public key received in the registration message. The user now verifies if the human-readable hash of the public key provided with the SEP matches the one displayed by the MP. It the two match, the user approves the pairing to continue and the MP acknowledges a successful registration to the SEP. However, if the registration message is received by an incorrect or malicious MP, the user would not notice any registration requests on its own MP. The user can then reset the device to re-initialize the authentication process. The only detail that will be revealed to an unknown or hostile MP in this case would be the public key of the SEP. Such a provisioning mechanism allows incremental deployment along with inter-operation of SEPs and MPs from different manufactures.

However, this approach might not be suitable in a large scale industrial deployment where a large number of SEPs need to be provisioned and installed within a small time span. In such a scenario, the hash of the public key can be printed as a barcode and the owner can use a barcode scanner to feed all the public keys into the MP. Alternatively, at an additional cost, RIFD tags and scanners can be used to add the public keys to the MP.

It is important to note that both these provisioning schemes only aim to authenticate the SEPs to a Mirror Proxy. Authenticating the other way around is not required because, for the subset of the deployment space on which we focus, the communication between the SEP and the MP is mostly one-directional.

Chapter 5

Implementation

In this chapter we describe the implementation of the proof-of-concept prototype developed for the architecture presented in Chapter 4. In our prototype, a Sleeping End Point (SEP) was implemented using the Arduino Ethernet shield[1] over an Arduino Mega board as shown in Figure 5.1. Our implementation uses the standard C99 programming language on the Arduino Mega board without any operating system.

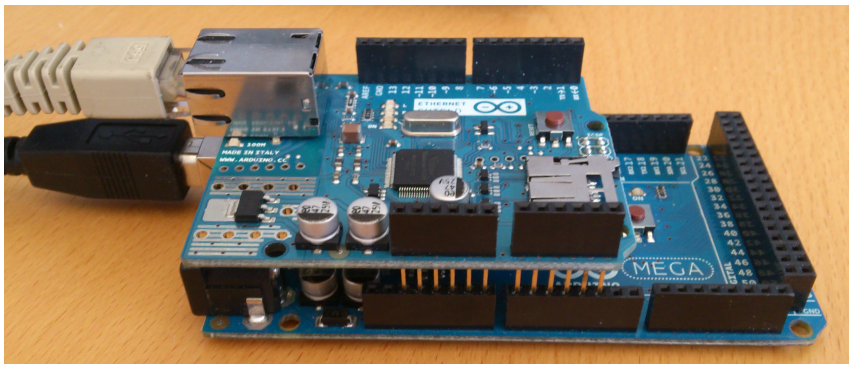


Figure 5.1: Arduino SEP

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 MP.

We chose the Relic library [13] 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 ATmega2560 processor of the Arduino Mega board as well as on the 64-bit processor of the client. The public-private key pair was

[1] Arduino Ethernet Shield, http://arduino.cc/en/Main/ArduinoEthernetShield

generated on the Arduino board at runtime using the default Arduino pseudo-random number generator. We used the Elliptic Curve Digital Signature Algorithm (ECDSA) to sign data updates at the SEP and verify them at the client. The standard NIST-K163 curve parameters (163-bit Kobltiz curve over binary field) were used for the ECDSA algorithm. While compiling the Relic library for our prototype, we used the fast configuration detailed in Appendix A without any assembly optimizations. The location of the MP was pre-configured into the SEP by hardcoding the IP address. We used an IPv4 network for communication over public IP addresses obtained from a DHCP server running in the network.

We used the Ericsson Gateway [73] running on a x86 linux machine to model the Mirror Proxy and the Resource Directory. The gateway implements the CoAP base specification [132] in the Java programming language and extends it to add the support for Mirror Proxy and Resource Directory REST interfaces. We developed a minimalistic CoAP C-library for the Arduino SEP and for the client requesting data updates for a resource. This library was implemented as a team effort at Ericsson Research. The library has small SRAM requirements and uses stack-based allocation only. It is inter-operable with the Java implementation of the CoAP protocol running on the gateway. The C-library was modified by the current author to port it to the Arduino Mega board modeling the SEP. The current author also modified the library to add ECDSA signature generation and signature verification functionality to this library. In the following sections we discuss how a SEP in our prototype registers and updates content with the MP. We also discuss how a client in the prototype can retrieve signed data updates.

5.1 Caching Data Updates

The SEP registers with the MP by sending a Confirmable CoAP POST message as shown in figure 5.2. This registration message includes a temperature resource in the CoRE link format along with the public key of the SEP in the JWK [80] 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 goes into the energy saving sleep mode. If however, for some reason the registration message or the ACK is lost or the entry cannot created by the MP, the SEP would re-

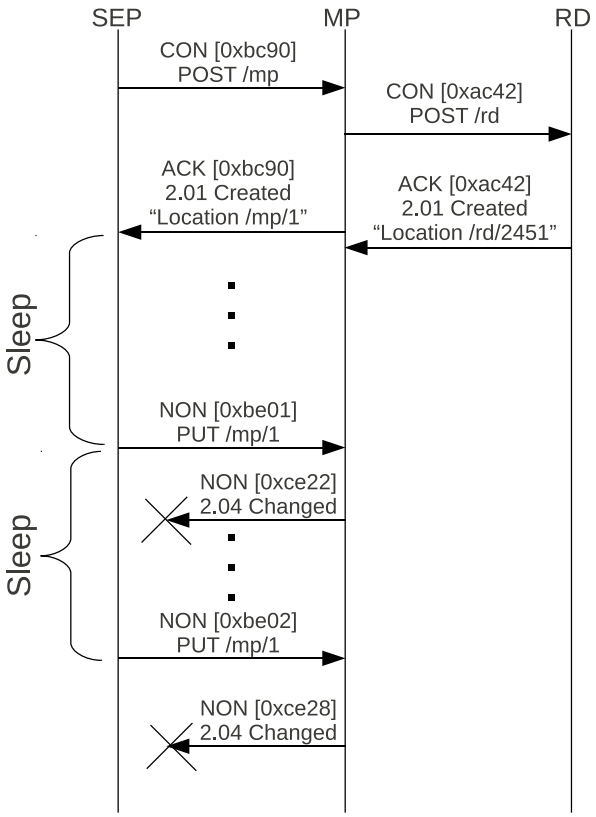


Figure 5.2: Registering and Caching Updates

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 5.2.

The SEP now wakes up at pre-determined intervals to update the cached data. In our prototype, we use hardcoded temperature values in the SenML format as sample data. This SenML data is signed with ECDSA algorithm and sent to the MP in the JWS [81] format with Non-Confirmable CoAP PUT messages as shown in 5.2. This allows the SEPs to return to the sleep mode without having to wait for any acknowledgements. However, even when Non-Confimable CoAP PUTs are used, the MP still sends a success or failure message as shown in figure 5.2. The SEP in this case remains in the sleep mode and is unaware of any packets sent to it. Nonetheless, it may be beneficial to have a CoAP option, as suggested by Vial [143], for requesting the sender to suppress any response and avoid creation of unnecessary network traffic.

5.2 Retrieving Data Updates

A client that wishes to obtain data updates from the SEP first contacts the RD as shown in Figure 5.3. We assume that the location of the RD is known to the client through DHCP [45] 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. Thus the Mirror Proxy serves data to requesting clients in a transparent manner.

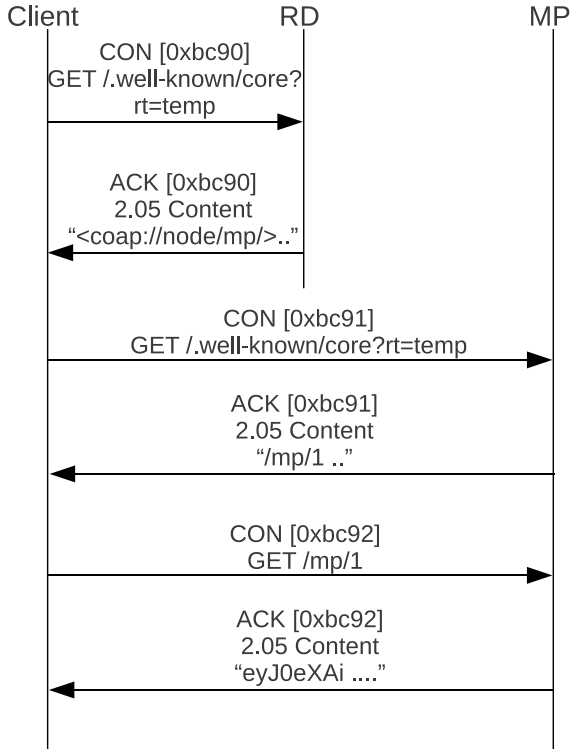


Figure 5.3: Retrieving Updates

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 5.3 to obtain the actual signed content in the

JWS [81] 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 [126] or by using HTTP [55] over SSL [60].

Hartke [132] discusses an *observe* mode for CoAP clients. This mode allows CoAP clients to use the proposed CoAP *observe* option and register their interest in a smart object for its current measured value. The smart objects are responsible for maintaining a list of interested client observers and pushing updates to the client whenever its state changes and a new data update is available. However, in this communication model, the smart objects still need to be online for extended durations to receive and register interests. This prevents them from sleeping for long durations to save energy. In our communication model, as discussed by Vial [143], this difficulty can be overcome by registering an *observe* interest at the Mirror Proxy. The MP maintains a list of interested CoAP client and updates them whenever it receives a new update from the SEP as shown in figure 5.4. The MP sends

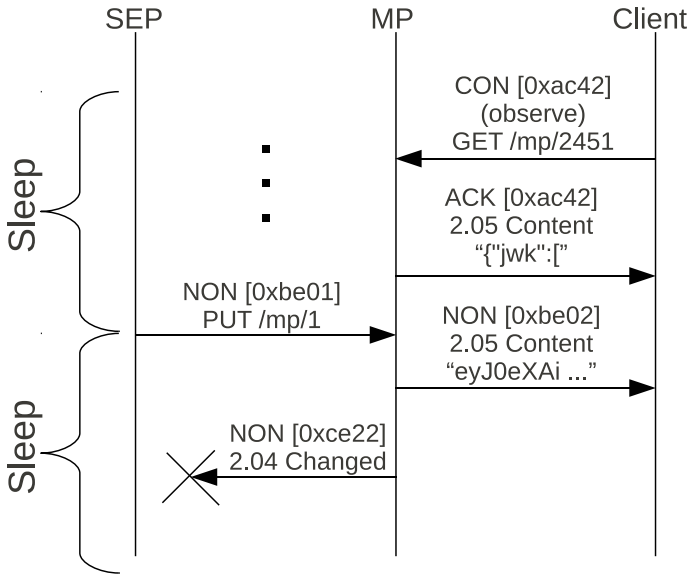


Figure 5.4: MP Updating Interested Clients

the public key piggybacked in response to an interest registration message from the client. It then pushes the signed data updates to the client as and when it receives them from the SEP. The client can verify the integrity and authenticity of these subsequent updates using the public key received at the time of registration.

5.3 Summary

In our proof-of-concept prototype we used an Arduino board to model a Sleeping End Point that sent signed temperature updates in the SenML format. The SenML data updates were signed using the ECDSA signature algorithm available in the Relic library. In this prototype implementation, the Arduino board did not transition into the energy-saving sleep mode after sending a data update. This transition to the energy-saving sleep mode is platform-specific and can be implemented according to the resource-constrained device being used. We used only one Arduino board acting as a single Sleeping End Point. The Arduino SEP used the minimalistic CoAP C-library developed for communication with the gateway hosting the RD and the MP. This library included only the basic functionality to send Confirmable and Non-Confirmable CoAP POST and PUT messages as well as receive CoAP ACK messages. Only the required subset of CoAP options such as *LocationPath*, *UriHost* and *ContentType* were included in this library and some error conditions were not handled. The prototype used sequence numbers in the signed updates for preventing replay attacks. A client requesting data updates for a temperature resource was modeled on an x86 linux machine. The current author was entirely responsible for implementing the client using the same CoAP C-implementation used on the SEP. Although, this prototype implementation used the Ethernet MAC-layer protocol, other protocols such Zigbee [6] and Bluetooth low energy [5] would work in a similar fashion without requiring any changes to the architecture.

Chapter 6

Discussion

We begin this chapter by presenting a post-implementation analysis of the communication model and the prototype. We then briefly evaluate the methodological principles used in thesis. Finally, we discuss the security considerations associated with the architecture presented in this thesis.

6.1 Architecture Overview

There are several fragmented security solutions for defending smart object networks. While some of the defense mechanisms focus on provisioning and secure authentication schemes [25, 89], others focus on data confidentiality [32, 88] only. Similarly, as shown in Section 3.1, several authors [24, 72, 138] demonstrate the applicability of using public-key cryptography on resource constrained devices, but none of them successfully build an architecture around it for secure communication.

In this thesis, we have developed a secure and energy-efficient communication model based on asymmetric cryptography that considers the security challenges encountered by a smart object during its entire lifecycle. We also implemented a prototype of this communication-model to demonstrate its feasibility and support its standardization. The architecture is aimed at smart objects that sleep for long durations during their operational phase to save energy and form a large subset of the deployment space. It uses the concept of a Mirror Proxy [143] to cache and serve data updates received from these sleeping smart objects and extends this concept to add security components. The entire communication is based on the Constrained Application Protocol (CoAP) [132] communication protocol over an IP network.

The architecture uses the standard resource representation and resource discovery mechanisms. Measurements from smart objects are reported in the

standardized SenML [79] format while the public keys and signed data are communicated in the standard JSON based JWK [80] and JWS [81] formats, respectively. Since CoAP can easily be mapped to HTTP [55], and we use an IP network with standardized technologies, our architecture can seamlessly inter-operate with other devices on the Internet.

The provisioning methods suggested for the architecture in Section 4.5 allow incremental deployment where new smart objects can be added to network as and when required. These modular schemes also allow smart objects and mirror proxies from different manufactures to inter-operate smoothly.

6.2 Evaluation of Methodology

> *We reject kings, presidents and voting. We believe in rough consensus and running code*

is a famous quote by David Clark [71] describing the standardization process at IETF. To study the feasibility of the architecture proposed and provide strong arguments in support of its standardization, we developed a proof-of-concept implementation in this thesis. Although the results from the implementation were encouraging, a large scale simulation is also required to analyze its feasibility with a large number of distributed Mirror Proxies caching content from several thousand Sleeping End Points.

In this thesis, we use easily accessible libraries for implementing asymmetric cryptography on resource constrained devices within a short span of time. The performance analysis of these libraries was promising and strongly supported the argument that public-key cryptography can be implemented on these devices. However, a more focused implementation effort that includes all well-known theoretical optimizations and uses platform-specific assembly instructions can further improve the efficiency of these algorithms. The measurements of SRAM consumption and execution time were performed with a common seed to the random number generator. Nonetheless, in order to correctly benchmark the performance, a more thorough analysis is required. The energy consumption calculated for various curves only gives an estimate since it does not take into account the amount of SRAM consumed. Accuracy of these measurements can be improved by employing a method similar to the one used by Margi et al. [100].

Our prototype uses the Arduino Ethernet shield as the communication module. Although it serves the purpose for a proof-of-concept implementation, we need to analyze the feasibility of the architecture with other, more efficient and widely deployed smart object MAC-layer protocols such as Zigbee [6] and Bluetooth low energy [5]. Finally, in our performance evaluation,

we only give a rough estimate of the flash memory consumption for the different libraries. A more detailed evaluation is required to obtain accurate figures.

6.3 Security Considerations

In Chapter 1, we have discussed why ensuring security in smart object networks is critical. We also present some of common obstacles for developing a secure communication model in smart object networks such as limited battery energy and computational power. In Chapter 4, we have developed a secure communication model based on public-key cryptography and, in this section, we analyze the security aspects associated with this architecture.

While evaluating the performance of public-key cryptography and implementing the prototype, we use use the default pseudo-random number generator available on Arduino boards. However it is important to use strong cryptographic (pseudo) random number generators such as Fortuna [54] in real-world deployments. These cryptographic random number generators require high quality seeds. Eastlake et al. [47] discuss some of the common pitfalls encountered while generating randomness for seeding material. They recommend several hardware techniques that may be used as entropy sources and emphasize the importance of correctly implementing these random number generators with the following statement

> *The use of pseudo-random processes to generate secret quantities can result in pseudo-security*

The ECDSA signature algorithm not only requires random numbers during key generation, but also during each signature operation. Therefore it is critical to have sufficient entropy to protect the private key from inadvertent disclosure. A large scale take down of the Sony PlayStation 3 system [23] was possible because of an incorrectly implemented random number generator used for the ECDSA algorithm. Bernstein et al. [22] propose a signature scheme that only requires entropy during key generation and not during each signature operation. If implemented correctly, this scheme would provide the flexibility to generate the keys on a platform with easy access to entropy sources and then feed them into the smart object at the time of manufacture.

We have used the classical Dolev-Yao [43] intruder model as the threat model while designing our security architecture. In the Dolev-Yao model, a malicious entity can eavesdrop, intercept, modify and replay any message in the network. Our communication model ensures end-to-end data-object integrity to defend against modification of any data updates sent from the

Sleeping End Points (SEPs). The message freshness mechanisms discussed in Section 4.4 protect the architecture from replay attacks by an eavesdropper in the network. Since the communication between the Mirror Proxy and a client is protected by SSL [60] or DTLS [126], an attacker cannot obtain meaningful data from simple eavesdropping.

However, the Dolev-Yao model does not take into account the fact that smart object networks are also vulnerable to node (in our case a node is the SEP) capture attacks [115, 146]. In our architecture, a node capture does not disrupt the security of the communication between other SEPs and the MP as long as a routing path exists between the two.

Stajano and Anderson [136] introduce the concept of *sleep deprivation torture* attack, also referred to as *denial-of-sleep* attack in wireless sensor networks. In a denial-of-sleep attack, a malicious entity reduces the opportunity for a resource constrained device to enter into power-saving sleep mode by sending legitimate requests for processing. Martin et al. [101] further categorize denial-of-sleep into three categories

- *Service Request Attack*: where valid service requests are repeated with the intention of draining power.
- *Benign Service Attack*: where a power intensive operation is requested to drain the battery.
- *Malignant Attack*: where an adversary penetrates a constrained device and alters the existing programs to consume more power than needed.

However in our communication model, the SEPs do not serve to client requests directly and a Mirror Proxy (MP) is used to add a level of indirection. This protects the SEPs from any denial-of-sleep attacks. Moreover, this also ensures that only the SEPs need to be authenticated to the MP which is easier than authenticating the other way around.

It is important to note that our communication model uses raw public keys in an efficient manner and does not require any certificates or certification authorities. If leap-of-faith provisioning is acceptable, the communication model can work without any pre-configuration. However in this leap-of-faith provisioning, neither the MP nor the SEP securely authenticate the other end-point. As discussed by Pham and Aura [122], such a bi-directional leap-of-faith is always susceptible to man-in-middle-attacks.

The architecture allows the MPs to host multiple applications (for example, temperature and pressure measurements reported from different sets of smart objects) which do not have to interact or rely on each other. Finally, the MPs can have several administrative domains with different access policies depending on the SEP or the client it is serving.

6.4 Reflections

Internet of Things (IoT) is on the path to form an ever increasing part of our physical environment and it is envisioned that there would 50 billion connected devices by the year 2020 [8]. Liu [97] in his lecture at Harvard points out that the social influence of IoT will surpass that of the Internet. As discussed in Chapter 1, the importance of IoT is further asserted from the fact that IoT is included by the US National Intelligence Council in the list of six "Disruptive Civil Technologies" with potential impacts on US national power [7]. The economic importance of IoT can also be inferred from the study of Fleisch [58], who observed that the most important driving factor for hundreds of IoT applications was the need to reduce the real-world to virtual-world transactional costs. It is clearly evident then, that IoT is going to have a huge impact on future businesses and personal lives. This thesis aims to support the large-scale acceptance and deployment of IoT by solving critical security issues associated with a subset of the smart object deployment space.

Brignall [76] argues that Internet can be viewed as a new structure for social control and is similar to a Panopticon [18], a prison structure that allows people in authority to monitor *inmates* who are not aware of the fact that they are being monitored. Liu [97] warns of the danger that IoT may also turn into a Panopticon structure if the security and privacy challenges associated with it are not addressed. However, in this thesis, we only deal with the security vulnerabilities associated with a subset of the entire smart object deployment space and a separate research effort is required to look into how privacy-preserving architectures can co-exist with such security solutions.

This thesis did not deal with any issues that have direct environmental impacts. However, our security architecture contains an energy-efficient communication model and we believe that such energy efficient designs can increase the life-span of smart objects and thereby reduce the e-waste that might be created when these smart objects or their batteries are discarded. This would play an important role in reducing the potential harmful impacts to the environment especially since the number of such smart objects is increasing at a fast pace.

As such, it is important to understand the political, economic, social and cultural problems associated with the Internet of Things. Addressing these challenges along with innovative technical solutions would ensure sustainable development of IoT. Such a sustainable development can enhance the quality of our future lives in every sphere.

Chapter 7

Conclusion

The Internet of Things is fast evolving and there are several existing deployment examples such as monitoring of structural defects in bridges with pressure sensors [36] and tracking of shipments with positioning sensors [129]. With advances in communication technologies and embedded hardware, simple physical devices have transformed into smart objects that understand and react to their environment. These smart objects often communicate among themselves with no human interaction. Even though there has been recent success in their adoption, there are several challenges that hinder the wide-scale deployment and acceptance of smart objects. One such critical challenge is ensuring security in these networks. We have explained in Chapter 1 that ensuring security in smart object networks is non-trivial because of the energy and computational constraints of these devices. Previous research also demonstrated that considering the vulnerabilities in the entire lifecycle of a smart object is important when designing a new security solution.

In this thesis, we developed a secure and energy-efficient architecture for a subset of the smart object deployment space. We focused on smart objects that sleep for long durations in their operational phase to save energy. From the outset, we decided to use public-key cryptography because of its flexibility over symmetric key cryptography. Although there have been several past studies which demonstrated the feasibility of public-key cryptography on constrained devices, we wanted to find public libraries that could be easily ported for use on 8-bit platforms. We found four such libraries and evaluated their performance in terms of execution time, SRAM consumption and energy consumption. The results from the evaluation were encouraging and reinforced our belief that software implementation of public-key cryptography on constrained devices is not only possible but can also be efficient.

Our architecture used the concept of Mirror Proxy [143] to cache updates sent from the "sleepy" smart objects. This delegation mechanism allowed

the smart objects to behave as clients of the Mirror Proxy rather than as servers that respond to client requests directly. The smart objects no longer had to stay online for serving requests and were able to save energy by sleeping for long durations in their operational phase. The resources of a smart object were represented in the standard CoRE Link format [131]. A Resource Directory [133] was used to maintain the list of resources hosted on other distributed smart objects. The communication in this architecture was based on the standard Constrained Application Protocol (CoAP) [132] designed for M2M and smart object networks.

The smart objects in the architecture registered their public key with the Mirror Proxy either using a leap-of-faith mechanism or with the secure authentication schemes suggested in Section 4.5. Thereafter, the objects woke up at pre-determined intervals to send signed updates. This allowed the clients to receive updates even when the smart objects were in sleep mode. It also allowed them to verify the integrity and authenticity of these updates end-to-end over a multi-hop network topology. Messages were protected against replay attacks with the use of freshness schemes such as sequence numbers. In this architecture, the public key was communicated in the standard JWK [80] format while the signatures were communicated in the standard JWS [81] format.

To the author's best knowledge, this is one of the first energy-efficient standards-compliant security architectures to be developed. The implemented prototype not only provides an insight into the operation of existing standards but also demonstrates the feasibility of the communication model and provides strong arguments in support of its standardization. However, this thesis takes only the first steps in solving the security problems for a subset of the deployment space. Possible future work items include:

- Performance evaluation of the libraries and a prototype with strong random number generator.
- Investigate if side-channel attacks are possible in this communication model.
- Implement a large scale simulation of the architecture to study its performance with a large number of smart objects.
- Evaluate the architecture with other MAC layer protocols such as Zigbee [6] and Bluetooth low energy [5].
- Study other provisioning and freshness schemes that may be suitable for this architecture.

Bibliography

[1] MIT AUTO-ID labs. `http://autoid.mit.edu/cs/`. Accessed 02.04.2012.

[2] Recommended elliptic curves for federal government use. National Institute of Standards and Technology, July, 1999.

[3] Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 1997. IEEE Computer Society LAN MAN Standards Committee.

[4] SEC 2: Recommended elliptic curve domain parameters. Certicom Research, 2000.

[5] IEEE std 802.15. 1-2005 part 15.1: Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs). Wireless Personal Area Networks Working Group White Paper.

[6] ZigBee specification. 344–346. Zigbee Alliance Document 053474r13.

[7] Six technologies with potential impacts on US interests out to 2025. Conference report CR 2008-07, National Intelligence Council, April 2008.

[8] More than 50 billion connected devices - taking connected devices to mass market and profitability. Ericsson White Paper.

[9] Aboba, B., Levkowetz, H., Vollbrecht, J., Blunk, L., and Carlson, J. RFC 3748: Extensible authentication protocol (EAP), June 2004.

[10] Al-Kayali, A. Elliptic curve cryptography and smart cards. *SANS Institute 17* (2004).

[11] Aranha, D., Dahab, R., López, J., and Oliveira, L. Efficient implementation of elliptic curve cryptography in wireless sensors. *Advances in Mathematics of Communications 4*, 2 (2010), 169–187.

[12] Aranha, D., López, J., Oliveira, L., and Dahab, R. Efficient implementation of elliptic curves on sensor nodes.

[13] Aranha, D. F., and Gouvêa, C. P. L. RELIC is an efficient library for cryptography. `http://code.google.com/p/relic-toolkit/` Accessed 04.04.2012.

[14] Aura, T. RFC 3972: Cryptographically generated addresses (CGA), March 2005.

[15] Baar, M., Köppe, E., Liers, A., and Schiller, J. Poster abstract: The scatterweb MSB-430 platform for wireless sensor networks. In *Contiki Workshop* (2007).

[16] Bailey, D., and McGrew, D. AES-CCM cipher suites for TLS. Internet draft, IETF, October 2011.

[17] Baumgartner, T., Chatzigiannakis, I., Fekete, S., Koninis, C., Kröller, A., and Pyrgelis, A. Wiselib: A generic algorithm library for heterogeneous sensor networks. In *European Conference on Wireless Sensor Networks* (2010), Springer, pp. 162–177.

[18] Bentham, J. *Panopticon or the inspection house*, vol. 2. 1791.

[19] Bergmann, O. tinyDTLS - a basic DTLS server template. `http://tinydtls.sourceforge.net/`. Accessed 04.04.2012.

[20] Bergmann, O., Gerdes, S., and Bormann, C. Simple Keys for Simple Smart Objects. In *Smart Object Security Workshop, IETF 83* (2012).

[21] Berners-Lee, T., Masinter, L., and McCahill, M. RFC 1738: Uniform resource locators (URL), December 1994.

[22] Bernstein, D., Duif, N., Lange, T., Schwabe, P., and Yang, B. High-speed high-security signatures. In *Cryptographic Hardware and Embedded Systems (CHES)* (2011), Springer, pp. 124–142.

[23] Bernstein, D. J., Lange, T., and Schwabe, P. The security impact of a new cryptographic library. In *Industry Proceedings of the 10th International Conference on Applied Cryptography and Network Security (ACNS)* (2012 (to appear)).

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

[25] Bohge, M., and Trappe, W. An authentication framework for hierarchical ad hoc sensor networks. In *Proceedings of the 2nd ACM Workshop on Wireless security* (2003), pp. 79–87.

[26] Boneh, D., and Boyen, X. Short signatures without random oracles. *Advances in Cryptology (EUROCRYPT)* (2004), 56–73.

[27] Boneh, D., et al. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS 46*, 2 (1999), 203–213.

[28] Boneh, D., Lynn, B., and Shacham, H. Short signatures from the weil pairing. *Journal of Cryptology 17*, 4 (2004), 297–319.

[29] Bos, J., Kaihara, M., Kleinjung, T., Lenstra, A., and Montgomery, P. Solving a 112–bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction. *International Journal of Applied Cryptography 2*, 3 (2012), 212–228.

[30] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and Yergeau, F. Extensible markup language (XML). *World Wide Web Journal 2*, 4 (1997), 27–66.

[31] Calvet, J., and Noll, J. Subscriber identity module, 2005. US Patent App. 10/594,559.

[32] Çam, H., Özdemir, S., Nair, P., Muthuavinashiappan, D., and Ozgur Sanli, H. Energy-efficient secure pattern based data aggregation for wireless sensor networks. *Computer Communications 29*, 4 (2006), 446–455.

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

[34] Chan, H., Perrig, A., and Song, D. Random key predistribution schemes for sensor networks. In *Symposium on Security and Privacy* (2003), IEEE, pp. 197–213.

[35] Cheneau, T., Sambra, A., and Laurent, M. A trustful authentication and key exchange scheme (TAKES) for ad hoc networks. In *5th International Conference on Network and System Security (NSS)* (2011), IEEE, pp. 249–253.

[36] Cho, S., Spencer Jr, B., Jo, H., Li, J., and Kim, R. Sensing & measurement bridge monitoring using wireless smart sensors. *International society for optics and photonics*. `http://spie.org/x84931.xml?highlight=x2406&ArticleID=x84931`. Accessed 4.4.2012.

[37] Cohen, H., Miyaji, A., and Ono, T. Efficient elliptic curve exponentiation using mixed coordinates. *Advances in Cryptology (ASIACRYPT)* (1998), 51–65.

[38] Colin, O. Initial configuration of resource-constrained devices. Internet draft, IETF, January 2010.

[39] Crockford, D. RFC 4627: The application/json media type for javascript object notation (JSON), July 2006.

[40] Denis, T. Libtomcrypt. `http://libtom.org`. Accessed 04.04.2012.

[41] Dierks, T., and Rescorla, E. RFC 5246: The transport layer security (TLS) protocol version 1.2, August 2008.

[42] Ding, C., Pei, D., and Salomaa, A. *Chinese remainder theorem*. World Scientific Singapore, 1996.

[43] Dolev, D., and Yao, A. On the security of public key protocols. *IEEE Transactions on Information Theory 29*, 2 (1983), 198–208.

[44] Douceur, J. The sybil attack. In *Peer-to-peer Systems* (2002), Springer, pp. 251–260.

[45] Droms, R. RFC 1531: Dynamic host configuration protocol, October 1993.

[46] Du, W., Deng, J., Han, Y., Varshney, P., Katz, J., and Khalili, A. A pairwise key predistribution scheme for wireless sensor networks. *ACM Transactions on Information and System Security (TISSEC) 8*, 2 (2005), 228–258.

[47] Eastlake, D., Crocker, S., and Schiller, J. RFC 1750: Randomness recommendations for security, March 2005.

[48] Elkins, M. RFC 2015: MIME security with pretty good privacy (PGP), October 1996.

[49] Elson, J. *Time synchronization in wireless sensor networks*. PhD thesis, University of California, Los Angeles, 2003.

[50] Eriksson, J., Österlind, F., Finne, N., Tsiftes, N., Dunkels, A., Voigt, T., Sauter, R., and Marrón, P. COOJA/MSPSim: Interoperability testing for wireless sensor networks. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques* (2009), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

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

[52] Fabri, A., and Pion, S. CGAL: the computational geometry algorithms library. In *Proceedings of the 17th International Conference on Advances in Geographic Information Systems (SIGSPATIAL)* (2009), ACM, pp. 538–539.

[53] Farrell, S., Kutscher, D.and Dannewitz, C., Ohlman, B., Keranen, A., and Hallam-Baker, P. Naming things with hashes. Internet draft, IETF, April 2012.

[54] Ferguson, N., and Schneier, B. *Practical cryptography*, vol. 141. Wiley New York, 2003.

[55] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. RFC 2616: Hypertext transfer protocol – HTTP/1.1, June 1999.

[56] Fielding, R., and Taylor, R. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT) 2*, 2 (2002), 115–150.

[57] FIPS-186. Digital signature standard (DSS). *National Institute of Standards and Technology, US Department of Commerce* (1993).

[58] Fleisch, E. What is the internet of things? *When Things Add Value. Auto-ID Labs White Paper WP-BIZAPP-053, Auto-ID Lab St. Gallen, Switzerland* (2010).

[59] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and Yegin, A. RFC 5191: Protocol for carrying authentication for network access (PANA), March 2006.

[60] Freier, A., Karlton, P., and Kocher, P. The SSL protocol version 3.0. Internet draft, IETF, November 1996.

[61] Gallant, R., Lambert, R., and Vanstone, S. Faster point multiplication on elliptic curves with efficient endomorphisms. *Advances in Cryptology (CRYPTO)* (2001), 190–200.

[62] Ganeriwal, S., Kumar, R., Adlakha, S., and Srivastava, M. Network-wide time synchronization in sensor networks. Tech. rep., University of California, Dept. of Electrical Engineering, 2002.

[63] Garcia-Morchon, O., Kumar, S., Struik, R., Keoh, S., and Hummen, R. Security considerations in the IP-based internet of things. Internet draft, IETF, March 2012.

[64] Gay, D., Levis, P., Von Behren, R., Welsh, M., Brewer, E., and Culler, D. The nesC language: A holistic approach to networked embedded systems. *ACM Sigplan Notices 38*, 5 (2003), 1–11.

[65] Gehrmann, C., Mitchell, C., and Nyberg, K. Manual authentication for wireless devices. *RSA Cryptobytes 7*, 1 (2004), 29–37.

[66] Gollakota, S., Ahmed, N., Zeldovich, N., and Katabi, D. Secure in-band wireless pairing. In *USENIX Security Symposium* (2011).

[67] Goodrich, M., Sirivianos, M., Solis, J., Tsudik, G., and Uzun, E. Loud and clear: Human-verifiable authentication based on audio. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS)* (2006).

[68] Gupta, V., Wurm, M., Zhu, Y., Millard, M., Fung, S., Gura, N., Eberle, H., and Chang Shantz, S. Sizzle: A standards-based end-to-end security architecture for the embedded internet. *Pervasive and Mobile Computing 1*, 4 (2005), 425–445.

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

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

[71] Harris, S. RFC 3160: The tao of IETF. a novice's guide to the internet engineering task force, August 2001.

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

[73] Höller, J. 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`.

[74] Hu, Y., Perrig, A., and Johnson, D. Wormhole attacks in wireless networks. *IEEE Journal on Selected Areas in Communications 24*, 2 (2006), 370–380.

[75] Huang, C. LOFT: Low-overhead freshness transmission in sensor networks. In *IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing (SUTC)* (2008), pp. 241–248.

[76] III, T. B. The new panopticon: The Internet viewed as a structure of social control, 2002. `http://theoryandscience.icaap.org/content/vol003.001/brignall.html`.

[77] Jansen, W., Gavrila, S., and Korolev, V. Proximity-based authentication for mobile devices. *Security and Management* (2005), 398–404.

[78] Jara, A., Zamora, M., and Skarmeta, A. An architecture based on internet of things to support mobility and security in medical environments. In *7th IEEE Consumer Communications and Networking Conference (CCNC)* (2010), pp. 1–5.

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

[80] Jones, M. JSON web key (JWK). Internet draft, IETF, December 2011.

[81] Jones, M., Bradley, J., and Sakimura, N. JSON web signature (JWS). Internet draft, IETF, December 2011.

[82] Jonsson, J., and Kaliski, B. RFC 3447: Public-key cryptography standards (PKCS) #1: RSA cryptography specifications version 2.1, February 2003.

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

[84] Karlof, C., Sastry, N., and Wagner, D. TinySec: A link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems* (2004), pp. 162–175.

[85] Karlof, C., and Wagner, D. Secure routing in wireless sensor networks: Attacks and countermeasures. *Ad hoc networks 1*, 2-3 (2003), 293–315.

[86] Kaufman, C. RFC 4306: Internet key exchange (IKEv2) protocol, December 2005.

[87] Kent, S., and Seo, K. RFC 4301: Security architecture for the internet protocol, December 2005.

[88] Kim, D., Shazzad, K., and Park, J. A framework of survivability model for wireless sensor network. In *The First International Conference on Availability, Reliability and Security (ARES)* (2006), IEEE, p. 8.

[89] Kivinen, T. Minimal IKEv2. Internet draft, IETF, February 2011.

[90] Koblitz, N. Elliptic curve cryptosystems. *Mathematics of computation 48*, 177 (1987), 203–209.

[91] Koblitz, N. CM-curves with good cryptographic properties. *Advances in Cryptology (CRYPTO)* (1992), 279–287.

[92] Kuptsov, D., Nechaev, B., and Gurtov, A. Securing medical sensor network with HIP. In *2nd International ICST Conference on Wireless Mobile Communication and Healthcare (MobiHealth)* (2011).

[93] Lehmer, D. On euler's totient function. *Bulletin of American Mathematical Society 38* (1932), 745–757.

[94] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., et al. TinyOS: An operating system for sensor networks. *Ambient intelligence 35* (2005).

[95] Liu, A., and Ning, P. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In *International Conference on Information Processing in Sensor Networks (IPSN)* (2008), IEEE, pp. 245–256.

[96] Liu, D., and Ning, P. Location-based pairwise key establishments for static sensor networks. In *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks* (2003), ACM, pp. 72–82.

[97] Liu, Y. Preliminary exploration on social impacts of internet of things (iot) and countermeasures, 2011. Lecture.

[98] Madden, S., Franklin, M., Hellerstein, J., and Hong, W. TAG: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review 36*, SI (2002), 131–146.

[99] Madden, S., Szewczyk, R., Franklin, M., and Culler, D. Supporting aggregate queries over ad-hoc wireless sensor networks. In *Mobile Computing Systems and Applications* (2002), IEEE, pp. 49–58.

[100] Margi, C., Petkov, V., Obraczka, K., and Manduchi, R. Characterizing energy consumption in a visual sensor network testbed. In *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM)* (2006), IEEE.

[101] Martin, T., Hsiao, M., Ha, D., and Krishnaswami, J. Denial-of-service attacks on battery-powered mobile computers. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications* (2004), IEEE, pp. 309–318.

[102] Masinter, L., Berners-Lee, T., and Fielding, R. RFC 3896: Uniform resource identifier (URI): Generic syntax, January 2005.

[103] Masinter, L., and Sollins, K. RFC 1737: Functional requirements for uniform resource names, December 1994.

[104] McCurley, K. The discrete logarithm problem. *Proceedings of Symposium in Applied Math 42* (1990), 49–74.

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

[106] Miller, V. Use of elliptic curves in cryptography. *Advances in Cryptology (CRYPTO)* (1985), 417–426.

[107] Mills, D. RFC 1305: Network time protocol (NTP) version 3, March 1992.

[108] Mockapetris, P. RFC 882: Domain names: Concepts and facilities, November 1983.

[109] Montgomery, P. Modular multiplication without trial division. *Mathematics of computation 44*, 170 (1985), 519–521.

[110] Morain, F., and Olivos, J. Speeding up the computations on an elliptic curve using addition-subtraction chains. In *Theoretical Informatics and Applications* (1990).

[111] Moskowitz, R. HIP diet exchange (DEX). Internet draft, IETF, March 2011.

[112] Moskowitz, R., Nikander, P., Jokela, P., and Henderson, T. RFC 5201: Security architecture for the internet protocol, February 2004.

[113] Mouly, M., Pautet, M., and Foreword By-Haug, T. *The GSM system for mobile communications*. Telecom Publishing, 1992.

[114] Mulligan, G. The 6LoWPAN architecture. In *Proceedings of the 4th workshop on Embedded networked sensors* (2007), ACM, pp. 78–82.

[115] Mykletun, E., Girao, J., and Westhoff, D. Public key based cryptoschemes for data concealment in wireless sensor networks. In *IEEE International Conference on Communications (ICC)* (2006), vol. 5, pp. 2288–2295.

[116] Ngai, E., Liu, J., and Lyu, M. On the intruder detection for sinkhole attack in wireless sensor networks. In *IEEE International Conference on Communications (ICC)* (2006), vol. 8, IEEE, pp. 3383–3389.

[117] Nottingham, M. RFC 5998: Web linking, October 2010.

[118] Park, S., Kim, K., Haddad, W., Chakrabarti, S., and Laganier, J. IPv6 over low power WPAN security analysis. Internet draft, IETF, March 2006.

[119] Park, T., and Shin, K. LiSP: A lightweight security protocol for wireless sensor networks. *ACM Transactions on Embedded Computing Systems (TECS) 3*, 3 (2004), 634–660.

[120] Perrig, A., Stankovic, J., and Wagner, D. Security in wireless sensor networks. *Communications of the ACM 47*, 6 (2004), 53–57.

[121] Perrig, A., Szewczyk, R., Tygar, J., Wen, V., and Culler, D. SPINS: Security protocols for sensor networks. *Wireless networks 8*, 5 (2002), 521–534.

[122] Pham, V., and Aura, T. Security analysis of leap-of-faith protocols. In *7th International Conference on Security and Privacy in Communication Networks (SecureComm)* (2011), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[123] Postel, J. RFC 768: User datagram protocol, August 1980.

[124] Postel, J. RFC 791: Internet protocol, September 1981.

[125] Rasmussen, K., Castelluccia, C., Heydt-Benjamin, T., and Capkun, S. Proximity-based access control for implantable medical devices. In *Proceedings of the 16th ACM conference on Computer and communications security* (2009), pp. 410–419.

[126] Rescorla, E. and Modadugu, N. RFC 4347: Datagram transport layer security (DTLS), April 2006.

[127] Saxena, N., Ekberg, J., Kostiainen, K., and Asokan, N. Secure device pairing based on a visual channel. In *IEEE Symposium on Security and Privacy* (2006).

[128] Schneider, J., and Kamiya, T. Efficient XML interchange (EXI) format 1.0. *W3C Working Draft 19* (2008).

[129] Schoeneman, J., and Sorokowski, D. Authenticated tracking and monitoring system (ATMS) tracking shipments from an australian uranium mine. In *Proceedings of The 31st IEEE Annual International Carnahan Conference on Security Technology* (1997), pp. 231–240.

[130] Sethi, M., Arkko, J., Keranen, A., and Rissanen, H. Practical considerations and implementation experiences in securing smart object networks. Internet draft, IETF, March 2012.

[131] Shelby, Z. CoRE link format. Internet draft, IETF, January 2012.

[132] Shelby, Z., Hartke, K., Bormann, C., and Sturek, D. Constrained application protocol (CoAP). Internet draft, IETF, March 2012.

[133] Shelby, Z., and S., K. CoRE resource directory. Internet draft, IETF, November 2011.

[134] Solinas, J. Efficient arithmetic on koblitz curves. *Designs, Codes and Cryptography 19*, 2 (2000), 195–249.

[135] Spilker, J. *The Global Positioning System: theory and applications*, vol. 2. AIAA, 1996.

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

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

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

[139] Urien, P., Nyami, D., Elrharbi, S., Chabanne, H., Icart, T., Pépin, C., Bouet, M., Cunha, D., Guyot, V., Pujolle, G., et al. HIP tags privacy architecture. In *3rd International Conference on Systems and Networks Communications (ICSNC)* (2008), IEEE, pp. 179–184.

[140] van der Laan, E. AVRCryptoLib. `http://www.emsign.nl/`. Accessed 04.04.2012.

[141] Van Greunen, J., and Rabaey, J. Lightweight time synchronization for sensor networks. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications* (2003), pp. 11–19.

[142] Vanstone, S. Responses to NIST's proposal. *Communications of the ACM 35* (1992), 50–52. (Communicated by John Anderson).

[143] Vial, M. CoRE mirror proxy. Internet draft, IETF, November 2011.

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

[145] Weis, S., Sarma, S., Rivest, R., and Engels, D. Security and privacy aspects of low-cost radio frequency identification systems. *Security in pervasive computing* (2004), 50–59.

[146] Westhoff, D., Girao, J., and Acharya, M. Concealed data aggregation for reverse multicast traffic in sensor networks: Encryption, key distribution, and routing adaptation. *IEEE Transactions on Mobile Computing 5*, 10 (2006), 1417–1431.

[147] Wheat, D. *Arduino Internals*. Apress, 2011.

[148] Yergeau, F. RFC 3629: UTF-8, a transformation format of ISO 10646, November 2003.

[149] Ylonen, T., and Lonvick, C. RFC 4251: The secure shell (SSH) protocol architecture, January 2006.

[150] Yuan, L., and Qu, G. Design space exploration for energy-efficient secure sensor network. In *Proceedings. The IEEE International Conference on Application-Specific Systems, Architectures and Processors* (2002), IEEE, pp. 88–97.

Appendix A

Relic Configurations

Relic uses Cross platform make (CMake)[1] for selecting the components and building a binary for a particular platform. The two configurations used while experimenting with the library are as follows:

Relic Fast Configuration	**Relic Low-memory Configuration**
Multi-precision Arithmetic methods	
Comba Multiplication	Comba Multiplication
Comba Squaring	Comba Squaring
Montgomery Modular Reduction	Montgomery Modular Reduction
Sliding window modular exponentiation	Sliding window modular exponentiation
Stein's binary GCD algorithm	Stein's binary GCD algorithm
Binary Field Arithmetic methods	
Integrated modular multiplication	Integrated modular multiplication
Integrated modular squaring	Integrated modular squaring
Fast polynomial reduction with a trinomial or pentanomial	Fast polynomial reduction with a trinomial or pentanomial
Square root by repeated squaring	Square root by repeated squaring
Trace computation by repeated squaring	Trace computation by repeated squaring
Solving a quadratic equation by half-trace computation	Solving a quadratic equation by half-trace computation
Inversion by the Extended Euclidean algorithm	Inversion by the Extended Euclidean algorithm

[1]Cross Platform Make, http://www.cmake.org/

Binary exponentiation	Binary exponentiation
Iterative squaring/square-root computation by consecutive squaring/square-root	Iterative squaring/square-root computation by consecutive squaring/square-root
Binary Elliptic Curve methods	
Projective Coordinates	Projective Coordinates
Right-to-left window (T)NAF method for point multiplication	Binary method for point multiplication
Left-to-right window (T)NAF method for fixed point multiplication	Left-to-right window (T)NAF method for fixed point mulitplication
Interleaving of window (T)NAFs for simultaneous multiplication and addition	Simple simultaneous multiplication and addition

Table A.1: Relic Library Configurations

Appendix B

IETF 83 and Workshop on Smart Object Security

The preliminary results from this thesis including the performance analysis of public-key cryptography and implementation experiences from the proof-of-concept prototype were presented at the Smart Object Security Workshop[1] preceding the IETF 83 meeting held in Paris. The results were documented in an Internet Draft [130] and were also presented at the Light Weight Implementation Guidance (LWIG) and the Security Area Advisory Group (SAAG) working groups of the IETF.

The overall feedback from the participants of the workshop and the working groups was positive and encouraging. Some important observations that were made during several discussions with the participants include:

- A consensus existed among the participants on the fact that software implementation of public-key cryptography on resource-constrained devices is possible.
- Existing cryptographic algorithms along with associated mathematical optimizations can provide acceptable performance. New algorithms specifically designed for smart object networks may not be needed. Besides, using existing standard algorithms also ensures easy interoperability with other entities on the Internet.
- At the time of this presentation, the energy consumption of the libraries had not been evaluated. It was pointed out that energy consumption is an important parameter to report even if it only provides an estimate.

[1] Workshop on Smart Object Security,
http://www.lix.polytechnique.fr/hipercom/SmartObjectSecurity/

- A hypothesis was put forth by one of the participants. It suggested that with the currently available hardware, performing computation on a smart object is more energy-efficient than transmitting data over a wireless interface. This hypothesis was broadly accepted by the participants, but it was also decided that it requires further investigation for conclusive evidence.

- This hypothesis supports our architecture which relies on end-to-end data-object integrity as it creates less network traffic compared to some of the other security alternatives such as CoAP over DTLS.

- Ensuring end-to-end data-object integrity is critical with intermediaries (such as the Mirror Proxy) in multi-hop network topologies. Our prototype in this direction was greatly appreciated. It was also accepted that this communication model fits well for the subset of the deployment space on which we focus.

- Although signature verification on resource-constrained devices was not required for our architecture, other participants were interested in a performance evaluation of signature verification on smart objects. Signature verification may be required when, for example, software and firmware updates received over the network need to be verified.

- Minimization of the number of intermediaries is important. Too many intermediaries make the communication model complex and increase the overall vulnerability of the system.

- Several Provisioning schemes for smart object networks were discussed. There were no conclusive results and it was decided that secure provisioning requires further research.

- Another challenging problem that came up during the discussions is the transfer of ownership in smart object networks. Smart objects that were previously provisioned with a proxy or a gateway may need to be transferred to a new owner. This can be non-trivial where there are several provisioned smart objects that are located in physically inaccessible areas.

It was also a great learning experience to attend the workshop and the IETF meeting. It provided an insight to the standardization process and the working of IETF. Meeting some of the experts from the industry and academia working in this area and listening to their feedback was enlightening.