Authorization Framework for the Internet-of-Things

Ludwig Seitz*, Göran Selander†, Christian Gehrmann*
*Security Lab, SICS Swedish ICT, Sweden
†Security Research, Ericsson Research, Sweden
{ludwig,chrisg}@sics.se, goran.selander@ericsson.com

***Abstract*—This paper describes a framework that allows fine-grained and flexible access control to connected devices with very limited processing power and memory.**

We propose a set of security and performance requirements for this setting and derive an authorization framework distributing processing costs between constrained devices and less constrained back-end servers while keeping message exchanges with the constrained devices at a minimum.

As a proof of concept we present performance results from a prototype implementing the device part of the framework.

Keywords*—*Internet of Things, Access control, Assertions

I. Introduction

Internet-of-Things (IoT) is a term commonly used today to describe a networked society, where everything that can benefit from a connection will be connected. This means that in contrast to the past where mainly mobile phones and computers were globally interconnected over the Internet, now all kinds of electronic equipment are about to come on-line. This trend is expected to accelerate over the coming years due to the decrease of hardware and network costs as well as the Internet technology maturity.

In this paper we address one of the important security challenges, *authorization and access control*, in the context of interconnected systems consisting of resource constrained devices not directly operated by humans. In particular, we focus on the problem where a single constrained device is communicating with *several* other devices or back-end computers.

This implies, that even if the device is primarily configured by one person or organization, it must be able to handle connections from other entities and these different entities may *not* have the *same access rights*, i.e., the device must be able to distinguish between requests from different entities, and enforce fine-grained authorization decisions.

Furthermore, we allow authorization decisions relating to a device to be based on local data only available to the device itself. The use of device local conditions for policy decision adds significant flexibility to the access control models that can be supported, since any parameter read by the device can be used to condition granting of a request.

The scope of this paper is to support *generic* authorization and access control procedures applicable to a variety of devices and access purposes. However, since the constrained devices are the limiting factor we must assume the worst case and design for a restricted computation and communication budget (transmission and reception are known to be costly for wireless sensor devices).

978-1-4673-5828-6/13/$31.00

Our contributions in this paper are a generic authorization framework supporting fine-grained and flexible access control to constrained devices, with the following properties:

- the device enforces the access control decision locally;
- the decision may be based on device local parameters;
- the framework is based on current Internet and access control standards;
- no extra messages are exchanged with the device compared to current deployments without access control;
- the execution times on a constrained device are reasonable.

The challenge in this work is to adapt standardized approaches from other domains to the resource constraints imposed by the Internet-of-Things settings.

The rest of the paper is organized as follows. In section II, we discuss related work in this area and provide the background information needed to understand the framework. We specify our requirements for an IoT authorization framework in section III. Section IV describes the overall framework in relation to the requirements, and section V discusses candidate authentication and key establishment protocols. The necessary authorization procedures are specified in section VI and the core entity of the framework, the Authorization Engine, is described in section VII. We discuss implementation results in section VIII and evaluate the security of the framework in section IX. Finally, section X provides a summary of the paper and examines the potential future work in this area.

II. Related work

The EU-project *Internet-of-Things Architecture* is working on a general framework for IoT including security. As part of this work the consortium has released a concept description for privacy and security for IoT services [1]. This concept includes usage of the *eXtensible Access Control Markup Language (XACML)* [2] and the *Security Assertion Markup Language (SAML)* [3] for IoT but does not discuss adaptations or changes to these standards which would be necessary in order for them to work efficiently in constrained environments.

Naedele [4] proposes a public-key based protocol for secure access and communication using a back-end authorization engine and signed capabilities. The device, after verifying the validity of the capability, initiates a protected session with a user which is used for further communication. However, the protocol requires several message exchanges in order to establish a secure session, lacks the option of device local conditions, and is not adapted to relevant standards.

Resch et al. surveys lightweight methods to secure geo-infrastructures with low power units in [5]. This scenario is very similar to that we consider and the results apply also to our scenario. The authors also suggest a novel security framework. In contrast to our suggested framework, this is built upon a security proxy function in the network. This breaks the end-to-end security for the service which is one of our basic requirements (see Section III).

Zhang et al. [6] describe a distributed privacy-preserving access control scheme for sensor networks. Their protocol requires users to purchase tokens from the device owner in order to access data from the device. The main focus of this work lies in protecting the privacy of the user towards the device and preventing token-reuse. It does not examine fine-grained access control enforcement on the devices, as well as local conditions.

In summary one can say that problem of authorization in the Internet of Things is not conclusively solved yet. Current approaches concentrate on a number of very specific problems whereas our approach is to design a generic authorization framework based on established and emerging standards. The novelty of our approach is that we have designed profiles and adaptations of these standards to enable or optimize their use with constrained devices.

III. Authorization Framework Requirements

As previously described, we target a setting of constrained devices and different entities accessing those devices. For this setting we list a set of security and performance requirements of a generic authorization framework. Naedele [4] also lists requirements on a similar basis, so we limit ourselves to additional requirements supporting fine-grained and flexible authorization:

- **R1:** The framework shall support
  - **a** differentiated access rules for different requesting clients,
  - **b** access control at the granularity of RESTful resources,
  - **c** policies based on local conditions (e.g. state of device, time, position).
- **R2:** All introduced security mechanisms shall be designed such that the total overhead due to computation and especially communication is as low as possible on the *device side*.
- **R3:** The authorization framework shall support secure access to access control related information.
- **R4:** The access control solution shall be dependent on a minimum of other functions.
- **R5:** The solution shall provide end-to-end protection (integrity and confidentiality) of relevant parts of the protocol messages, as well as replay protection.

Note that in order to be generally applicable, the authorization framework we are addressing must not be dependent on a specific authentication mechanism, key management or secure transport protocol. These security mechanisms must also comply with similar requirements. This is out of scope, but impacted the design of our proof of concept.

IV. Authorization Framework

In order to fulfill the requirements for fine-grained access control we have chosen to use the access control standard XACML [2], since it is the most predominant standard in the area, and has been used in industry to some extent[1]. With XACML we can cover both sub-requirements *R1.a* and *R1.b*.

Evaluating XACML policies is too heavyweight for constrained devices, therefore we propose to externalize most of the authorization decision process (*R2*), and have the device perform primarily authorization enforcement (*R5*).

In order to deal with local conditions affecting the access control decision (*R1.c*), either information about local conditions must be transported to an external policy decision point or some access control decision be made within the constrained device. The latter is preferred for several reasons: transporting information about local conditions for each policy decision introduces delays and adds a transmission costs to the device, and moreover, the local conditions may have changed at the time of enforcement. Furthermore, we can express local conditions as XACML *Obligations*, i.e. constraints under which an external authorization decision is valid.

In order to convey the authorization decisions from the external decision point to the device, we chose to use assertions, which are digitally signed data objects containing asserted information, and in particular we use SAML authorization decision assertions [7] as a template. An alternative would have been to use OAuth access tokens [8] as starting point, the end result would have been the similar, but we choose SAML since it is well integrated with XACML.

We therefore need the following three entities in our framework: A *Device (D)* hosting resources, a *User (U)* wishing to access a resource and an *Authorization Engine (AE)* located in the back-end, that performs policy evaluation and issues authorization assertions for resource access to the User. The User sends these assertions to the device along with the request. The AE is acting on behalf of a device owner who has configured the resource access policies.

Independent of the authorization mechanism, we also need a fourth entity that facilitates resource discovery, a *Resource Directory* maintaining descriptions of resources. Our design extends the use of the directory to also manage security data, relevant to devices (e.g. device public key, capability to process local conditions). Note that for privacy reasons, the resource descriptions may also be subject to access control (see *R3*), and should not be transmitted over an unprotected channel. It is straightforward to apply our access control framework to this data, but this is not discussed further in this paper.

The resulting architecture is illustrated in figure 1.

Given this architecture, the authorization framework requires the following minimal set of functions (consider *R4*):

- The AE must to be able to bind the User to the assertion. If the authorization decision depends on the User's identity the AE also needs to authenticate the User. In cases where the User's identity is not relevant (e.g. a purchased service) a pseudonym can be used

[1] See e.g. http://www.axiomatics.com/customers.html

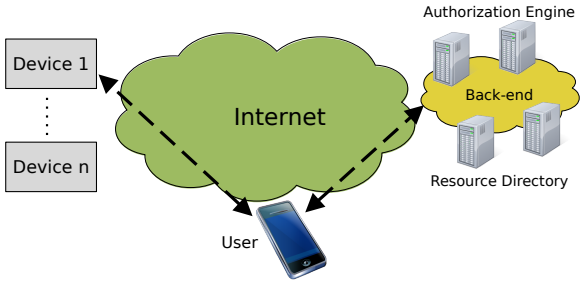

Fig. 1: The authorization architecture

instead. The binding can be achieved by including the public key or the pseudonym of the User in the assertion. The pseudonym can be authenticated using the scheme described in section V.

- The Device must to be able to verify that an assertion is valid and from a trusted source. To achieve this the AE needs to sign the message using a key that is known to, and trusted by the Device.
- The Device must to be able to bind the User to the assertion. This can be achieved by explicit or implicit authentication of the User (or the used pseudonym).

To comply with *R2* the protocols used to implement these functions should use a minimum of message exchanges with the Device, ideally not more than if the Device was accessed without authorization mechanisms. As transport protocol we build upon the IETF Constrained Application Protocol draft (CoAP) [9], adding security information to the CoAP messages where needed. CoAP is specifically designed for constrained devices and features a very low overhead compared to e.g. HTTP, nevertheless our framework is not specific to CoAP and would also work with other application layer protocols.

With the framework described above we can implement functions complying with all our requirements and additionally even fulfill those of Naedele [4].

V. Key establishment

Requirement *R5* assumes a key establishment procedure, and unless keys are provisioned, this has in turn been predated by an authentication procedure. Our authorization framework neither requires a particular authentication protocol nor key agreement procedure, but we must nevertheless account for how keys are established as that impacts what capacity is left in the device for authorization related tasks. We limit ourselves to two main candidates here.

One straightforward option suitable for CoAP is DTLS [10] based on raw public keys or pre-shared keys, in which case the DTLS record protocol provides encryption, integrity and replay protection of CoAP messages. For extremely constrained devices, however, the DTLS handshake may impose a considerable setup time.

As a proof of concept, we modeled the framework on an object security based approach. This approach uses symmetric keys for object protection but works with both symmetric and asymmetric established keys: Assume first that Device and Authorization Engine have established each other's public keys. By including a nonce in the assertion, it is possible to use these static public keys in a Diffie-Hellman calculation to derive a shared secret symmetric key k_{AD} ($C(0e, 2s)$ scheme of [11]). By also including a verified public key of the User (obtained by the Authorization Engine in the assertion request) in the assertion, and a nonce in the payload the Device and User can perform the analog calculation and derive a symmetric key k_{UD}.

Assume now instead that Device and Authorization Engine have established shared symmetric keys. By including a unique User pseudonym instead of public key in the assertion, the Authorization Engine and Device can use a suitable one-way key derivation function to derive a symmetric key k_{UD}, which is also provided to the User in the response to the assertion request. Also in this case it is prudent to include a nonce in object and key derivation to avoid overuse of keys.

Hence given any key establishment between Device and AE we can without loss of generality assume that shared symmetric keys k_{AD} and k_{UD} with AE and U, respectively, are available in the Device after reception of the assertion. These shared keys are the basis for securing the data objects passed between AE and D (assertions), and U and D (payloads).

Note that there are also hybrid approaches, e.g. long lived DTLS used between Device and AE to establish a shared key which can be used for deriving key k_{UD} used to secure objects passed between User and Device.

VI. Authorization Procedures

As a result of the framework outlined in section IV, we need a set of procedures and protocols to perform the following tasks:

A. Device owners registering new devices and their relevant security data.
B. Users finding a device and requesting an authorization assertion for it.
C. Users accessing a device using a previously obtained authorization assertion.

In order to comply with our requirement *R2* we design our protocols such that they do not require additional message exchanges compared to unprotected CoAP exchanges.

A. Registering new devices

For this procedure, we assume the existence of a resource directory such as the IETF Resource Directory [12] which supports procedures for the device to initiate registration of resource description to the directory. We assume that security relevant data for a device such as its public key, the AE it trusts, its owner, and the Obligations which it can process, may be registered as device meta-data in the directory, and can be queried by relevant entities. Publishing this meta-data can follow the same procedure as for publication of the device's resources.

B. Getting an authorization assertion

In order to access a resource on a device, the User needs not only to find the URI of the resource, but also to acquire

an authorization assertion and a cryptographic key to use in security protocols with the device. The resource URI and device related security parameters can be retrieved from the Resource Directory. Among these is the address of the AE trusted by the device.

The User requests an assertion to access a particular resource from this AE, which internally runs the XACML request-response protocol to find out if the User is granted access (see Section VII). If so, the AE returns an assertion and a Device Key to the User.

Depending on whether asymmetric or symmetric keys are used, the assertion contains either a public key or a unique pseudonym of the User. In the former case, the Device Key is the public key of the Device. In the latter case, the Device Key is the derived secret key k_{UD} (Section V). The assertion is signed[2] by the AE using asymmetric or symmetric keys.

Protocol, authentication mechanism and secure transport between User and AE are out of scope of this paper. Assuming that the User and the AE are not resource constrained, common Internet standards such as HTTP and TLS can be applied.

C. Accessing a device

In order to access the resource on the device, the User now sends a CoAP request *including the assertion* to the Device, secured with a protocol/crypto suite supported by the Device. CoAP supports the use of optional request information to be carried as a CoAP *Option* interspersed between header and payload. We propose to introduce an *Assertion Option* in CoAP. Furthermore, in our object security approach we replaced the CoAP payloads with object secured equivalents based on the Device Key obtained from the AE.

The device verifies the assertion, matches the access rights authorized in the assertion with the actual access request, and verifies the local conditions (if any) specified in the assertion.

If all verifications are successful the request is granted with consequential processing and response. Replay protection is provided by giving the assertions a short, pre-defined validity time, and storing on the device a list of recently used assertion identifiers.

While DTLS offers bundled encryption and integrity protection of both payload and headers, the object security approach allows for a trade-off between protection against performance. Depending on the trust model, assertion and payload may need to be encrypted because eavesdropping will reveal information about the User's request, which may be privacy sensitive. Wrapping the the payloads as secure objects allows differentiated protection of the content based on its sensitiveness.

For example, in a CoAP GET request, the assertion could be integrity protected only, while the response payload would be encrypted and integrity protected. In a CoAP PUT/POST the assertion and request payload would be integrity protected and the response would be unprotected.

The security parameters published by the device should specify which protection mode is to be used, and the crypto suite identifiers should preferably be standardized.

[2]The term *signature* is also used for Message Authentication Codes.

VII. Authorization Engine

The Authorization Engine consists of two components: An access control system and an assertion issuing system.

The access control system produces policy-based access control decisions using XACML. How the policies are created and administrated is out of scope for this paper.

When a User is granted access by the access control system, the assertion issuing system encodes the authorization decision as an assertion.

It is possible that the access granted by such an assertion depends on parameters known only to the device, in which case the device will evaluate those and grant or deny access based on the outcome of this evaluation. This means that at least some devices will perform more than pure enforcement of access control decisions.

To enable the device to enforce the authorization decision, the assertion needs to provide the following information:

- Which resource does the decision applies for.
- Which action (GET, PUT, POST, DELETE) does the decision apply for.
- Which subject does the decision apply for, and how can this subject be authenticated (if necessary).
- Which assertion engine has issued this assertion (this information might be implicit from the signature of the assertion).
- Under which other conditions is the assertion valid (expiration date, replay protection, parameters evaluated by the device at access time).

Since the full syntax of XACML Responses and SAML Assertions includes a large number of features, we have defined a subset of both standards, in order to simplify the processing on the Device. Furthermore the XML representation of this subset is too verbose for efficient transmission over limited channels, therefore we have defined a compact JSON-based notation for our SAML and XACML subset. This approach reduces the size of the assertion roughly by a factor of ten.

The assertion shown in the following example would have a size of 208 bytes without pretty printing. The corresponding XML assertion would be 2281 bytes large.

```
01 {
02   "ID": "ID_ffda55f9...097bdd21e6",
03   "II": "2013-02-15T10:02:52Z",
04   "IS": "AAA-Server",
05   "SK": "BvDgLAXSHe...0RLhfwS1fue",
06   "ST": {
07     "OB":{
08       "NB":"09:00:00Z",
09       "NA":"17:00:00Z"
10     },
11     "ACT": "GET",
12     "RES": "coap://node346/tempSensor"
13   }
14 }
```

ID is the assertion identifier, II is the issue instant in UTC format, IS is the identifier of the assertion issuer and SK represents the subject of the assertion, using a public key for subject confirmation.

The authorization decision statement is represented by ST, it contains an abbreviated XACML Obligation OB representing a local condition that is verified on the device. In this case we have a *not-before* (NB) and *not-after* (NA) time that constrains the access permission.

The ACT element is the action and the RES the target resource URI authorized by the assertion. These parameters represent the XACML request.

Note that the XACML response is implied to be a *PERMIT* response. The Authorization Engine does not issue assertions for authorization decisions other than *PERMIT*. Furthermore the *Subject* part of the XACML request is omitted since the Authorization Engine enforces that it corresponds to the SK element of the assertion.

The device needs to know how to process the Obligation, otherwise it must reject the assertion. Section VI describes how a device can publish the types of Obligations it can process.

VIII. Implementation

The device part of our framework was implemented using the object security based approach and symmetric keys (see Section V) on an example platform: The *Arduino Mega 2560* board[3]. This board features a 16 MHz processor, 256 kB of Flash Memory, 8 kB of SRAM, and 4 kB of EEPROM. We chose this board in order to test our approach on the low end of the performance spectrum for target constrained devices.

The board was programmed in C using a custom implementation of the CoAP protocol stack, the *Cryptosuite*[4] library for HMAC-SHA256 and an optimization of the 8-bit AES implementation by Brian Gladman[5].

Processing the CoAP messages on the device, including our authorization handling, requires roughly 7.3 kB of static memory (including Arduino internals such as UDP, Ethernet, SPI libraries, etc), which places us close to the upper limit of what this board can do.

From the required operations the most time consuming ones unsurprisingly turned out to be encrypting, decrypting, integrity protection, and integrity verification. Other operations such as matching the assertion to the requested action turned out to consume only negligible time.

We chose to use the IETF JSON Web Encryption (JWE) [13], an emerging secure object standard, for wrapping the assertion and payload. Note that this wrapping expands the payload size drastically. For example a typical sensor reading could be a 4-byte integer. If that would be protected by AES encryption and a HMAC message authentication code, we would have 128 bytes of encrypted text due to the block-size padding and another 160 bytes for the MAC.

Table I shows the resulting performance figures.

[3]http://arduino.cc/en/Main/ArduinoBoardMega2560

[4]https://github.com/Cathedrow/Cryptosuite

[5]http://gladman.plushost.co.uk/oldsite/AES

Integrity verification of POST request/assertion	*58 ms / 100 ms*
Decryption of POST request/assertion	*231 ms*
Encryption of GET response	*192 ms*
Integrity protection of GET response	*101 ms*

TABLE I. Execution time for cryptographic processing

The processing times were confirmed by measuring in the CoAP client the round trip time of one protected POST message and one protected GET message, which equals to the sum of the corresponding figures in the table plus a fixed time corresponding to the round trip time of the corresponding unprotected message.

These tests used a chip-set without any cryptographical co-processor. Using such specialized hardware could reduce memory use, battery consumption, and improve performance.

IX. Security Evaluation

In the present framework, we aim to protect the following *assets*: The data on devices, the devices themselves, and the services offered by devices.

Our measures to protect these assets are to enforce fine-grained restrictions on accessing the devices (as opposed an all-or-nothing approach, that would just require authentication). Due to the setup of our framework, we also need to protect authorization decisions, the authorization policies, and relevant attributes to make these decisions.

Note that only the protection of authorization decisions needs to be verified on the device, everything else is performed on more powerful back-end machines.

The Authorization Engine is a Trusted Third Party from the point of view of the device owner, which if compromised could e.g. issue assertions to unauthorized parties or use a derived key k_{UD} to decrypt an eavesdropped GET response.

The end-to-end security setting has two sides. Since all data is verified and protected in the device there are no intermediary attack targets for breaking confidentiality or integrity. But also since the device is in principle open for access from arbitrary users, additional overload protection mechanisms may be needed, e.g. external firewall functionality restricting the number of simultaneous requests and/or verifying assertions before forwarding (if possible).

An alternative approach is to use a gateway that has full, direct access to the devices it manages, and filters access requests based on its access control policies. Such an approach has the advantage that all authorization handling is moved to an entity without the resource constraints present on the devices. However one disadvantage is that we cannot maintain end-to-end protection of the protocol messages, since the gateway needs to be able to read them. Thus privacy critical requests cannot be protected should the User distrust the gateway. Furthermore this approach is not applicable to a scenario featuring devices only locally accessible in isolated places.

X. Conclusion and Future Work

We have presented a generic authorization framework for IoT devices built upon existing Internet and access control

standards supporting fine-grained and flexible access control to constrained devices.

The key components in this framework are the Authorization Engine and our newly designed assertion profile defined as subset of SAML and XACML and compactly represented in a JSON notation. Of special significance we used XACML obligations to enable any kind of local decisions in the device. Supporting components are the extension of the Resource Directory for publication of device capabilities for local decisions and enforcement, and key management procedures used to establish security between the Device and the AE/User.

Performance critical parts of this framework have been implemented and tested using a object security based approach on an example device and thereby shown that the authorization procedures can be executed in a reasonable time-frame on certain classes of constrained devices. The security evaluation elucidates the trade-offs and assumptions that where made for this framework and specifies which security assurances the framework provides.

The use of JWE as wrapper format for secure object is suitable for the assertion but highly non-optimal for payloads of a few bytes, which are common in CoAP. Both the JWE header and the crypto payload could be made more compact for this kind of deployment. Potential future work include exploring and standardizing the use of stream-ciphers and MAC for JWE. Other topics for standardization are our assertion profile of SAML and XACML, and device registration of security related meta-data using the Resource Directory.

ACKNOWLEDGMENT

We would like to thank Alexander Maximov from Ericsson Research for his optimization of the 8-bit AES code.

REFERENCES

[1] A. Serbanati, A. S. Segura, A. Oliverau, Y. B. Saied, N. Gruschka, D. Gessner, and F. Gomez-Marmol, "Internet of Things Architecture, Concept and Solutions for Privacy and Security in the Resolution Infrastructure," EU project IoT-A, Project report D4.2, February 2012, http://www.iot-a.eu/public/public-documents.

[2] S. Godik, and T. Moses (eds.), "eXtensible Access Control Markup Language (XACML)," Organisation for the Advancement of Structured Information Standards (OASIS), Standard Version 2.0, February 2005. [Online]. Available: http://www.oasis-open.org/committees/xacml

[3] S. Cantor, J. Kemp, R. Philpott, and E. Maler (eds.), "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML)," Organisation for the Advancement of Structured Information Standards (OASIS), Standard Version 2.0, March 2005. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[4] M. Naedele, "An Access Control Protocol for Embedded Devices," in *Proceedings of the fourth IEEE Conference on Industrial Informatics, INDIN*. Singapore: IEEE, August 2006, pp. 565–596.

[5] B. Resch, B. Shulz, M. Mittlboeck, and T. Heistracher, "Pervasive geo-security a lightweight triple-A approach to securing distributed geo-service infrastructures," *International Journal of Digital Earth*, vol. 5, no. 4, pp. 1–18, 2012.

[6] R. Zhang, Y. Zhang, and K. Ren, "Distributed Privacy-Preserving Access Control in Sensor Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1427–1438, 2012.

[7] E. Rissanen, and H. Lockhart (eds.), "SAML 2.0 Profile of XACML Version 2.0," Organization for the Advancement of Structured Information Standards (OASIS), Committee Specification, August 2010, http://www.oasis-open.org/committees/xacml.

[8] L. Daigle and O. Kolkman, "The OAuth 2.0 Authorization Framework," Internet Engineering Task Force (IETF), Request For Comments (RFC) 6749, October 2012, http://www.ietf.org/rfc/rfc6749.txt.

[9] Z. Shelby, K. Hartke, and C. Bormann, "Constrained Application Protocol (CoAP)," Internet Engineering Task Force, Internet-Draft draft-ietf-core-coap-14, March 2013, work in progress.

[10] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," Internet Engineering Task Force (IETF), Request For Comments (RFC) 6347, January 2012, http://www.ietf.org/rfc/rfc6347.txt.

[11] E. Barker, L. Chen, M. Smid, and A. Roginsky, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography," NIST, Special Publication 800-56A, August 2012.

[12] Z. Shelby, S. Krco, and C. Bormann, "CoRE Resource Directory," Internet Engineering Task Force, Internet-Draft draft-shelby-core-resource-directory-05, February 2013, work in progress.

[13] M. Jones, E. Rescorla, and J. Hildebrand, "JSON Web Encryption (JWE)," Internet Engineering Task Force (IETF), Internet-Draft draft-ietf-jose-json-web-encryption-08, December 2012, work in progress.