Towards Viable Certificate-based Authentication for the Internet of Things

René Hummen[⋆], Jan H. Ziegeldorf[⋆], Hossein Shafagh[⋆†], Shahid Raza[†], Klaus Wehrle[⋆]

[⋆]Communication and Distributed Systems, RWTH Aachen University, Germany
Email: {hummen, ziegeldorf, shafagh, wehrle}@comsys.rwth-aachen.de
[†]Swedish Institute of Computer Science, Kista, Sweden
Email: {hossein, shahid}@sics.se

ABSTRACT

The vision of the Internet of Things considers smart objects in the physical world as first-class citizens of the digital world. Especially IP technology and RESTful web services on smart objects promise simple interactions with Internet services in the *Web of Things*, e.g., for building automation or in e-health scenarios. Peer authentication and secure data transmission are vital aspects in many of these scenarios to prevent leakage of personal information and harmful actuating tasks. While standard security solutions exist for traditional IP networks, the constraints of smart objects demand for more lightweight security mechanisms. Thus, the use of certificates for peer authentication is predominantly considered impracticable. In this paper, we investigate if this assumption is valid. To this end, we present preliminary overhead estimates for the certificate-based DTLS handshake and argue that certificates – with improvements to the handshake – are a viable method of authentication in many network scenarios. We propose three design ideas to reduce the overheads of the DTLS handshake. These ideas are based on (i) pre-validation, (ii) session resumption, and (iii) handshake delegation. We qualitatively analyze the expected overhead reductions and discuss their applicability.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection (e.g., firewalls)*

Keywords

Internet of Things; TLS; Authentication; Certificates

1. INTRODUCTION

To enable secure end-to-end communication for the Web of Things (WoT) [19], lightweight variants of standard security protocols are currently being developed. Recent approaches and implementations predominantly favor symmetric keys or raw public keys for peer authentication [1, 11, 12].

We argue that certificates can solve a number of challenges where particularly symmetric key-based solutions fall short. Specifically, certificates allow authentication of objects and services *across network boundaries* without the need for pre-configured *pair-wise* state. Meta information encoded in a certificate (e.g. group membership or service type) can be used to authorize communication in the context of machine-to-machine interactions. Moreover, compromised objects, or objects and services that are no longer authorized to communicate with others can simply be excluded by announcing these on pre-configured revocation lists. Finally, certificates are an established method of authentication in large-scale systems such as the Internet. Hence, the extensive amount of infrastructure, tools, and expertise can be reused.

However, we also realize that new challenges arise when using certificates on resource-constrained objects. These challenges result from the limited available CPU, ROM, RAM, and energy resources. Especially processing of long certificate chains may be infeasible on a per-connection basis. Furthermore, the transmission of certificates as well as checking of revocation lists increases radio transmissions and, thus, energy consumption considerably. Lastly, certificate validation depends on further functionality that may otherwise not be required, e.g., time synchronization.

In this paper, we present initial design ideas that aim at making certificate-based authentication feasible for resource-constrained objects. Our discussion focuses on Datagram TLS (DTLS) [13]. Still, our observations and ideas also apply to other security protocols, e.g., IKEv2 or HIP. Specifically, we propose to perform *certificate pre-validation* at on-path network entities. Furthermore, we promote *session resumption* as a mechanism to minimize processing and transmission overheads. To this end, we extend the existing DTLS session resumption mechanism for session resumption without client-side state. Finally, we present a standard-compliant procedure to *delegate the security association establishment* to more capable off-path devices.

The rest of this paper is organized as follows. Section 2 outlines the target network scenario, capabilities of the participating objects, and our basic assumptions. We analyze the overheads of certificate validation on smart objects in Section 3. Our improvements to the DTLS handshake are outlined in Sections 4 and 5. Finally, Section 6 summarizes related work and Section 7 concludes this paper.

2. NETWORK SCENARIO

We assume an abstract network scenario consisting of smart objects, gateways, and services. Smart objects are

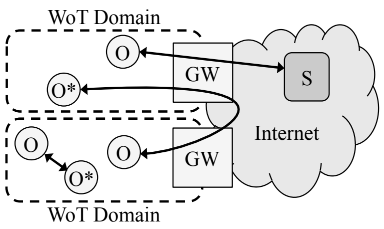

Figure 1: Network scenario with resource-constrained objects (O and O*) that connect to other objects or services (S) via a gateway (GW). Arrows indicate specific communication paths.

IP-enabled and spread across WoT domains. Gateways connect these objects to a backbone infrastructure such as the Internet. Hence, objects can communicate internally within the WoT domain, across WoT domains, and with services in the local network or the Internet (see Figure 1).

Smart objects have limited computational power, memory, and energy. With respect to RAM and ROM, we distinguish between two classes of smart objects: i) class O objects are equipped with about ten kilobytes of RAM and *several tens* of kilobytes of ROM, whereas ii) class $O^\star$ objects have tens of kilobytes of RAM and *tens to hundreds* of kilobytes of ROM. This classification is similar to the classification into class 1 and class 2 objects in [3].

The resources at a gateway range from those of a commodity router to a powerful network appliance depending on the number of smart objects in the WoT domain. Likewise, services run on server hardware or constitute Cloud services with elastic resources. Finally, we assume that the RESTful protocol CoAP [17] for resource-constrained objects is used to transmit application data and that both, smart objects and services, support the DTLS protocol.

3. CERTIFICATE-BASED DTLS

In this section, we discuss preliminary overhead estimates of the certificate-based DTLS handshake with mutual authentication of the communicating peers. Specifically, we consider transmission overheads, processing time, as well as RAM and ROM requirements at a smart object. We identify the minimum capabilities of objects required to perform certificate-based DTLS handshakes in an end-to-end fashion. Moreover, our analysis reveals aspects of the DTLS handshake that demand for further optimization.

3.1 Transmission Overheads

A full DTLS handshake consists of up to six message flights that may each include multiple messages (see Figure 2). Flights 1 and 2 serve as a return routability test for Denial of Service (DoS) protection. Flights 3, 4 and 5 implement the negotiation of the security parameters, peer authentication, and session key establishment. If certificate-based authentication is used, certificate chains are exchanged within flights 4 and 5. The `Finished` messages in flights 5 and 6 finalize the handshake and verify its correctness.

As shown in Figure 2, a full certificate-based DTLS handshake consists of up to 15 messages. These messages may require *fragmentation* at the DTLS layer or below. For example, even short certificates exhibit sizes above 220 bytes [6] and constantly exceed the limited frame size of link layers such as IEEE 802.15.4 (i.e., 127 bytes). However, such

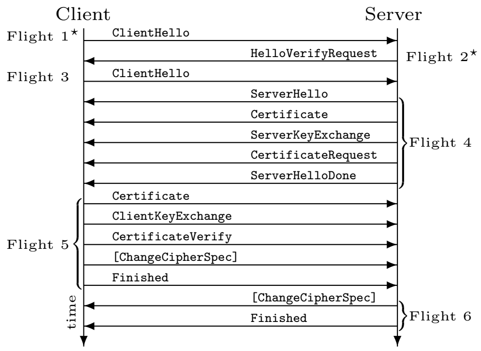

Figure 2: Full DTLS handshake protocol. Messages marked with * are optional.

fragmentation has been shown to potentially be harmful [7]. Additionally, fragmentation increases the actual number of transmitted packets. This increase is especially critical as DTLS defines a retransmission mechanism for lost packets that operates on a *per-flight basis*. As a result, a single lost packet causes the retransmission of an entire flight, thus further increasing the DTLS network overhead.

Furthermore, the packet buffer at a receiving object must be large enough to hold an entire DTLS message. Thus, RAM constraints of an object may prevent the successful reception of large certificates or long certificate chains.

Finally, additional network overheads result from the need for a time synchronization protocol such as NTP and a certificate status protocol such as the Online Certificate Status Protocol (OCSP). These protocols are required to verify the validity period and revocation status of certificates.

3.2 Processing Times

CoAP defines `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8` as the mandatory cipher suite for certificate-based authentication. The smallest curve for this suite is NIST P-256 [10]. With this curve (but also for smaller curves), signature verification and the Diffie-Hellman key agreement denote the dominating cryptographic operations with respect to the processing time of the DTLS handshake.

To estimate these cryptographic overheads, we analyzed the processing time for ECDSA and ECDH operations with curve P-256 on a Tmote Sky mote using implementations from the cryptographic library *relic*[1]. We performed 100 independent ECDSA signature generations and verifications as well as ECDH key-pair generations and key agreements.

We found that a single signature verification, and thus the verification of a certificate, requires about 6 s. This overhead grows linearly with the length of the certificate chain. Additional cryptographic overhead results from the authenticated ephemeral Diffie-Hellman key exchange. Here, the generation of the Diffie-Hellman key-pair requires 2.1 s, whereas 2.5 s are needed to sign the public value. Furthermore, the derivation of a shared secret requires 4.3 s.

To summarize, the cryptographic per-handshake overheads add up to a considerable 15 s for a certificate-based DTLS handshake even with minimal certificate chains of length 2. While these overheads may be acceptable during sporadic configuration phases of a smart object, such delays are likely to render certificate-based security solutions inapplicable if performed on a per-connection basis.

[1]http://code.google.com/p/relic-toolkit/

Functionality	PV	SR	HD
Communication			
DTLS handshake	=	-	-
Additional (NTP, OSCP)	x	-	x
Computation			
Signatures (generation & verification)	=	-	x
Key establishment	=	-	x
Memory			
DTLS protocol	=	+	-
Cryptography (ECC, SHA, AES)	=	=	x
Certificates (ASN.1 Parsing)	=	=	x
Additional (NTP, OSCP)	x	=	x

Table 1: Qualitative improvements with pre-validation (PV), session resumption (SR), and handshake delegation (HD) categorized in additional (+), unchanged (=), less (−), and removed overhead (x).

3.3 RAM and ROM Requirements

With respect to RAM and ROM requirements, we analyzed the tinyDTLS[2] implementation and the public-key library relic with the *msp430-size* and *msp430-objdump* tools. We found that relic is the major contributor to the overall ROM overhead with about 21 kB of ROM. Symmetric key cryptography and the implementation of the cryptographic hash function require about 5 kB. The DTLS protocol implementation itself requires about 10 kB of ROM.

In combination with the code size needed for Contiki OS with a simple CoAP server (41 kB), this overhead amounts to about 77 kB. Likewise, static RAM requirements add up to above 11 kB. Notably, these overhead estimates do not yet consider program functionality for ASN.1 parsing, time synchronization, and certificate status verification.

4. REDUCING END-TO-END OVERHEADS

The results presented in Section 3 indicate that end-to-end certificate verification imposes non-negligible communication, computation, and memory overheads for smart objects. In this section, we propose mechanisms that reduce these overheads for objects of class $O^\star$ through on-path pre-validation and session resumption. As the overheads may prevent tightly resource-constrained objects of class O from performing certificate verification themselves, we additionally present a handshake delegation procedure in Section 5.

4.1 Pre-validation at the Gateway

If a smart object communicates with an object or service outside its WoT domain, the DTLS traffic traverses the interconnecting gateway. This gateway can be dimensioned with the necessary computation and bandwidth resources, depending on the number of attached objects, to act as a gatekeeper to the WoT domain. Specifically, we propose to delegate certificate chain status validation and cryptographic certificate chain pre-verification to the gateway.

When a gateway observes a certificate-based DTLS handshake, it verifies the status of the certificates in the inbound `Certificate` message. To this end, it verifies that their validity has not yet expired and that they have not yet been revoked, e.g, via OCSP. Additionally, it cryptographically verifies the transmitted certificate chain against the configured root certificates of the smart object located in the WoT domain. These certificates could, e.g., be deployed at the gateway during configuration of the smart object. The exact procedure, however, is future work.

The gateway only forwards the inbound `Certificate` message to the smart object, if the certificate pre-validation was successful. Otherwise, it sends a notification message to the smart object informing it about the invalid credentials of the peer. For this purpose, we propose a DTLS alert or an ICMP message that conveys "*invalid DTLS credentials*" along with the observed DTLS Session ID to the smart object.

With benign gateways, only *valid certificates* are sent towards the WoT domain. This reduces the communication overhead resulting from undesired handshakes. Furthermore, pre-validation decreases the memory requirements at a smart object along with network overheads as time synchronization and certificate status validation are no longer performed in the WoT domain (see Table 1). However, as a trade-off, a malicious gateway could forward invalid certificates and implicitly claim these to be valid. As a result, a smart object would establish a DTLS session despite expired or compromised certificates. Still, to exploit this fact in a Man-in-the-Middle (MitM) attack, the gateway would need to be in possession of the corresponding private key.

If a smart object would not *re-verify the signatures* of the certificate chain when receiving a `Certificate` message, the gateway could mount a MitM attack by replacing the leaf certificate with an arbitrary certificate that certifies its own public key. Hence, certificate parsing and cryptographic operations cannot be omitted at a smart object if the gateway is not completely trusted (see Table 1).

4.2 Session Resumption

As shown in Sections 3.1 and 3.2, transmitting and processing of long certificate chains incurs considerable overhead in a DTLS handshake. In order to decrease these overheads, we promote the extensive use of session resumption mechanisms for DTLS-enabled smart objects.

The key idea behind session resumption is to only perform expensive operations once during an initial handshake. The peers then maintain minimal session state, even after session teardown. This allows the peers to use the stored state to authenticate each other and to re-establish the secure channel in subsequent handshakes without the need for certificate transmissions and expensive cryptographic operations.

Two types of session resumption have been proposed for TLS/DTLS. On the one hand, an abbreviated handshake where *both peers maintain session state* across connections is standardized in [4]. Such session resumption is advantageous in scenarios with peers that have similar resource constraints. On the other hand, [14] specifies an extension of the handshake that allows for *server-side offloading of encrypted session state* towards the client during the initial handshake. This state offloading trades off a minimized memory burden on the server side with an increased memory overhead at the client side. Hence, such a session resumption mechanism is highly beneficial when a resource-constrained server in the WoT domain is contacted by another object or service that is equipped with more resources.

However, when a smart object initiates a handshake with, e.g., an Internet service, the asymmetry in resources is contrary to the scenario for server-side state offloading. Hence, in addition, we propose a session resumption mechanism that affords *client-side offloading of encrypted session state*. This mechanism is based on the signaling extensions and

[2]http://tinydtls.sourceforge.net/

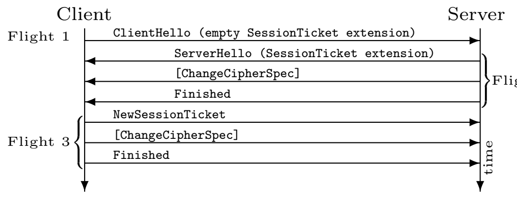

Figure 3: Message flights for the abbreviated TLS handshake with client-side state-offloading.

messages described in [14]. More concretely, the client and the server first perform a full DTLS handshake that is similar to the handshake depicted in Figure 2. However, in contrast to the normal handshake, the `ClientHello` message contains an empty `SessionTicket` extension that indicates the client's support for our session resumption mechanism. Likewise, the server's `ServerHello` message contains an empty `SessionTicket` extension indicating its own support. Finally, the client sends an additional `NewSessionTicket` message containing its encrypted session state in flight 5.

The session resumption uses an abbreviated handshake with three flights (see Figure 3). The `ClientHello` message contains an empty `SessionTicket` signaling the client's support of the session resumption mechanism. The `ServerHello` message includes a `SessionTicket` extension containing the encrypted client-side state of the previous handshake. Flight 3 additionally includes a `NewSessionTicket` message with the encrypted session state of the client for the next handshake. After the session resumption handshake completed, the peers have re-established the secure channel.

While the session resumption functionality causes a slight increase in the memory footprint for the DTLS protocol, the main improvements are the reduced communication and processing overheads for both peers (see Table 1). The negotiation of the appropriate session resumption strategy during the initial handshake and the exact content of the offloaded state for smart objects are future work in this area.

5. HANDSHAKE DELEGATION

As shown in Section 3, the overheads resulting from certificates may be excessive for tightly resource-constrained objects of class O. Our ideas proposed in Section 4 allow to reduce these overheads, but still require a complete initial handshake with a reduced set of certificate-related functionalities. To make certificate-based authentication viable in scenarios with clients of class O, we outline a process that allows to *delegate* the initial handshake to the *owner* of the object. This allows us to unburden the object from all certificate-related overheads as well as most DTLS handshake complexities resulting from, e.g., long packet flights.

The handshake delegation is based on the TLS session resumption without server-side state [14]. It assumes that each smart object O_i has a unique identifier ID_{O_i} and a secret key K_{O_i}. The identifier of an object (e.g., its MAC address) and the secret key may, e.g., be embedded in the object during manufacturing. Furthermore, we assume the existence of a lookup infrastructure for the network address of an object in the WoT domain based on its identifier. The object O_i may, e.g., register to a CoRE Resource Directory [18] when joining the WoT domain. We now briefly describe our proposed delegation procedure (depicted in Figure 4).

1. Out-of-band shared-secret exchange: The smart object O_i and its owner exchange the object identifier ID_{O_i}

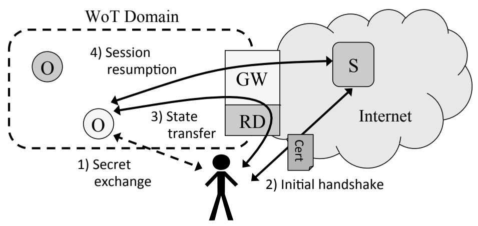

Figure 4: The handshake delegation procedure.

and the secret key K_{O_i} via a secure side channel. For example, the owner may obtain ID_{O_i} and K_{O_i} by scanning a QR-code inside the packaging of the object.

2. Certificate-based TLS or DTLS handshake: The owner establishes a TLS or DTLS session on behalf of the object O_i with the desired service S or another object of class $O^\star$. The *peer object or service* thereby authenticates itself to the owner by means of certificates. The *owner* may either use certificate-based authentication at the TLS/DTLS layer or a username/password combination at the application layer to identify herself to the peer depending on the peer's authentication requirements. In addition, the owner requests a session resumption ticket from the peer within the TLS/DTLS handshake. After a successful handshake, the owner is in possession of the peer's encrypted session-state s that contains all necessary information for the peer to re-establish the current session without the need for further public key operations or certificate transmissions. At this point, the owner terminates the connection with the peer.

3. Session-state transfer: The owner transfers the peer's encrypted session-state s along with its own stored session state s' to the smart object O_i in order to enable the object to resume the DTLS session. To this end, the owner *encrypts and authenticates* both states with the secret key K_{O_i} from step 1, e.g., using AES with CCM: $Enc_{K_{O_i}}(s||s') = c$. Furthermore, the owner requests the network address of the object from the responsible CoRE Resource Directory based on the object identifier ID_{O_i}. She then transmits c to a dedicated CoAP resource at the smart object O_i. Upon receiving the state, the object verifies the authenticity of the state and decrypts the contained information.

4. DTLS session resumption: During this final step, the object O_i uses the received session states in order to resume the DTLS session that was established between the owner and the peer. To this end, the object and the peer perform a session resumption handshake. As the established states of the certificate-based DTLS handshake are used in the handshake to perform mutual authentication, neither certificate-related functionality nor public key operations are required at the smart object O_i. Instead, the delegated state binds the owner's identity at the peer to the object O_i.

To achieve handshake offloading in case of a *constrained server* that is contacted by a more capable client, our proposed session resumption mechanism without client-side state could be used. The client would first connect to a powerful session establishment server in an initial handshake. Afterwards, the establishment server would transfer the encrypted client state to the constrained server as described above and redirect the client to the constrained server. However, the details of such a procedure are future work.

While the transitive security of this approach is weaker than the end-to-end security of DTLS, our delegation procedure removes the need for all certificate-related functionality on a smart object (see Table 1).

6. RELATED WORK

We focus our discussion of related work on approaches that aim at i) enabling TLS/DTLS for smart objects, or ii) reducing the overhead of certificate-based authentication.

In [6] and [8], the authors show that certificate-based authentication with TLS/DTLS is feasible for constrained objects. However, their approaches either do not allow for certificate-based authentication *towards* objects or rely on hardware support (TPM) for computations and key storage. Contrarily, our work focuses on design-level ideas to reduce the DTLS overhead for constrained clients and servers.

The Server-Based Certificate Validation Protocol [5] enables a client to delegate certificate validation to a trusted server. However, this client-centric approach further increases the communication overhead within a WoT domain on a per-handshake basis. Still, its messaging structures and policy representations could be reused in our pre-validation approach to set up validation policies at the gateway.

In [15, 9, 16], the authors propose TLS extensions that allow clients to cache static server information such as certificates. While caching enables to omit information during the handshake, it typically imposes a higher processing and memory burden than required for session resumption, especially as expensive cryptographic operations still need to be performed on a per-handshake basis.

In [2], the authors propose to delegate the IKE session establishment to the gateway. In contrast to our delegation procedure, their approach inherently reveals the end-to-end session keys to an on-path entity and does not consider the case of secure communication within a WoT domain.

7. CONCLUSIONS

Certificates are assumed to incur excessive overheads for many resource-constrained network scenarios and are dismissed as a means of authentication. In this paper[3], we estimate transmission, processing, and memory overheads of the certificate-based DTLS handshake protocol. Our results indicate that certificates are indeed too heavy-weight for *tightly* resource-constrained objects and even involve non-negligible overheads for more capable objects.

To reduce these overheads, we propose three design ideas for the DTLS handshake. First, certificate pre-validation at the gateway effectively turns this on-path entity into a gatekeeper for constrained objects. Second, we promote session resumption as a complementary mechanism to reduce transmission and processing overheads and present an extension of existing mechanisms. Third, for objects that lack the resources to perform a certificate-based DTLS handshake themselves, we propose a standard-compliant handshake delegation procedure that enables the owner of an object to perform the handshake on behalf of her object.

We are currently in the process of implementing and evaluating the certificate-based DTLS handshake for smart objects in order to derive detailed overhead results. Afterwards, our main goal is to implement and evaluate the mechanisms proposed in this paper and to proof their feasibility and improvements. We strongly believe that mechanisms beyond mere header or certificate compression are required to cater for the tight resource constraints in many WoT scenarios and that our work offers first ideas in this direction.

[3]This work was partly funded by SensorCloud (01MD11049) and CALIPSO (FP7-ICT-2011.1.3-288879).

8. REFERENCES

[1] O. Bergmann, S. Gerfes, and C. Bormann. Simple Keys for Simple Smart Objects. In *Workshop on Smart Object Security*, 2012.

[2] R. Bonetto, N. Bui, V. Lakkundi, A. Olivereau, A. Serbanati, and M. Rossi. Secure communication for smart IoT objects: Protocol stacks, use cases and practical examples. In *Proc. of IEEE WoWMoM*, 2012.

[3] C. Bormann. Guidance for Light-Weight Implementations of the Internet Protocol Suite. draft-ietf-lwig-guidance-02 (WiP), IETF, 2012.

[4] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, IETF, 2008.

[5] T. Freeman, R. Housley, A. Malpani, D. Cooper, and W. Polk. Server-Based Certificate Validation Protocol (SCVP). RFC 5055, IETF, 2007.

[6] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, and S. Chang Shantz. Sizzle: A standards-based end-to-end security architecture for the embedded Internet. *Pervasive and Mobile Computing*, 2005.

[7] R. Hummen, J. Hiller, H. Wirtz, M. Henze, H. Shafagh, and K. Wehrle. 6LoWPAN Fragmentation Attacks and Mitigation Mechanisms. In *Proc. of ACM WiSec*, 2013.

[8] T. Kothmayr, C. Schmitt, W. Hu, M. Bruenig, and G. Carle. A DTLS Based End-To-End Security Architecture for the Internet of Things with Two-Way Authentication. In *Proc. of IEEE SenseApp*, 2012.

[9] A. Langley. Transport Layer Security (TLS) Snap Start. draft-agl-tls-snapstart-00 (WiP), IETF, 2010.

[10] D. McGrew, D. Bailey, M. Campagna, and R. Dugal. AES-CCM ECC Cipher Suites for TLS. draft-mcgrew-tls-aes-ccm-ecc-06 (WiP), IETF, 2013.

[11] R. Moskowitz. HIP Diet EXchange (DEX). draft-moskowitz-hip-rg-dex-06 (WiP), IETF, 2012.

[12] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig. Securing Communication in 6LoWPAN with Compressed IPsec. In *Proc. of IEEE DCOSS*, 2011.

[13] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, IETF, 2012.

[14] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. RFC 5077, IETF, 2008.

[15] S. Santesson and H. Tschofenig. Transport Layer Security (TLS) Cached Information Extension. draft-ietf-tls-cached-info-13 (WiP), IETF, 2012.

[16] H. Shacham, D. Boneh, et al. Fast-track session establishment for tls. In *Proc. of NDSS*, 2002.

[17] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. Constrained Application Protocol (CoAP). draft-ietf-core-coap-13 (WiP), IETF, 2012.

[18] Z. Shelby, S. Krco, and C. Bormann. CoRE Resource Directory. draft-shelby-core-resource-directory-04 (WiP), IETF, 2012.

[19] D. Zeng, S. Guo, and Z. Cheng. The web of things: A survey. *Journal of Communications*, 2011.