Delegation-based Authentication and Authorization for the IP-based Internet of Things

René Hummen*, Hossein Shafagh¶, Shahid Raza‡, Thiemo Voigt‡§, Klaus Wehrle*
*Communication and Distributed Systems, RWTH Aachen University, Germany
Email: {hummen, wehrle}@comsys.rwth-aachen.de
¶Department of Computer Science, ETH Zurich, Switzerland
Email: shafagh@inf.ethz.ch
‡SICS Swedish ICT, Kista, Sweden §Uppsala University, Sweden
Email: {shahid, thiemo}@sics.se

***Abstract*—IP technology for resource-constrained devices enables transparent end-to-end connections between a vast variety of devices and services in the Internet of Things (IoT). To protect these connections, several variants of traditional IP security protocols have recently been proposed for standardization, most notably the DTLS protocol. In this paper, we identify significant resource requirements for the DTLS handshake when employing public-key cryptography for peer authentication and key agreement purposes. These overheads particularly hamper secure communication for memory-constrained devices. To alleviate these limitations, we propose a delegation architecture that offloads the expensive DTLS connection establishment to a delegation server. By handing over the established security context to the constrained device, our delegation architecture significantly reduces the resource requirements of DTLS-protected communication for constrained devices. Additionally, our delegation architecture naturally provides authorization functionality when leveraging the central role of the delegation server in the initial connection establishment. Hence, in this paper, we present a comprehensive, yet compact solution for authentication, authorization, and secure data transmission in the IP-based IoT. The evaluation results show that compared to a public-key-based DTLS handshake our delegation architecture reduces the memory overhead by 64 %, computations by 97 %, network transmissions by 68 %.**

I. Introduction

The proliferation of IP technology for constrained network environments enables a new class of networked devices with highly limited computation and memory resources to autonomously participate in the Internet of Things (IoT). Specifically, IP technology enables a constrained device to communicate in an end-to-end fashion with other constrained devices or services that are located in remote network domains (see Fig. 1). An IP-enabled sensor device in a body area network, for example, can transparently send its gathered medical data to an e-health service without the need for any application-layer interactions at the gateway [1]. However, in such scenarios, the transmitted information may be routed via untrusted network infrastructure such as the Internet. Thus, peer authentication and end-to-end data protection are crucial requirements to prevent eavesdropping on sensitive information or malicious triggering of harmful actuating tasks.

To afford *interoperable* network security between endpoints from independent network domains, variants of traditional end-to-end IP security protocols have recently been proposed for constrained devices and the networks formed by them. These protocol variants include Datagram TLS (DTLS) [2], HIP DEX [3], and minimal IKEv2 [4]. All of these protocol variants consider public-key cryptography in their protocol design. As this type of cryptography incurs significant processing and transmission overheads in constrained network environments [5], research and standardization currently aim at reducing the public-key-related overhead during the protocol handshake. Proposed approaches in this context include session resumption mechanisms [6] and caching of static handshake information such as certificates [7]. However, as we identify in this paper, the considerable RAM and ROM requirements render the use of public-key cryptography infeasible for a wide range of constrained devices in the first place.

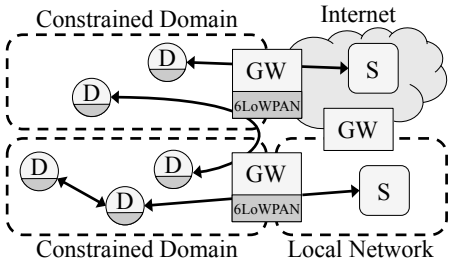

Fig. 1. Constrained devices (D) communicate with each other and with local or Internet-based services (S) via a gateway (GW). Entities belonging to a constrained network domain are equipped with the 6LoWPAN layer. Arrows indicate the achievable connectivity with our delegation architecture.

Our contributions in this paper are threefold. First, we detail and quantify the impact of public-key cryptography on the memory requirements of a DTLS protocol implementation for constrained devices. We chose DTLS for our analysis as it is anticipated to become the primary IP security solution for the IoT[1]. Second, we propose a delegation architecture that enables memory-constrained devices to communicate securely across independent network domains. Specifically, we separate the DTLS connection establishment from the protection of application data and offload the connection establishment to a delegation server. By subsequently handing over the established connection context to the constrained device, this device no longer needs to implement expensive public-key cryptogra-

[1]The DTLS In Constrained Environments (DICE) working group was recently chartered at the IETF with the goal to improve the applicability of DTLS for constrained devices and the networks formed by them.

978-1-4799-4657-0/14/$31.00

phy for the connection establishment and can leverage efficient symmetric-key cryptography for the protection of application data. Third, we show how our delegation architecture naturally provides authorization functionality for a constrained network domain. More precisely, by leveraging its central role in the initial connection establishment, the delegation server can exercise control over the initialization of new connections for local constrained devices. The evaluation results confirm that our proposed delegation architecture considerably improves the feasibility of DTLS-protected communication across network domains for memory-constrained devices. In addition, our architecture achieves similarly low transmission and computation overheads as a purely symmetric-key-based DTLS handshake.

This paper is structured as follows. Section II gives a brief overview of the network scenario and the DTLS protocol. In Section III, we highlight the memory impact of public-key cryptography on a DTLS protocol implementation. We then introduce our delegation architecture in Section IV. Here, we also show how to authorize new connection establishments. In Section V, we discuss the security considerations of our proposed delegation architecture. We then present the results of our detailed evaluation in Section VI. Finally, Section VII examines related work and Section VIII concludes this paper.

II. Prerequisites

We now provide a brief overview of our abstract network scenario and the DTLS protocol as the basis of our work.

A. Network Scenario

As depicted in Fig. 1, our assumed network scenario consists of constrained devices, local or Internet-based services, and gateways that interconnect the different network domains and communication end-points. Constrained devices are assumed to communicate via constrained link layer technologies such as IEEE 802.15.4. Hence, transmissions within a constrained network domain involve lossy wireless links with significant packet size limitations. Additionally, we assume constrained devices to be IP-enabled and to be equipped with 6LoWPAN [8], an IPv6 adaptation layer for constrained link layers standardized at the IETF. 6LoWPAN-enabled gateways then act as mere IP packet forwarders and connect the constrained network domains to the local IP network infrastructure or the Internet via regular wired or wireless connections.

With respect to processing and storage resources, we assume that constrained devices are equipped with only a few MHz of computation power, several kilobytes of RAM and several tens of kilobytes of ROM. Furthermore, constrained devices may be battery-powered. Thus, besides computation and memory overheads, energy efficiency constitutes an additional important factor when considering protocol feasibility in constrained network environments. Gateways and services, on the contrary, run on commodity, wall-powered network and server hardware, respectively. Hence, these devices denote comparably unconstrained nodes in our abstract network scenario. Notably, this network scenario reflects a wide range of IP-enabled network setups, e.g., in the context of e-health, home automation, and industrial control applications.

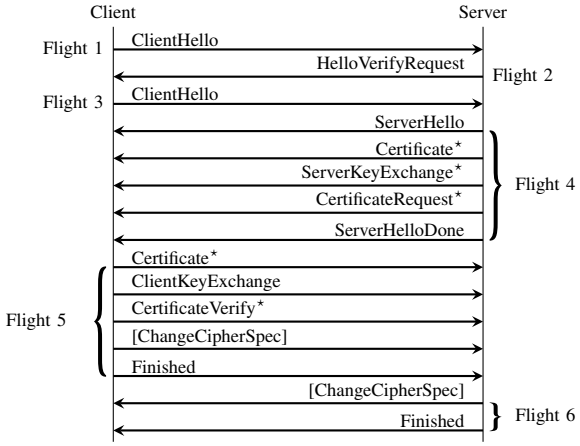

Fig. 2. Message flow of the DTLS protocol handshake. Messages marked with a star (*) are optional and their transmission depends on the negotiated cryptographic primitives for peer authentication and key agreement.

B. The Datagram TLS Protocol

The DTLS protocol is the datagram variant of the widely used Transport Layer Security (TLS) protocol. DTLS was originally developed to secure streaming applications such as VoIP. To this end, the DTLS protocol design replaces the underlying reliable transport channel of TLS with an unreliable channel, e.g., as provided by UDP. Besides this modification, the main design goal of the DTLS specification is to preserve the semantics and security guarantees of the TLS protocol.

To establish a new DTLS security context for the protection of application data, a client and a server perform a connection handshake that consists of up to 15 messages (see Fig. 2). These handshake messages are bundled into 6 message flights. Flights 1 and 2 serve as a return-routability test for Denial of Service (DoS) protection. The central part of the handshake consists of flights 3, 4, and 5. Here, the peers first negotiate the cipher suites to be used for the cryptographic operations. The peers then authenticate each other and carry out a key agreement for the protection of the subsequent handshake messages and application data. In this context, DTLS supports a wide range of public-key-based primitives and optionally provides for a purely symmetric-key-based handshake. The `Finished` messages in flights 5 and 6 conclude the handshake. These messages allow both peers to verify the correctness of the performed handshake. The peers, therefore, compute a hash over the entire handshake and transmit this information to the communication partner in an encrypted form. Successful decryption and hash verification then validate the derived session keys and the integrity of the entire handshake. Finally, the peers use the derived session keys to protect the transmission of subsequent application data.

III. Public-Key Cryptography in the DTLS Handshake

Public-key cryptography is the de-facto standard for peer authentication across independent network domains in the Internet. It enables two peers to only exchange the *public* portion of a public/private key-pair over untrusted infrastructure. Based on these credentials, the owner of the corresponding private key can then securely be authenticated. By augmenting the public key with further information such as a domain name

via certificates, the public key can additionally be bound to a specific device or network domain. Hence, for peer authentication, both communicating end-points only need to trust a common certificate authority and are not required to maintain a pre-shared security context.

However, public-key cryptography involves significant computation, transmission, and memory overheads in the context of constrained devices [5], [6]. Here, we focus on the memory aspects of public-key cryptography and show that its use in IP security protocols may incur prohibitive overheads regarding constrained devices with limited memory resources. We now highlight and quantify the memory impact of public-key cryptography on a DTLS protocol implementation for constrained devices and its protocol mechanisms. We structure our discussion along the following two dimensions: i) run-time memory requirements, and ii) implementation size. We refer to Section VI for detailed information about our evaluation setup.

Run-time memory requirements. During the memory analysis of our public-key-enabled DTLS implementation, we found that the support of the certificate-based handshake requires a notable amount of RAM resources for constrained devices. Specifically, public-key functionality and certificate parsing require about 1.4 kB of static RAM in our evaluation setup. This overhead mainly stems from the *relic toolkit*[2] as the underlying cryptographic library of our protocol implementation. The choice of a different cryptographic library would not have changed this observation significantly as similar RAM requirements were also reported for other open source libraries with support for elliptic curve cryptography (ECC) [9].

In addition to this inherent overhead of public-key cryptography, its application in the DTLS handshake also influences the memory requirements of the DTLS protocol mechanisms. Most importantly, we found that the packet processing of the certificate-based handshake requires about 1.3 kB of static RAM for the correct handling of out-of-order or lost handshake messages. This high overhead is exclusively due to the specific design traits of the DTLS protocol. More precisely, a *receiving peer* must maintain the original sending order of the delivered messages when computing the handshake verification hash for the `Finished` message. As a result, this peer must buffer all packets of a message flight until it can restore the correct sending order. Likewise, the retransmission mechanism of the DTLS protocol requires a *sending peer* to buffer sufficient information for the recreation of an entire message flight as retransmissions operate on a per-flight basis (see Fig. 2).

Considering the entire DTLS protocol functionality, we observed static RAM requirements of about 6.2 kB and a maximum stack size of about 1.8 kB for the certificate-based handshake in our evaluation setup. To put these numbers into perspective, e.g., the TelosB platform [10] is equipped with only 10 kB of RAM. Hence, the RAM requirements of public-key cryptography in IP security protocols such as DTLS may often be prohibitive for such constrained devices. This is especially true when considering the additional memory requirements of the operating system and the application logic.

Implementation size. Similar to our observations concerning the run-time memory requirements, our analysis also revealed an extensive utilization of ROM resources. Specifically, more than 50 % of the total ROM requirements of 41.3 kB for our public-key-enabled DTLS implementation can be attributed to the *relic toolkit*. Also here, the use of a different cryptographic library would not have dramatically changed this observation as other libraries with ECC support likewise require 16 kB of ROM or more [9]. Interestingly, certificate parsing – another aspect that often is considered costly with respect to memory resources – can be implemented with as few as 1.5 kB of ROM when focusing on a critical subset of all standardized certificate extensions and skipping over unsupported ones. Hence, while public-key support itself requires extensive ROM resources on constrained devices, mere certificate parsing only adds a modest overhead to the certificate-based DTLS handshake. This is an especially important observation when considering potential hardware support for ECC operations.

Still, already the base functionality of the DTLS protocol, i.e., excluding public-key cryptography, incurs a non-negligible ROM overhead of about 17 kB in our protocol implementation. A major part of this overhead stems from the complex handshake mechanisms in the DTLS protocol design. Importantly, this unavoidable overhead is further elevated by additional functionality that is required for the verification of certificates during the DTLS handshake. More precisely, certificate verification requires loose but global *time synchronization* to verify the validity period of a certificate as well as functionality for *certificate status verification* to ensure that a certificate has not been revoked at the time of handshake execution. Thus, the certificate-based DTLS handshake quickly requires extensive ROM resources. This is especially critical for devices that, e.g., are based on the 16-bit MSP430 micro-controller platform. These devices can only address a maximum memory space of 64 kB for both RAM and ROM combined. For such and similarly memory-constrained devices, the certificate-based DTLS handshake may therefore often be infeasible.

To summarize, public-key cryptography enables secure communication with low management efforts as cryptographic identities can be exchanged over untrusted infrastructure and can additionally be bound to specific devices via certificates. However, the certificate-based DTLS handshake incurs memory overheads that often are prohibitive for constrained devices. Hence, to enable secure communication for devices with insufficient RAM or ROM resources regarding the certificate-based handshake, an alternative approach to establish a secure connection across independent network domains is needed.

IV. Design

With support for a symmetric-key-based handshake, the DTLS protocol already provides an efficient alternative to public-key cryptography. More precisely, symmetric-key-based peer authentication and key agreement in the DTLS handshake incur near zero additional memory overhead as the required cryptographic primitives are also used for other tasks in the protocol design, e.g., the protection of application data. However, the symmetric-key-based DTLS handshake requires secret keys to be pre-shared and readily available at both communication end-points when establishing a secure end-to-end connection. In other words, the symmetric-key-based handshake requires a key provisioning mechanism that securely deploys secret information at the end-points *before* a secure

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

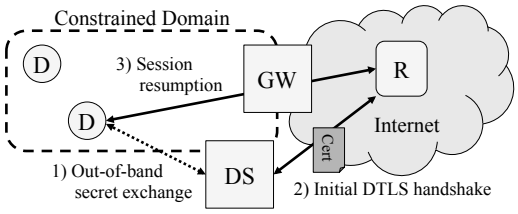

Fig. 3. The handshake delegation procedure. When a new constrained device (D) joins the local network domain, the delegation server (DS) imprints a master key in the constrained device. DS can then perform a certificate-based DTLS handshake with a remote end-point (R) on behalf of D. By encrypting the security context with the master key and offloading it to R, DS can securely hand over the security context of the established DTLS connection to D.

DTLS connection can be established. To solve this dilemma especially when the end-points belong to independent network domains and communicate over untrusted infrastructure, we now present the design of our *delegation architecture* as a simple key provisioning mechanism. We then show how our architecture can be used to *authorize* communication inside a local network domain and for remote services.

A. Delegating the DTLS Connection Establishment

The main goal of our proposed delegation architecture is to reconcile the following two conflicting objectives. On the one hand, we aim at facilitating established best practices for peer authentication in the Internet. We, thus, intend to re-use existing security protocols and public-key infrastructures for authentication purposes across independent network domains. On the other hand, we want to enable constrained devices with limited memory resources to participate in secure communication with remote end-points. We thereby primarily focus on the scenario where the remote end-point, e.g., an Internet service, has sufficient resources for a standard DTLS protocol implementation as well as for the support of public-key cryptography and certificate validation functionality.

Architectural overview. As depicted in Fig. 3, the main building block of our proposed delegation architecture is the introduction of a *delegation server* that allows to separate the initial DTLS connection establishment from the subsequent protection of application data. The main task of the delegation server is to provide a constrained device with the necessary security context to securely communicate with a remote end-point. Hence, our delegation server has a similar role to a key distribution center in protocols for local network security such as Kerberos. However, in contrast to these approaches, our proposed delegation server does not maintain pre-shared secret keys for all potential communication partners. Instead, it establishes an *on-demand security context* with a remote end-point to achieve a scalable solution with respect to the vast number of Internet services that a constrained device can potentially communicate with. To establish this on-demand security context, the delegation server acts on behalf of a constrained device during a certificate-based DTLS handshake. The delegation server then hands over the established security context to the constrained device (see steps 2 and 3 in Fig. 3).

Bootstrapping a constrained device. To allow for a protected exchange of the security context between the delegation server and the constrained device, we require both entities to share a symmetric key, i.e., the *device-specific master key* (see step 1 in Fig. 3). This key may, e.g., be imprinted in the device via physical contact [11] or wireless communication [12] with the delegation server when bootstrapping the device into the local network domain. In addition, this bootstrapping procedure provides the delegation server with information about the DTLS cipher suites that are supported by the constrained device for the protection of application data. With this information, the delegation server can then establish a DTLS connection with a remote end-point on behalf of the constrained device.

Delegation procedure. In our delegation architecture, we assume that the delegation server and constrained devices in a local network domain are administered by a *common operator*. This operator can, e.g., be the owner of a constrained device in a private network scenario or a group of professional network administrators in an enterprise setting. To afford a new interconnection between a constrained device and a remote end-point, the operator instructs the delegation server to establish a DTLS connection with this end-point. During this connection establishment, the delegation server and the remote end-point mutually authenticate each other. More precisely, the delegation server authenticates the remote end-point via certificates during the DTLS handshake. To this end, the delegation server maintains a pool of trusted certificates similar to, e.g., today's web browsers. The delegation server, in turn, either authenticates to the remote end-point via certificates during the DTLS handshake or via an application-level password immediately after the handshake completed. In the former case, the operator additionally needs to provide the remote end-point with a certificate identifying the delegation server prior to the handshake, e.g., via the web-interface of an Internet service.

The main goal of our delegation procedure is to hand over the security context of the above connection to a constrained device. To this end, the delegation server and the remote end-point employ the session resumption extension of the DTLS protocol [13], [14] during the above connection establishment. With session resumption, the delegation server and the remote end-point agree on maintaining sufficient information of the original connection to resume the connection after it has been terminated. Session resumption additionally allows one end-point to transfer its security context to the other end-point in an encrypted *session ticket* for safe-keeping. This enables the offloading end-point to remain stateless while a connection is inactive and to re-claim its security context in a subsequent session resumption handshake. The delegation server leverages this mechanism to push its security context over to the remote end-point. It thereby encrypts its security context with the device-specific master key that it shares with the constrained device. The delegation server further attaches the IP address of the constrained device to the session ticket in plain form. After the handshake between the delegation server and the remote end-point has completed, the delegation server closes the connection and purges its local connection state.

At this point, the remote end-point can establish a secure connection with the constrained device (see step 3 in Fig. 3). To this end, it initiates a session resumption handshake with the constrained device at the previously announced IP address. Likewise, the delegation server can trigger the session resumption handshake at the constrained device by sending an initiation command with the IP address of the remote end-

point to the constrained device. During the session resumption handshake, the remote end-point transfers the session ticket with the encrypted security context of the delegation server to the constrained device. The constrained device then decrypts and uses this context to authenticate and to re-establish the previous connection with the remote end-point.

Notably, the session resumption handshake has an abbreviated form and neither requires public-key cryptography nor certificate-related functionality as it is strongly based on the previously established security context. This allows us to unburden the constrained device from all certificate-related overheads as well as most DTLS handshake complexities that result from large message flights. Moreover, session resumption enables the end-points to tear down and to efficiently re-establish a DTLS connection when needed. This allows to handle multiple DTLS connections even on memory-constrained devices. Finally, new session keys for the protection of application data are derived during each session resumption handshake. Hence, security properties do not degrade even if large amounts of data were to be transmitted.

B. Authorizing Inter- and Intra-Domain Communication

By establishing secure connections on behalf of a constrained device, our delegation server can effectively control which remote end-point this device can communicate with. To prevent an unauthorized end-point from interacting with a constrained device, we require all constrained devices to only accept DTLS handshakes involving its device-specific master key. These master keys are only shared with the delegation server. The delegation server in turn only establishes new connections to authenticated services that were triggered by the operator of the local network domain. With these policies, our delegation architecture naturally allows to restrict connections to remote end-points that were authorized by an operator.

Until now, we only employ our delegation architecture – and thus its authorization capabilities – for remote end-points. To also facilitate authorization of communication within the local network domain, we extend our delegation architecture to connection establishments between local constrained devices. Importantly, the delegation server obtains the device-specific master key of each local constrained device during the bootstrapping procedure. This enables the delegation server to perform a symmetric-key-based DTLS handshake with a constrained device *A* on behalf of another constrained device *B*. During this handshake, the delegation server employs the DTLS session resumption extension and encrypts the established security context for device *B* in a session ticket. This allows each of the two constrained devices to trigger a session resumption handshake as presented above. As the initial handshake involves the device-specific master key between the delegation server and device *A*, device *A* is ensured that the current connection has been authorized by the operator of the local network domain. Likewise, device *B* verifies this fact by successfully decrypting the session ticket with its device-specific master key during the session resumption handshake.

Revocation of authorizations. The operator of the local network domain may eventually decide to revoke the authorization of a specific DTLS connection involving a local constrained device or a remote end-point. To handle such a revocation, the delegation server includes a connection identifier in the encrypted part of its session tickets. When a connection is revoked, the delegation server instructs the constrained device to add the identifier to a list of invalid connections. The constrained device then closes a potentially active connection and no longer accepts handshakes involving this identifier.

As the connection blacklist may grow in size over time and occupy non-negligible memory resources at a constrained device, the delegation server can also instruct a device to change the decryption key for the session tickets. This step invalidates previously issued session tickets and enables the device to flush its blacklist. To afford such resetting of the blacklist without the need to completely re-bootstrap the constrained device, the delegation server no longer directly employs the device-specific master key when encrypting security contexts for the constrained device. Instead, the delegation server and the constrained device perform a symmetric-key-based DTLS handshake as part of the bootstrapping procedure. The peers then employ the *derived key* of this handshake, which would otherwise be used for the protection of application data, to encrypt and decrypt security contexts in session tickets. To reset the blacklist, the delegation server and the constrained device simply perform another symmetric-key-based DTLS handshake and establish a new derived key. The delegation server then re-establishes the security contexts with those end-points that the device is still authorized to interact with.

V. Security Considerations

We now briefly discuss attacks that an adversary can mount against our delegation architecture in the order of their severity.

Compromised remote end-point. If an adversary compromised a remote end-point, e.g., an Internet service, this adversary could retrieve a *large quantity* of stored security contexts for inactive connections between constrained devices and the compromised end-point. While session tickets are always stored in encrypted form at the remote end-point, the adversary could still get access to the security contexts of the compromised end-point. This, however, is similarly true for the private key of the remote end-point when employing public-key cryptography. Hence, we leverage current best practices for public-key cryptography to mitigate such an attack. Specifically, we require the remote end-point to encrypt its own security context while a connection is inactive and revoke compromised security contexts. For revocation purposes, we facilitate the revocation procedure that we presented above.

Compromised delegation server. An adversary may also compromise the delegation server and retrieve all device-specific master keys of the local network domain. To prevent the adversary from issuing new or revoking existing authorizations for local constrained devices, the operator must revoke the certificate of the delegation server and issue a new one after the delegation server has been sanitized or replaced. Moreover, local devices must be re-bootstrapped and imprinted with new device-specific master keys. After that, the authorized connections for all devices in the local network domain can be re-established by the delegation server. While limited to a single network domain, our delegation server must be hardened against this attack similar to a key distribution center for local network security solutions to prevent information leakage.

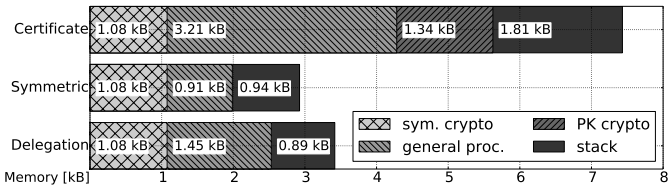

Fig. 4. RAM overhead of the certificate-based DTLS handshake, the symmetric-key-based DTLS handshake, and our delegation architecture. Our delegation approach requires slightly more RAM than the symmetric-key-based handshake, but considerably less than the certificate-based handshake.

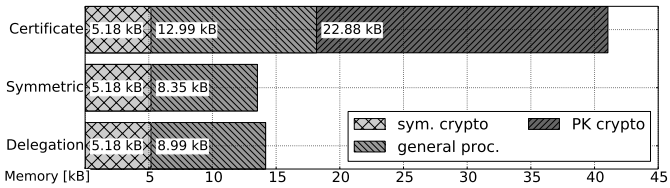

Fig. 5. ROM overhead of the certificate-based DTLS handshake, the symmetric-key-based DTLS handshake, and our delegation architecture. The public-key library constitutes the major contributor to the ROM requirements of the certificate-based DTLS handshake.

Compromised constrained device. If an adversary compromised a constrained device, the adversary could gain access to its master key. Hence, the compromised device must be re-bootstrapped and imprinted with a new master key. Likewise, the delegation server must revoke and re-establish authorized connections for this specific device to prevent the adversary from impersonating the device towards others. Notably, this attack is not unique to our delegation architecture and similarly exists when employing public-key cryptography.

VI. Evaluation

For our evaluation, we extended the symmetric-key-based tinyDTLS implementation[3] with support for the certificate-based handshake. Moreover, we added the required session resumption functionality for our delegation architecture. As constrained devices, we employed the WiSMote platform [15] featuring a 16 MHz MSP430 micro-controller, 16 kB of RAM, 128 kB of ROM, and an IEEE 802.15.4 radio interface. This platform supports extended 20 bit addressing. We used Contiki OS [16] in version 2.7 as the underlying operating system and deployed our WiSMotes in the public testbed FlockLab [17] for evaluation purposes. Notably, while we were unable to run the certificate-based DTLS handshake on TelosB motes, the extended addressing space of the WiSMotes enabled us to run and compare the certificate-based DTLS handshake, the symmetric-key-based handshake, and our proposed delegation architecture. Importantly, we also successfully verified that our delegation architecture is supported by TelosB motes.

With respect to the cryptographic primitives, we followed current security recommendations for constrained network environments [18]. Specifically, we used elliptic curve NIST P-256 for public-key operations, AES-128 for symmetric-key operations, and SHA256 for hashing purposes. For the public-key operations, we employed the open source *relic toolkit*. To evaluate the certificate-based DTLS handshake, we created a self-signed certificate for our own root certificate authority (CA) based on the above elliptic curve. This certificate was pre-provisioned on all devices in our evaluation setup. Furthermore, we issued per-device certificates that were signed by our root CA. With this setup, the certificate-based handshake required the transmission of one certificate per end-point and one signature verification for the validation of a certificate chain. Note that our measured transmissions constitute a lower bound on the certificate-related overhead as device certificates in our evaluation setup had a small size (i.e., 0.37 kB) and did not involve intermediate certificates in the certificate chain.

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

A. Reduced RAM and ROM Requirements

To derive RAM and ROM estimates for our DTLS implementation, we analyzed the Contiki binaries with the *msp430-size* and *msp430-objdump* tools. We distinguished between three variants: i) with certificate-based DTLS functionality, ii) limited to the purely symmetric-key-based DTLS handshake, and iii) with additional support for our delegation architecture. Furthermore, we analyzed the maximum stack size of the different DTLS handshake variants by initializing the RAM resources of the constrained devices with a well-defined pattern as the first task during the device boot-up phase. After the successful conclusion of the handshake, we then dumped the RAM content. By determining the amount of dynamic memory that was overwritten during the execution of the handshake, we were able to estimate the maximum stack size of our DTLS implementation.

As illustrated in Fig. 4 and 5, the certificate-based DTLS handshake requires almost three times the RAM and ROM resources of the symmetric-key-based handshake. Public-key cryptography constitutes the key differentiator with an overhead of 1.34 kB of static RAM and 22.88 kB of ROM. Notably, also the memory overhead of the DTLS base functionality increases when comparing the symmetric-key-based to the certificate-based DTLS handshake. This increase primarily stems from the additional and more complex DTLS message processing and the increased retransmission buffers for the longer message flights of the certificate-based handshake (compare non-marked messages in Fig. 2 with the complete handshake). Importantly, the retransmission-related RAM overhead further grows when larger certificates or longer certificate chains are employed than the ones used in our evaluation setup.

In addition to these static memory overheads, the maximum stack size of the certificate-based DTLS handshake (1.81 kB) is almost twice as high as the stack size of the symmetric-key-based handshake (0.94 kB). Again, the most significant overhead driver is the public-key library that dynamically allocates memory during its initialization and for the actual cryptographic operations. Overall, the certificate-based DTLS handshake trades a low management effort of peer authentication and key agreement across independent network domains for significant memory requirements. These overheads, however, may often be prohibitive for constrained devices that exhibit memory limitations similar to the TelosB platform with 48 kB of ROM, 10 kB of RAM, and 16 bit addressing.

In contrast, our proposed *delegation architecture* enables a constrained device to offload all public-key- and certificate-related operations to the delegation server. This relieves the

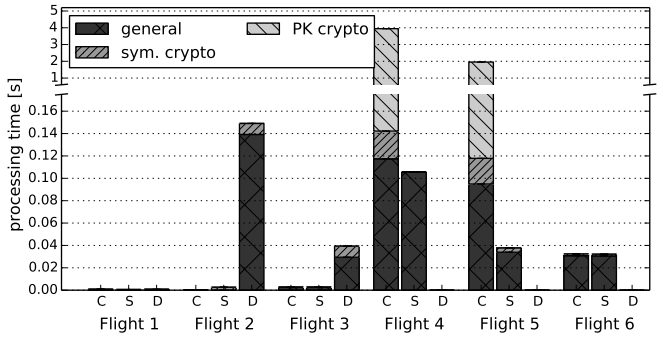

Fig. 6. Client-side computation overhead for the certificate-based DTLS handshake (C), the symmetric-key-based DTLS handshake (S) and our delegation architecture (D) on a per flight basis. Our delegation architecture incurs a low processing overheard, similar to the symmetric-key-based handshake.

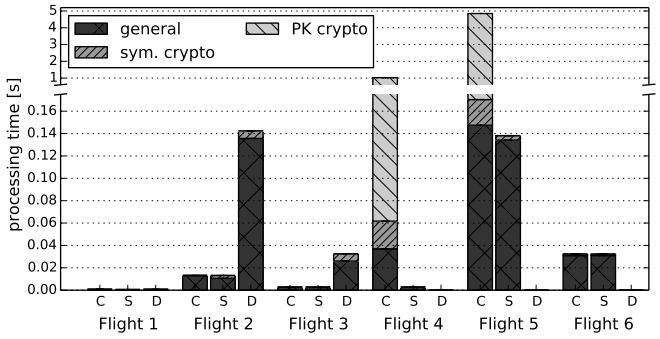

Fig. 7. Server-side computation overhead for the certificate-based DTLS handshake (C), the symmetric-key-based DTLS handshake (S) and our delegation architecture (D) on a per flight basis. The processing overheads on the server-side are similar to those on the client-side.

constrained device from a considerable memory burden. As depicted in Fig. 4 and 5, the memory requirements of our proposed delegation architecture, in fact, only show a marginal increase compared to the purely symmetric-key-based DTLS handshake. More precisely, RAM and ROM requirements increase by 0.54 kB and 0.64 kB, respectively. This slight increase originates from the additional protocol logic for the session resumption extension and the storage buffer for a small number of session tickets. To summarize, devices with sufficient memory resources for the symmetric-key-based DTLS handshake are very likely to also support our delegation architecture and, thus, can participate in authorized and secure communication across independent network domains.

B. Additional Run-time Improvements

We also evaluated the run-time performance of our proposed delegation architecture in constrained network environments. As baselines for this evaluation, we measured the computation and transmission overheads of the DTLS protocol for the certificate-based and symmetric-key-based handshakes between two wirelessly connected WiSMotes. We then compared these baselines against the results for our delegation architecture. The presented results denote averages over 100 measurement runs. The standard deviation was below 13.9 ms for the public-key-related operations and below 1.55 ms for the remaining processing tasks.

In-node processing time. Fig. 6 and Fig. 7 depict the computation overhead on the client- and the server-side for the different DTLS handshake variants. The certificate-based DTLS handshake incurs an overall processing overhead of about 6 s per end-point. This significant overhead primarily stems from the public-key operations required for the certificate-based peer authentication (i.e., 4.32 s) and the key agreement (i.e., 1.32 s) during the DTLS handshake. Combined, all public-key operations in the certificate-based DTLS handshake demand about 95 % of the overall processing time per end-point, thus dwarfing the remaining processing overheads of 0.05 s for the symmetric-key operations and of 0.25 s for the general packet processing (e.g., see message flights 4 and 5 in Fig. 6). The verification of the certificate chain in our evaluation scenario only requires a single signature verification, i.e., 1.9 s. Importantly, this certificate verification overhead grows linearly with number of intermediate certificates in a certificate chain.

In contrast, the purely symmetric-key-based DTLS handshake exhibits a considerably reduced processing overhead of about 0.18 s per peer. Our *delegation architecture* achieves the same low overhead. Notably, the verification and decryption of a session ticket (about 8.59 ms) and the generation of a new session ticket for a subsequent DTLS connection with the same end-point (about 11.08 ms) incur additional overheads compared to the symmetric-key-based DTLS handshake (e.g., see message flight 2 in Fig. 6). However, these additional overheads are compensated by the fewer DTLS messages and, consequently, the decreased packet processing cost for the omitted messages in the session resumption handshake. Overall, our proposed delegation architecture considerably reduces the processing overhead compared to the certificate-based DTLS handshake and achieves an equal processing efficiency to the symmetric-key-based DTLS handshake.

Transmission overhead. As shown in Fig. 8, our general observations for the in-node processing also apply to the measured transmission overheads of the different DTLS handshake variants. More precisely, the certificate-based DTLS handshake demands a total of 1609 bytes for the transmission of all handshake messages. In contrast, the symmetric-key-based handshake and our delegation architecture require transmissions of 458 bytes and 515 bytes, respectively.

Regarding the certificate-based handshake, all 15 messages of the DTLS protocol are needed for the connection establishment (see all messages in Fig. 2). When transmitted over size-constrained IEEE 802.15.4 radio links, these messages must additionally be split into 23 packet fragments due to their extensive message size. Such fragmentation has previously been shown to potential be harmfully [19] and is especially critical when considering the lossy nature of the wireless links. Specifically, with DTLS, already the loss of a single fragment of a message flight results in the retransmission of the entire flight. As each additional fragment of a flight adds to the overall probability of a lost packet and hence a retransmission, flights consisting of multiple fragments are undesirable. Still, the longest message flight of the certificate-based DTLS handshake, i.e., flight 4, must be split into 8 fragments. Moreover, as this flight contains certificate information, larger certificates or longer certificate chains than the ones used in our evaluation setup not only add to the immediate transmission cost, but also increase the non-negligible probability of retransmissions.

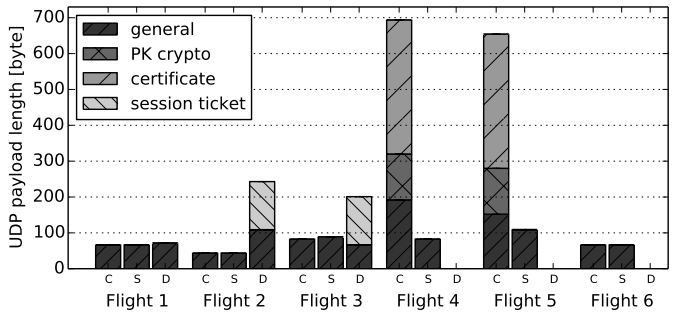

Fig. 8. Transmission overhead for the certificate-based DTLS handshake (C), the symmetric-key-based DTLS handshake (S) and our delegation architecture (D) on a per flight basis. Our delegation approach constitutes a similar transmission overhead as the symmetric-key-based handshake.

Contrarily, the symmetric key-based handshake allows to reduce the number of handshake messages to 10 (see unmarked messages in Fig. 2). As the individual message sizes also are considerably smaller than those of the certificate-based handshake, only 9 fragments need to be transmitted for the complete handshake. Notably, the `ServerHello` and `ServerHelloDone` messages fit into a single fragment. Our delegation architecture achieves the same reduction in message fragmentation (i.e., 9 message fragments) and thus incurs similar transmission overheads as the purely symmetric-key-based handshake. The major individual contributor for the transmission overhead of our delegation architecture are the two session tickets in the session resumption handshake, each of which has a size of 110 bytes. Hence, while exhibiting a slight increase in the absolute transmission overhead, our delegation architecture incurs the same amount of message forwarding overhead as the symmetric-key handshake.

To conclude, our proposed delegation architecture significantly outperforms the certificate-based DTLS handshake with respect to computation, memory, and transmission overheads. At the same time, it enables peer authentication and authorization across independent network domains. Finally, it only adds marginal memory requirements and achieves the same low computation and transmission overheads as the standard symmetric-key-based DTLS handshake.

VII. Related Work

For our discussion of related work, we distinguish three main research directions: i) alternative delegation and authorization approaches for the IP-based IoT, ii) related progress concerning cryptographic primitives, and iii) further IP security protocol improvements in the context of the IoT.

Several *delegation and authorization approaches* were recently presented that aim at offloading expensive IP security protocol operations to more powerful devices. The authors in [20], [21], [22] propose to delegate the public-key operations of DTLS, TLS, and IKEv2 to an on-path gateway, respectively. These approaches share the end-to-end security keys with the gateway. Thus, the authors effectively require gateways to be fully trusted and to be hardened against potential attacks. In contrast, our delegation architecture offloads the initial connection establishment to an off-path delegation server. Hence, our architecture does not require similar assumptions about the underlying infrastructure and supports the deployment of commodity hardware. Unlike our work, these approaches also do not consider revocation in their design. Saied et al. [23], [24], [25] propose to split the private key of the constrained device into multiple blocks. Proxies inside the constrained network domain then perform public-key operations based on their share of the private key. The proposed approaches, however, add a considerable number of network messages to the protocol handshake. In contrast, our delegation architecture decreases computation, memory, as well as transmission overheads. The Server-based Certificate Validation Protocol (SCVP) [26] enables a client to delegate certificate validation to a trusted server. Still, SCVP does not allow to offload other public-key-related operations of the certificate-based DTLS handshake. In [27], we previously presented initial ideas on delegating the certificate-based DTLS handshake via session resumption mechanisms. This paper extends on our previous work by contributing a significantly refined design and a detailed evaluation of our proposed delegation architecture.

Concerning *related progress of cryptographic primitives*, Hu et al. [28] show that public-key cryptography can be implemented and efficiently be used via dedicated hardware modules for constrained devices. Kothmayr et al. [29] subsequently present an implementation of the certificate-based DTLS handshake based on such hardware support. However, while allowing to decrease overheads that are directly attributable to the public-key operations, the certificate-based DTLS handshake involves significant additional overheads stemming from the protocol design itself and from the need for time synchronization and certificate status verification functionality. Importantly, our delegation architecture also allows to reduce these overheads and does not mandate special-purpose hardware for constrained devices. Implicit certificates [30] allow to decrease the transmission and verification overheads of traditional public-key certificates by super-imposing certificate elements. Still, implicit certificates do not alleviate the need for public-key cryptography on constrained devices. Garcia-Morchon et al. [31] propose a polynomial scheme as a replacement for public-key cryptography in the DTLS handshake. However, similar to the symmetric-key-based DTLS handshake, their approach depends on the secure provisioning of the polynomial shares before a DTLS connection can be established. Hence, they require a key provisioning mechanism similar to our delegation architecture for secure communication across independent network domains.

Further IP security protocol improvements for the IoT predominantly focus on adapting the protocol run-time properties to the special characteristics of constrained devices. To decrease handshake transmissions, the proposed Cached Information Extension for TLS [7] enables a client to cache static server information and to omit this information during subsequent handshakes. In [32], we previously presented a header compression mechanism for DTLS. Complementary to our work here, such mechanisms allow to further decrease the transmission overhead of our proposed delegation architecture. Moreover, in [6], we introduced comprehensive session resumption, denial-of-service protection, and retransmission mechanisms for DTLS, HIP DEX, and minimal IKEv2 that are tailored towards constrained network environments. Especially, the proposed session resumption functionality provides the

necessary protocol mechanisms to also enable our delegation architecture for HIP DEX and minimal IKEv2.

VIII. Conclusion

In this paper, we analyzed the impact of public-key cryptography on the certificate-based DTLS handshake and identified significant RAM and ROM requirements. These requirements render the certificate-based handshake infeasible for a wide range of constrained devices. To still enable secure communication between constrained devices and remote end-points, we proposed a delegation architecture that allows to separate the initial DTLS connection setup from the subsequent protection of application data. This delegation architecture is based on the introduction of a delegation server. By leveraging the central role of the delegation server in the initial connection establishment, our delegation architecture also allows to authorize connections inside a local network domain as well as between a constrained device and a remote end-point.

As the evaluation results show, our delegation architecture nearly achieves a three-fold reduction of the memory requirements compared to the certificate-based DTLS handshake. Moreover, our architecture only incurs marginally increased memory overheads considering the symmetric-key-based DTLS handshake. Importantly, we achieve the same low computation and transmission overheads as a purely symmetric-key-based handshake. Hence, in conclusion, our proposed delegation architecture provides a comprehensive, yet compact solution for authentication, authorization, and secure data transmission for the IP-based IoT.

Acknowledgments

The authors would like to thank Christoph Walser and Niclas Finne for their valuable support. This work was partly funded by the German Federal Ministry of Economics and Technology under the project funding reference number 01MD11049 and the Swedish Foundation for Strategic Research (SSF). The responsibility for the content of this publication lies with the authors.

References

[1] S.-J. Jung and W.-Y. Chung, "Non-Intrusive Healthcare System in Global Machine-to-Machine Networks," in *Proc. of IEEE Sensors Journal, (Volume:13, Issue:12)*, 2013.

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

[3] R. Moskowitz and R. Hummen, "HIP Diet EXchange (DEX)," draft-moskowitz-hip-dex-01 (WiP), IETF, 2012.

[4] T. Kivinen, "Minimal IKEv2," draft-kivinen-ipsecme-ikev2-minimal-01 (WiP), IETF, 2012.

[5] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks," in *Proc. of IEEE PerCom*, 2005.

[6] R. Hummen, H. Wirtz, J. H. Ziegeldorf, J. Hiller, and K. Wehrle, "Tailoring End-to-End IP Security Protocols to the Internet of Things," in *Proc. of IEEE ICNP*, 2013.

[7] S. Santesson and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension," draft-ietf-tls-cached-info-16 (WiP), IETF, 2014.

[8] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944, IETF, 2007.

[9] M. Sethi, J. Arkko, A. Keranen, and H. Rissanen, "Practical Considerations and Implementation Experiences in Securing Smart Object Networks," draft-aks-crypto-sensors-02 (WiP), IETF, 2012.

[10] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *Proc. of ACM/IEEE IPSN*, 2005.

[11] F. Stajano and R. Anderson, "The resurrecting duckling: Security issues for ad-hoc wireless networks," in *Proc. of Security Protocols*, 1999.

[12] C. Kuo, M. Luk, R. Negi, and A. Perrig, "Message-in-a-bottle: User-friendly and secure key deployment for sensor nodes," in *Proc. of ACM SenSys*, 2007.

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

[14] R. Hummen, J. Gilger, and H. Shafagh, "Extended DTLS Session Resumption for Constrained Network Environments," draft-hummen-dtls-extended-session-resumption-01 (WiP), IETF, 2013.

[15] Arago Systems, "WiSMote," http://www.aragosystems.com/images/stories/WiSMote/Doc/wismote_en.pdf, [Online, last accessed: 08.04.2014].

[16] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proc. of IEEE LCN*, 2004.

[17] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel, "FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems," in *Proc. of ACM/IEEE IPSN*, 2013.

[18] Z. Shelby, K. Hartke, and C. Bormann, "Constrained Application Protocol (CoAP)," draft-ietf-core-coap-18 (WiP), IETF, 2013.

[19] 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.

[20] J. Granjal, E. Monteiro, and J. S. Silva, "End-to-end transport-layer security for Internet-integrated sensing applications with mutual and delegated ECC public-key authentication," in *Proc. of IFIP Networking*, 2013.

[21] S. Fouladgar, B. Mainaud, K. Masmoudi, and H. Afifis, "Tiny 3-TLS: A trust delegation protocol for wireless sensor networks," in *Proc. of Security and Privacy in Ad-Hoc and Sensor Networks*, 2006.

[22] 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 *In Proc. of IEEE WoWMoM*, 2012.

[23] Y. B. Saied and A. Olivereau, "D-HIP: A distributed key exchange scheme for HIP-based Internet of Things," in *Proc. of IEEE WoWMoM*, 2012.

[24] Y. B. Saied and A. Olivereau, "(k, n) threshold distributed key exchange for HIP based internet of things," in *Proc. of ACM MobiWac*, 2012.

[25] Y. B. Saied and A. Olivereau, "HIP Tiny Exchange (TEX): A distributed key exchange scheme for HIP-based Internet of Things," in *Proc. of ComNet*, 2012.

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

[27] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle, "Towards Viable Certificate-based Authentication for the Internet of Things," in *Proc. of ACM HotWiSec*, 2013.

[28] W. Hu, P. Corke, W. Shih, and L. Overs, "secfleck: A public key technology platform for wireless sensor networks," in *Proc. of EWSN*, 2009.

[29] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, "DTLS based security and two-way authentication for the Internet of Things," *in Ad Hoc Networks (Volume:11, Issue:8)*, 2013.

[30] G. Zaverucha, "Standards for Efficient Cryptography 4: Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV), Working Draft," Certicom Research, March 2011.

[31] O. Garcia-Morchon, S. L. Keoh, S. Kumar, P. Moreno-Sanchez, F. Vidal-Meca, and J. H. Ziegeldorf, "Securing the IP-based internet of things with HIP and DTLS," in *Proc. of ACM WiSec*, 2013.

[32] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight Secure CoAP for the Internet of Things," in *IEEE Sensors Journal (Volume:13, Issue:10)*, 2013.