Tailoring End-to-End IP Security Protocols to the Internet of Things

René Hummen, Hanno Wirtz, Jan Henrik Ziegeldorf, Jens Hiller, Klaus Wehrle
Communication and Distributed Systems, RWTH Aachen University, Germany
Email: {lastname}@comsys.rwth-aachen.de

***Abstract*—Recent standardization efforts focus on a number of lightweight IP security protocol variants for end-to-end security in the Internet of Things (IoT), most notably DTLS, HIP DEX, and minimal IKEv2. These protocol variants commonly consider public-key-based cryptographic primitives in their protocol design for peer authentication and key agreement. In this paper, we identify several performance and security issues that originate from these public-key-based operations on resource-constrained IoT devices. To illustrate their impact, we additionally quantify these protocol limitations for HIP DEX. Most importantly, we find that public-key-based operations significantly hamper a peer's availability and response time during the protocol handshake. Hence, IP security protocols in the IoT must be tailored to reduce the need for expensive cryptographic operations, to protect resource-constrained peers against DoS attacks targeting these cryptographic operations, and to account for high message processing times. To this end, we present three complementary, lightweight protocol extensions for HIP DEX: i) a comprehensive session resumption mechanism, ii) a collaborative puzzle-based DoS protection mechanism, and iii) a refined retransmission mechanism. Our focus on common protocol functionality allows to generalize our proposed extensions to the wider scope of DTLS and IKE. Finally, our evaluation confirms the considerable achieved improvements at modest trade-offs.**

I. Introduction

Standardization bodies including ETSI, the IETF, and the ZigBee Alliance are heavily pushing towards IP technology for transparent end-to-end connectivity between constrained devices and services in the Internet of Things (IoT). To secure these end-to-end connections, focus has shifted towards lightweight variants of existing IP security protocols, most notably Datagram TLS (DTLS) [1], the HIP Diet EXchange (DEX) [2], and minimal IKEv2 [3]. While DTLS optionally provides a pre-shared key-based handshake, HIP DEX and minimal IKEv2 mandate public-key cryptography in their protocol design. Although public-key cryptography is sensible specifically for peer authentication across administrative domains, its use renders the application of these protocols in constrained network environments challenging.

We identify three major challenges in the handshake phase of the considered protocols that directly stem from the use of public-key cryptography and strongly hinder the protocols' applicability in the IoT. First, the use of public-key primitives in the protocol handshakes requires a significant amount of transmissions and computation time. If constrained devices are to afford this overhead regularly, sleep deprivation and power-intensive radio usage decrease the lifetime of energy-constrained devices. More importantly, the devices are unable to fulfill their original tasks, e.g., processing of sensed information or packet forwarding, while the CPU is busy performing cryptographic operations. Second, a *single* malicious host can misuse the handshake protocols to mount Denial of Service (DoS) attacks that target the constrained resources of an IoT device, e.g., by performing multiple handshakes in short succession. The standardized DoS protection mechanisms of IP security protocols, however, do not suffice to defend against such attacks [4]. Third, retransmissions of handshake messages, including those that cause expensive operations, commonly follow fixed timeout-based approaches. Hence, these mechanism do not account for the processing time of said operations and commonly cause premature retransmissions.

In this paper, we detail the above challenges in constrained network environments and quantify their impact for one of the candidate protocols. We choose HIP DEX for our analysis as it already features a concise handshake that has specifically been designed for constrained devices. Hence, it allows us to isolate the impact of public-key-based operations on a protocol handshake. Based on our analysis, we propose three complementary and lightweight protocol extensions for HIP DEX. Addressing the above challenges respectively, we propose i) a *comprehensive session resumption mechanism* to reduce the computation, memory, and transmission costs of the protocol handshake, ii) a *collaborative puzzle-based Denial of Service protection mechanism* that accounts for device and network heterogeneity, and iii) a *refined retransmission mechanism* that takes the varying processing times of handshake messages into account. Notably, we show that our protocol extensions generalize to DTLS and minimal IKEv2. Our evaluation confirms the considerable improvements of our extensions compared to an unmodified HIP DEX at modest memory tradeoffs.

This paper is structured as follows. Section II introduces the network scenario and gives a brief overview of the HIP DEX protocol. We then discuss our identified protocol performance and security issues with respect to constrained network environments in Section III. In Section IV, we present our proposed protocol extensions that further tailor the HIP DEX protocol to the characteristics of IoT network scenarios. Here, we additionally highlight how these extensions can be integrated in the IP security protocols DTLS and minimal IKEv2. We then discuss the security considerations of our proposed protocol extensions in Section V and present our evaluation results in Section VI. Finally, Section VII discusses related work and Section VIII concludes our paper.

II. Prerequisites

We now briefly present our abstract network scenario and the HIP DEX protocol as the basis for our analysis.

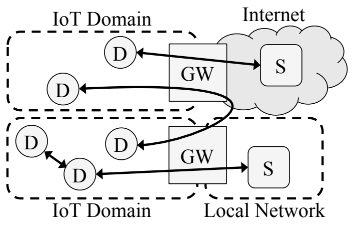

Fig. 1. Resource-constrained devices (D) communicate with each other and with local or Internet-based services (S) via a gateway (GW). Arrows indicate communication paths for specific security protocol handshakes.

A. Network Scenario

Figure 1 illustrates our abstract network scenario consisting of constrained devices in the IoT domain, services that are located in the local network or the Internet, and gateways that interconnect the different network domains. Constrained devices are equipped with only a few MHz of computational power, several kilobytes of RAM and several *tens* of kilobytes of ROM. Moreover, they may be battery-powered and are assumed to communicate over lossy wireless links, e.g., as provided by IEEE 802.15.4. Constrained devices can initiate new connections with other constrained devices or services, or they may respond to incoming connection requests. Gateways range from commodity routers to dedicated servers and connect the IoT domain to the Internet via a broadband connection. Services are assumed to run on common server hardware.

B. HIP DEX

The HIP Diet EXchange (DEX) [2] is a key management protocol that provides secure end-to-end connections in the IoT. It is currently being standardized at the IETF as a protocol variant of the Host Identity Protocol (HIP) [5]. HIP DEX preserves the general HIP architecture and protocol semantics. In particular, it inherits the cryptographic namespace that uses the public key of a device as its Host Identity (HI). This namespace is used to build a new layer in the network stack that is located between the network and the transport layer. A cryptographic fixed-length representation of the HI, the Host Identity Tag (HIT), serves as a stable device identifier at this layer. This allows to reduce the role of IP addresses to changeable locators and affords verifiable device identification as well as device mobility as additional protocol features.

From HIP, HIP DEX additionally adopts the out-of-band notification message type that enables the peers to communicate auxiliary information such as protocol errors or negotiation failures during the defined message exchanges. Finally, HIP DEX inherits the session lifecycle from HIP that begins with the session establishment handshake and concludes with the session tear down exchange. This enables the peers to explicitly close a connection and to flush all session-related protocol state at the end of the session lifecycle.

Notably, HIP DEX introduces two key differences to standard HIP that account for the characteristics of constrained network environments. First, it defines an *aggressive retransmission mechanism* to handle packet loss in constrained wireless network environments. This retransmission mechanism requires the Initiator to continually send I1 and I2 packets at a short interval in the order of milliseconds. The Responder then must reply to each Initiator-side packet. Second, HIP

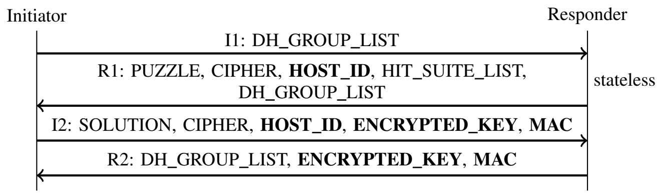
Initiator
Responder
I1: DH_GROUP_LIST
R1: PUZZLE, CIPHER, HOST_ID, HIT_SUITE_LIST, DH_GROUP_LIST
stateless
I2: SOLUTION, CIPHER, HOST_ID, ENCRYPTED_KEY, MAC
R2: DH_GROUP_LIST, ENCRYPTED_KEY, MAC

Fig. 2. Protocol diagram of the refined HIP DEX handshake.

DEX forfeits ephemeral Diffie-Hellman (DH) keys and digital signatures in its protocol design. Instead, it defines a *refined session establishment handshake* that is based on static DH keys for mutual peer authentication and key agreement.

As shown in Fig. 2, the refined session establishment handshake consists of a four-way handshake between an *Initiator* and a *Responder*. The Initiator triggers the handshake with an I1 message. The next messages implement a standard authenticated DH key agreement. Specifically, the peers exchange their public *static* DH keys in the HOST_ID parameters of the R1 and the I2 messages and derive a shared symmetric *Master Key*. To authenticate this key agreement, both peers prove the correctness of the Master Key using message authentication codes (MAC) in the I2 and R2 messages.

The peers additionally exchange secrets in the ENCRYPTED_KEY parameter of the I2 and R2 messages in order to generate a fresh session key for the protection of application data (e.g., via IPsec) in each handshake. These secrets are encrypted with the Master Key and thus are concealed from on-path network entities. The peers then derive the payload protection key based on the exchanged secrets.

To protect the expensive cryptographic operations and the state creation during the protocol handshake, HIP DEX employs a DoS protection mechanism with dynamically adjustable, cryptographic puzzles, similar to HIP. The Responder thereby chooses a random puzzle challenge I and a difficulty K. It includes these values in the PUZZLE parameter of the R1 message (see Fig. 2). When receiving a puzzle, the Initiator has to find a solution J such that an AES-based CMAC operation over the concatenation of the challenge I, the peers' HITs, and the solution J generates an output where at least the K lowest order bits are zero. The Initiator then responds I, J, and K in the SOLUTION parameter of the I2 message. Finally, the Responder verifies the puzzle with a *single* CMAC operation.

III. Public-key-related Protocol Design Issues

In this section, we identify performance and security issues in end-to-end IP security protocols that apply public-key cryptography in constrained environments. We specifically quantify impacts for HIP DEX and structure our discussion along the following three aspects: i) the choice and strength of the cryptographic primitives, ii) the consideration of resource heterogeneity in the provided DoS protection mechanisms, and iii) the suitability of the packet retransmission strategies.

A. Cryptographic Primitives

Given a concise protocol handshake, the resource demands of public-key primitives remain as the determining factor for the applicability of IP security protocols on constrained devices. Specifically, expensive handshake operations block the CPU, preventing the device from sleeping to save energy or processing other network packets or sensor readings.

HIP DEX forfeits the use of public-key signatures and ephemeral DH keys to reduce the computation overhead. Instead, it directly uses *static* DH keys for peer authentication and key agreement purposes. The handshake thus only involves a single cryptographic operation per peer instead of four, namely two for signature generation and verification and two for ephemeral DH key generation and key agreement. Hence, the protocol design trades the perfect forward secrecy and the non-repudiation properties of HIP for a significantly decreased protocol handshake overhead. Our evaluation shows that HIP DEX thereby reduces the handshake run-time by about 3.4 s[1].

Protocol issue: Already when using the smallest elliptic curve defined for HIP DEX (i.e., SECP160R1), a single DH operation dominates the handshake run-time with a processing time of about 0.7 s per peer. This overhead will further increase in the near future as larger curves of at least 224 bits are recommended by NIST to protect sensitive information after 2013 [6]. The smallest curve defined for HIP DEX satisfying this requirement is NIST P-256. However, this curve incurs a considerably higher computational overhead of about 1.9 s per peer. Hence, the need for public-key-based operations in the handshake of IP security protocols must further be reduced in order to preserve device resources for the actual use case.

B. DoS Protection

Highly diverse device and network resources in IoT scenarios, i.e., memory, processing power, and bandwidth, enable a *single* unconstrained adversary such as an Internet host to mount DoS attacks against the IP security protocol handshake on a constrained device. Specifically, an adversary can initiate multiple parallel handshakes and misuse expensive protocol operations in order to exhaust computation and memory resources. Likewise, handshakes initiated in short succession continually occupy these resources at the target device.

Predominantly, two protocol mechanisms are used to counter DoS attacks against the protocol handshake. DTLS and IKE use a *cookie mechanism* that requires the Initiator of a handshake to echo a Responder-defined nonce before the Responder invests computation or memory resources. This allows to blacklist malicious nodes if they can be uniquely identified by their IP address. However, IPv6-based communication, as envisioned for IoT scenarios, affords individual hosts to be multi-addressed or to change addresses over time, e.g., for privacy reasons [7]. As a result, a single attacker can pretend to represent multiple hosts and thus thwart cookie-based DoS protection mechanisms. *Puzzle-based mechanisms* as used by HIP DEX thus employ a different technique. By asking the Initiator to solve a cryptographic puzzle, the Responder can demand *adjustable* resource commitments before processing expensive handshake messages. Hence, the Responder can render the connection establishment expensive for a malicious Initiator that initiates handshakes in parallel or in succession.

Protocol issue: Puzzle-based DoS protection mechanisms invariably impact legitimate Initiators that aim to set up a connection with a Responder that is currently under attack. More precisely, the Responder requires the same resource commitments from legitimate and malicious Initiators since it cannot distinguish them during the handshake. High puzzle difficulties thereby are uncritical for unconstrained legitimate Initiators, but quickly become impractical to solve for constrained Initiators, thus causing handshakes to fail. Vice versa, low puzzle difficulties account for constrained legitimate Initiators but do not protect against an unconstrained adversary. Hence, DoS protection in the IoT needs to require a small resource commitment from constrained devices, while demanding a high resource commitment from unconstrained devices. However, IP addresses, host identities, and cipher suites do not carry sufficient semantic meaning about the peer to make an informed decision regarding the puzzle difficulty.

C. Retransmission Strategies

Protocols commonly define retransmission strategies to handle corrupt or lost packets. These strategies aim for the opposing goals of minimizing premature retransmissions as well as reducing handshake delay caused by packet loss. Especially in constrained network environments, premature retransmissions are highly undesirable because they waste energy resources on the forwarding path and occupy the wireless medium. However, transmission and processing times of individual messages for IP security protocols may vary significantly depending on device capabilities. For example, our evaluation shows that an ECDH operation on a desktop-class device only requires about 15 ms compared to 656 ms on a constrained device. As a result, response messages that also signal reception acknowledgement may be received with non-negligible delays if they involve expensive operations.

Protocol issue: IP security protocols commonly specify and implement fixed timeout-based retransmission mechanisms that do not account for the processing time of public-key operations. HIP DEX specifies an aggressive strategy that triggers retransmissions at time intervals in the order of milliseconds. However, public key operations may take seconds to complete on constrained devices. Hence, overly aggressive strategies inevitably cause retransmissions despite successful packet reception. In contrast, DTLS recommends a timeout of 1 sec that is to be doubled at each retransmission. While this strategy provokes less premature retransmissions, it quickly delays the protocol handshake if consecutive retransmissions are lost. Hence, adaptive retransmission strategies are required that take the message processing time at the peer into account.

IV. Tailoring HIP DEX to the IoT

We now present our three complementary and lightweight protocol extensions for HIP DEX that aim to alleviate the identified protocol limitations. First, we amortize expensive public-key-based operations of an initial protocol handshake across multiple connections with a comprehensive session resumption mechanism (Section IV-A). Second, we extend the HIP DEX puzzle mechanism to allow for collaborative DoS protection in network scenarios with high resource heterogeneity (Section IV-B). Finally, we propose an adaptive retransmission mechanism that accounts for the varying processing time of handshake operations on constrained devices (Section IV-C).

[1]We refer to Section VI for detailed information about our evaluation setup.

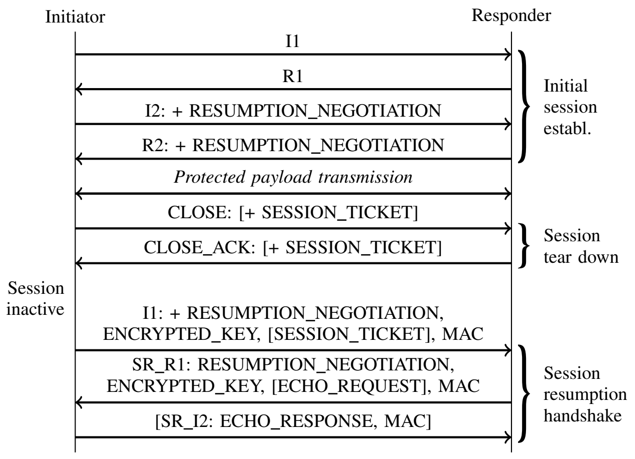
Initiator
Responder
I1
R1
I2: + RESUMPTION_NEGOTIATION
R2: + RESUMPTION_NEGOTIATION
Initial session establ.
Protected payload transmission
CLOSE: [+ SESSION_TICKET]
CLOSE_ACK: [+ SESSION_TICKET]
Session tear down
Session inactive
I1: + RESUMPTION_NEGOTIATION, ENCRYPTED_KEY, [SESSION_TICKET], MAC
SR_R1: RESUMPTION_NEGOTIATION, ENCRYPTED_KEY, [ECHO_REQUEST], MAC
[SR_I2: ECHO_RESPONSE, MAC]
Session resumption handshake

Fig. 3. Initial session establishment and abbreviated session resumption handshake. State can be offloaded during the session tear down. Brackets denote optional messages and parameters that depend on the session resumption type.

A. HIP DEX Session Resumption

Our main goal in this section is to amortize the cost of the DH key agreement in the HIP DEX handshake across multiple connections. We therefore propose a novel session resumption mechanism for HIP DEX that is inspired by similar mechanisms in TLS [8] and IKE [9]. The basic idea of session resumption is that peers only perform expensive protocol operations once during the *initial session establishment*. The storage of session state after the session teardown then enables efficient re-authentication and re-establishment of the secure payload channel in an *abbreviated session resumption handshake*. We also incorporate the Responder-side state-offloading session resumption of DTLS [10] and IKE [9] in our design.

In contrast to the existing mechanisms, we specifically tailor our approach to the constraints of IoT scenarios. More precisely, we propose a flexible session resumption mechanism that features an additional state-offloading type for constrained Initiators and an *explicit* negotiation of the session resumption type. Moreover, by delaying state-offloading until the session tear down, we enable constrained devices to store limited session state for their peers. Most importantly, our design aims at minimizing memory requirements of inactive sessions as well as reducing transmissions for the connection re-establishment.

Initial session establishment. As shown in Fig. 3, the communicating peers negotiate the session resumption type during the initial HIP DEX handshake. We thereby consider three possible types: i) *state compression*, where both peers maintain their own state across connections, ii) *Initiator-side state-offloading*, where the Responder stores the session state on behalf of the Initiator, and iii) *Responder-side state-offloading*, where the Initiator stores the session state on behalf of the Responder. The Initiator signals its supported session resumption types to the Responder in the RESUMPTION_NEGOTIATION parameter of the I2 message. The Responder then indicates the selected type in the RESUMPTION_NEGOTIATION parameter of the R2 message. To support legacy peers that do not support our extension, no session resumption is performed if at least one peer does *not* include the negotiation parameter in the handshake. The handshake then concludes as defined for the standard protocol.

If the peers agree on Initiator-side state-offloading, the Initiator compresses and encrypts its active session state and includes this state in the SESSION_TICKET parameter of the CLOSE message. It then discards the active session state similar to a standard session tear down. On reception of the CLOSE message, the Responder compresses its own state and stores it along with the received offloaded state. With Responder-side state-offloading, the Responder includes its session state in the SESSION_TICKET parameter of the CLOSE_ACK message. If neither peer is willing to store offloaded session state in addition to its own compressed state, the peers can still agree on session resumption using their respective compressed state by setting this resumption type in the RESUMPTION_NEGOTIATION parameter. In this case, no session tickets are transferred in the tear down exchange.

We intentionally delay state-offloading until the end of the session lifecycle to relieve peers from storing offloaded state while the session is still active. This design trait enables even constrained devices to store limited session state for constrained peers as the compressed and offloaded state combined is smaller than the state of an active session (see Section VI-A). Still, a peer may temporarily loose connectivity, thus preventing the negotiated state-offloading. In this case, the peers fall back to session resumption with state compression to re-establish the session independent of the negotiated type.

Abbreviated session resumption handshake. The abbreviated session resumption handshake consists of three cases that depend on which peer stores session state and which peer triggers the new connection establishment. We first describe the abbreviated handshake with state compression and detail the handshakes with state-offloading afterwards.

The session resumption handshake with *state compression* consists of the mandatory messages and parameters of the second handshake depicted in Fig. 3. The Initiator indicates this resumption type in the RESUMPTION_NEGOTIATION parameter of the I1 message. Moreover, the I1 message contains a fresh, encrypted secret for payload protection and a MAC. The encryption and the MAC operation are both based on the Master Key derived from the stored session state.

When receiving the I1 message, the Responder inspects the RESUMPTION_NEGOTIATION parameter and retrieves its own compressed state. To validate the message integrity and to authenticate the Initiator, it re-computes the Master Key and verifies the MAC. If this verification is successful, the Responder concludes the handshake with a new SR_R1 message that confirms the resumption type and includes a new secret for payload protection. The additional MAC parameter enables the Initiator to verify the message integrity and to authenticate the Responder. Note that the standard R1 message contains mandatory parameters that are not required in the abbreviated handshake, e.g., the peer's public key. We thus use a dedicated, tailored message type to reduce transmissions.

For *Responder-side state-offloading*, the Initiator additionally includes the Responder's session state in the SESSION_TICKET parameter of the I1 message. Upon reception, the Responder verifies the correctness of the received state and re-computes the Master Key to verify the included MAC parameter. Notably, the Responder only creates session state if the prior verifications were successful. The Responder then generates an SR_R1 response and includes an ECHO_REQUEST parameter to determine the freshness of the abbreviated handshake. The ECHO_REQUEST parameter requires the Initiator to echo the received challenge value in the ECHO_RESPONSE parameter of the SR_I2 message. The successful verification of the challenge and of the MAC in the SR_I2 message assures the Responder of the handshake freshness because only the legitimate Initiator can compute correct MACs for messages with Responder-specified content.

Finally, for session resumption with *Initiator-side state-offloading*, the Initiator sends a standard I1 message as it is unaware of the previous session. When receiving this message, the Responder looks up its own compressed state for the previous connection as well as the Initiator's offloaded state. If such state exists, the Responder assumes the role of the Initiator and sends an I1 message as in session resumption with Responder-side state offloading. The original Initiator changes its role and the handshake concludes as described above.

Required session state information. While session state is implementation and scenario-specific, it must meet a number of requirements to ensure a secure session resumption. Specifically, it has to include the peer's HIT as its cryptographic identifier. Otherwise, an adversary would be able to impersonate a peer by using the peer's HIT in an abbreviated session resumption handshake with state offloading. Furthermore, session state needs to include the agreed DH key and the negotiated cipher suite of the first handshake in order to re-compute the Master Key and to encrypt signaling information.

Most importantly, offloaded state must be encrypted and integrity protected. The corresponding key should only be known to the offloading peer. Possession of this key would allow an adversary to create forged encrypted states and to impersonate other devices towards the offloading peer. We propose the use of AES CCM for encryption and integrity protection, thus allowing to leverage potential AES hardware support on IoT devices. Finally, ticket invalidation, e.g., due to node compromise or de-authorization of a peer, may be achieved by changing the state encryption key.

Integration in DTLS and minimal IKEv2. The presented *explicit* session resumption type negotiation and session resumption with Initiator-side state-offloading can also be integrated with the existing mechanisms for DTLS and minimal IKEv2. As we specify in an IETF companion document [11], the session resumption negotiation may be included in DTLS as an extension of the ClientHello and the ServerHello handshake messages. Moreover, the standardized session resumption behavior can remain unchanged for state compression or Responder-side state-offloading. For Initiator-side state-offloading, the Initiator may re-use the NewSessionTicket message to transfer its session state. To resume a session, the Responder can then send the session state to the Initiator in the SessionTicket extension of the ServerHello message.

For minimal IKEv2, our session resumption negotiation may occur in the first two messages of the IKE handshake (i.e., the IKE_SA_INIT exchange). Similar to DTLS, the protocol can stay unchanged for state compression and Responder-side state-offloading. For Initiator-side state-offloading, the Initiator can offload session state in the ticket notification payload in the first message of the IKE_AUTH exchange. To resume the session, the stateless Initiator first sends a regular IKE_SA_INIT message as it is unaware of the Responder's support for the IKE session resumption mechanism. This message additionally contains a notification payload informing about the Initiator's session resumption support. The Responder then replies with an IKE_SA_INIT response that includes a TICKET_ACK notification payload. As a result, the Initiator starts the existing IKE_SESSION_RESUME exchange and the Responder transfers the session state in the response message.

B. Collaborative Puzzle-based DoS Protection for the IoT

In this section, we tailor the HIP DEX puzzle mechanism to constrained environments. Specifically, the HIP DEX specification recommends a puzzle difficulty of zero during regular network operation and a non-trivial puzzle difficulty during a DoS attack against the handshake. However, mechanisms to detect attacks and to select an appropriate puzzle difficulty that protects IoT devices remain open issues. To fill this gap, we propose a simple *attack detection and difficulty selection strategy* based on a sliding window mechanism. We propose a complementary handshake extension that allows an on-path gateway to *collaborate* in the puzzle difficulty selection to account for device and network heterogeneity. For further information about specific handshake and parameter structures, we refer to a second IETF companion document [12].

Attack detection and difficulty selection strategy. To detect attacks against the expensive DH operation or the state creation process, we propose that Responders maintain a *sliding window* that counts the number of performed DH operations within a fixed-length window period of, e.g., 1 min. Within this window period, Responders allow for a configurable number of legitimate handshakes and set the puzzle difficulty for these handshakes to zero. We represent this number by a device and scenario-specific *puzzle issuing threshold*. If this threshold is exceeded, Responders demand non-zero puzzle difficulties. Issuing puzzles based on the number of DH operations also covers attacks targeting scarce memory resources as the DH operation precedes the state creation process.

When issuing a puzzle with a non-zero difficulty, a Responder strives to push expensive operations to the next window period if its current threshold is exceeded. Thus, it sets the puzzle difficulty for the Initiator to require approximately one window period to solve the puzzle. To derive this difficulty value, we propose that a resource-constrained Responder bases the difficulty selection on its own resources as a first indicator of the peer's capabilities. A resource-constrained Responder then issues puzzles that protect against other resource-constrained devices, e.g., a malicious Initiator that is located in the same IoT domain. However, these puzzles do not defend against attacks from unconstrained peers outside the IoT domain.

Collaborative difficulty selection. To also consider unconstrained peers in the difficulty selection, we observe that an on-path gateway commonly has further information about the peer when a handshake crosses network boundaries. Specifically, the gateway can distinguish handshakes as originating from a local trusted network or an unknown and thus untrusted Internet hosts based on the switching port that the handshake was received on. Moreover, authenticated tunnels to the gateway allow to classify remote devices or networks as benign. Thus, we extend the HIP DEX handshake with a notification mechanism that enables a gateway to add a new VIA_UNTRUSTED_NETWORK parameter to a traversing I1 message if the peer does not belong to a trusted network domain. If this parameter is set, the Responder issues a high puzzle difficulty. Otherwise, it sets a low puzzle difficulty, assuming the peer to be located in a trusted domain. Thus, the impact of a DoS attack from an untrusted network on legitimate handshakes decreases significantly.

If the resource-constrained device constitutes a legitimate, but *untrusted* Initiator for a Responder that is located in a foreign network domain, the Responder issues a high puzzle difficulty. As a result, handshakes would not conclude successfully. However, if the Responder also implements our proposed attack detection and difficulty selection strategy, the peers can still successfully complete the handshake with a moderate delay. To this end, the Initiator drops a puzzle that it cannot solve in a timely fashion and requests a new puzzle from the Responder, thus effectively probing the Responder for a window period with low load and consequently a puzzle difficulty of zero. These periods typically occur during an attack when the adversary is busy solving a puzzle. Still, as an adversary can behave similar to the Initiator, this puzzle solving strategy denotes a race for a low threshold value.

Integration in DTLS and minimal IKEv2. DTLS and minimal IKEv2 both specify the use of a cookie mechanism to protect their protocol handshakes against DoS attacks. We propose to replace these cookie mechanisms with a cryptographic puzzle and to use the provided cookie as the puzzle challenge. An on-path gateway then can inspect the unprotected, initial handshake messages and assist with the puzzle difficulty selection similarly to our proposed extension for HIP DEX. As the main effort for this integration, we identify the additional specification of a parameter that conveys the puzzle solution.

C. Retransmission Mechanism Refinements

In this section, we first discuss the disadvantages of an aggressive retransmission strategy in constrained network scenarios, e.g., as proposed by HIP DEX. We then present our *adaptive* retransmission mechanism that employs multiple worst-case estimates for the retransmission timeout.

With HIP DEX, the Initiator is responsible for the retransmission of lost I1 and R1 messages (see Figure 2), while the Responder simply replies to each received I1 message. HIP DEX thereby specifies aggressive retransmissions with a fixed timeout value in the order of milliseconds, regardless of the network topology or the current network conditions. However, a fixed timeout that is lower than the round-trip time in situations with high network load inevitably causes premature I1 retransmissions by the Initiator and generates unneeded R1 responses by the Responder. This negative effect is further aggravated by the fact that R1 messages have to be fragmented into several link layer packets, thus placing an increased burden on the network. Hence, in contrast to the aggressive retransmission strategy in HIP DEX, we adopt the more conservative strategy of HIP [5] for I1 and R1 messages and base the timeout value on the *worst-case anticipated round-trip time*. This timeout can, for example, be derived based on the maximum RTT of *ping* messages. As the Responder does not perform expensive operations during the initial part of the handshake, it suffices to consider this worst-case network delay in a retransmission strategy for I1 and R1 messages.

The fragmentation of R1 messages necessitates an additional adaptation of the retransmission mechanism. Specifically, the Initiator may already have received *parts* of the R1 message before the retransmission timeout expires. If this part contains HIP DEX header information, the Initiator can correlate it to the corresponding handshake and infer that its original I1 message has been received correctly. Hence, the remaining R1 response may still be in transit. To exploit this fact, we propose to *pre-fetch* message information of incomplete messages from the lower protocol layers before triggering an I1 retransmission. This allows to delay the retransmission and to wait for further message fragments to arrive. Pre-fetching information across layers is typically feasible on constrained devices as their implementations often use cross-layer interactions for efficiency reasons. We also employ pre-fetching for later messages in the protocol exchange that exceed the size of a single frame. Note that additional protection mechanisms must be applied at the lower layers to defend against attacks targeting HIP DEX packet fragmentation [13].

In contrast to I1 and R1 messages, the cryptographic processing costs of I2 messages at the Responder may be significant. Hence, I2 messages inevitably cause retransmissions if they exceed small fixed or network delay-based timeouts. Thus, the Initiator has to take processing cost into account. We propose to *split the dual role* of R2 messages, i.e., information carrier and acknowledgment (ACK). To this end, the Responder sends a NOTIFY message to the Initiator *before* it commences any cryptographic processing. This message serves as an ACK for the I2 message reception. *After* the I2 message processing has finished, the Responder additionally sends the regular R2 response message. As a result, the Initiator can first base its I2 retransmissions on the *worst-case anticipated round-trip time* similar to I1 and R1 messages. Once the NOTIFY message has been received, the Initiator starts a second timeout that is based on the *worst-case anticipated computation time* of an I2 message at a resource-constrained Responder. For example, the evaluation data presented in this paper can be used as an indicator for this timeout value.

In conclusion, our proposed NOTIFY message allows to use a network delay-based timeout for retransmissions of a lost I2 message, while employing a second processing-based timeout for lost R2 messages. However, NOTIFY messages increase transmission costs and may be lost as well. Thus, unconstrained Responders should omit NOTIFY messages as they are able to quickly reply to all handshake messages.

Integration in DTLS and minimal IKEv2. For the adoption in DTLS, an Alert message can be used to acknowledge message reception before performing expensive handshake operations. As DTLS performs retransmissions per flight, this Alert message should also contain the sequence numbers of already received flight messages. On receipt, the node deduces that its peer currently performs an expensive protocol operation and only needs to retransmit missing flight messages in case the worst-case anticipated computation time is exceeded.

Our proposed ACK signaling does not directly translate to minimal IKEv2 as out-of-band notification messages during the first two protocol messages are specified to cause a handshake failure. Yet, the Responder is required to perform an expensive DH operation during this protocol phase. If this restriction was lifted during the standardization process of minimal IKEv2, out-of-band notifications could be used in addition to network delay-based timeouts and pre-fetching.

V. Security Considerations

We now briefly discuss attacks that adversaries inside or outside the IoT domain can mount against our extensions.

Replaying an abbreviated session resumption handshake. An eavesdropping adversary may replay an overheard session resumption handshake with compressed state as it contains no handshake-specific information that ensures freshness. This would allow the adversary to re-establish a previous session and consume scarce memory resources at the peer without prior authentication. To mitigate this attack, the session state contains a session resumption counter that is incremented after each successful session resumption and is used as a modifier when re-computing the Master Key. Hence, the Master Key and the MACs differ for each session resumption handshake. This allows to identify a replayed message because its MAC has been generated with an old Master Key.

DoS attacks against the session ticket mechanism. *Any* networked adversary could flood a target device with I1 messages containing forged session tickets, thus generating high decryption and verification load on the target device. To efficiently identify forged tickets that can be generated without restrictions on the adversary, an offloading peer should include a (random) plaintext key identifier in its tickets. An *eavesdropping* adversary could still reproduce these key identifiers in forged tickets from overheard legitimate session tickets. However, the adversary can generate similar computation load at the target device with the *standard* HIP DEX protocol by forging integrity-protected messages for overheard session establishments. Hence, session resumption does not open a new vector of attack. Finally, an eavesdropping adversary could replay an overheard session ticket in an abbreviated handshake with state offloading, thus misleading the offloading peer into re-establishing a previous session. However, the challenge-response mechanism in the abbreviated handshake enables the Responder to quickly expose this attack.

On-path attacks against our puzzle extension. The gateway adds a new parameter to the I1 message to notify a constrained Responder that it should issue a puzzle difficulty for a potentially unconstrained peer. This parameter is not cryptographically bound to the gateway. Hence, an on-path adversary could trick the Responder into issuing a puzzle difficulty for an untrusted network domain, although both peers are located in trusted domains. The resulting high puzzle difficulty would unnecessarily delay the handshake completion. However, an on-path adversary could also completely prevent the handshake by dropping handshake messages instead of forwarding them.

Forged message reception acknowledgements. The Responder cannot protect the integrity of a NOTIFY-based reception acknowledgement as it has not yet derived the Master Key when sending this message. Hence, an eavesdropping adversary can spoof reception acknowledgements for overheard I2 messages. However, spoofed acknowledgements do not

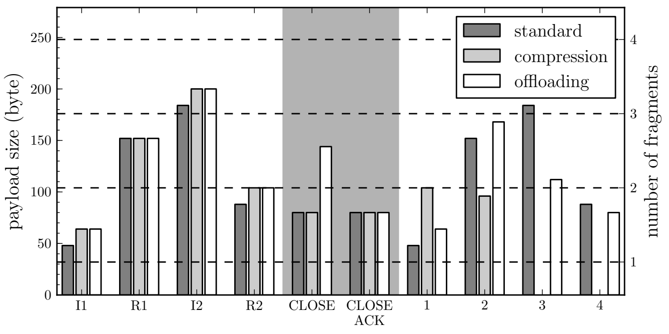

Fig. 4. Message sizes of two handshakes with an intermediate session tear down (marked with a grey background) for the *standard* HIP DEX protocol and for session resumption with state *compression* as well as with Initiator-side state-*offloading*. The dashed lines indicate the maximum content of a fragment with respect to the HIP DEX message size.

enable the adversary to prevent the handshake from completing successfully. Instead, the adversary can merely delay the handshake. This delay only occurs if the legitimate R2 message is actually lost on the forwarding path. Furthermore, the achievable delay is bounded by the worst-case anticipated computation time configured at the target device.

VI. Evaluation

For our evaluation, we implemented the HIP DEX protocol and the proposed protocol extensions for Contiki OS. We used Zolertia Z1 motes with a 16 MHz MSP430 micro-controller, 8 kB of RAM, 92 kB of ROM, and an IEEE 802.15.4 radio interface as our IoT devices. Moreover, we ported our HIP DEX implementation to Linux and implemented a simple firewall based on *netfilter* for the gateway functionality of the proposed puzzle extension. The Linux machines were equipped with an Intel Core i7 870 CPU. Our implementations use the *relic*[2] library for all public-key-based operations with the elliptic curve SECP160R1. Note that with larger curves especially the results for our proposed session resumption mechanism and our retransmission refinements further improve.

Regarding transmission overheads, we assumed *link layer* security in the IoT domain. We thus decreased the available payload size at the IPv6 adaptation layer (i.e., 6LoWPAN) by 21 bytes. The first fragment then contains up to 32 bytes and subsequent fragments at maximum 72 bytes of HIP DEX message content. By considering link layer security, our results show worst-case fragmentation for our proposed extensions.

A. HIP DEX Session Resumption

To quantify the improvements of the session resumption mechanism, we measured the transmission overhead and the processing time of two consecutive *standard* HIP DEX handshakes with an intermediate session tear down. We compared this baseline against measurements for *state compression* as the best-case and *Initiator-side state-offloading* as the worst-case session resumption type with respect to the achievable overhead reductions. Our results denote the average over 100 measurements with two wirelessly connected Z1 motes.

Transmission overhead. As shown in Fig. 4, session resumption with state compression marginally increases the message

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

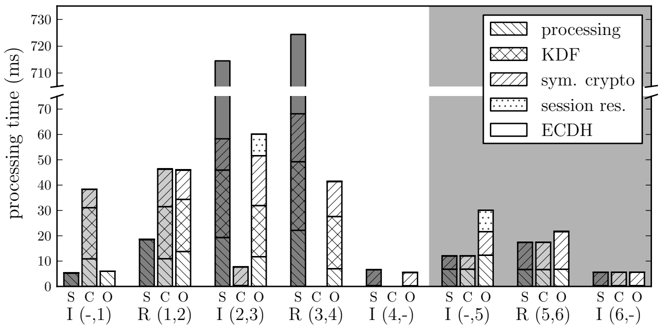

Fig. 5. Processing time of the HIP DEX handshake and the session tear down for the standard HIP DEX protocol (S) and for session resumption with state compression (C) as well as with Initiator-side state-offloading (O) for the Initiator (I) and the Responder (R). Numbers in brackets denote i-th packet processing and $(i+1)$-th packet generation.

sizes of the initial handshake and the session tear down exchange. However, the subsequent session resumption only requires 2 messages in the abbreviated handshake compared to 4 messages in a standard handshake. Each resumption and teardown cycle requires 360 bytes (i.e., 8 fragments), compared to 632 bytes (i.e., 15 fragments) for standard HIP DEX. State compression hence improves the transmission overhead by 43.0 % and reduces the number of fragments by 46.7 %.

For session resumption with Initiator-side state-offloading, the session tear down exchange requires additional 64 bytes to transfer the Initiator's session state. As a result, the CLOSE message requires a further fragment (see Fig. 4). The abbreviated handshake concludes after 4 messages, reducing the number of fragments by 1 (see third message of the second handshake in Fig. 4). In total, each resumption and tear down cycle requires 648 bytes (i.e., 15 fragments). Hence, state offloading marginally increases the transmission overhead by about 2.5 % with an identical number of transmitted fragments.

Processing time. Fig. 5 shows the average performance gain of session resumption compared to a standard handshake and session tear down. The standard deviation for all results was below 0.15 ms. The ECDH operations constitute the primary overhead of the standard handshake, requiring 656.25 ms per peer. The standard handshake amounts to 1469.41 ms for both peers combined. This compares to only 92.47 ms for the abbreviated session resumption handshake with state compression and to 159.17 ms for Initiator-side state-offloading. This 65 ms difference mainly stems from the state decryption and MAC operations for the additional messages in the latter case.

Notably, state compression has no performance impact on the session tear down exchange. In contrast, state offloading requires 8.42 ms for state encryption and slightly increases the MAC overhead due to the increased message size of the CLOSE message. Thus, Initiator-side state-offloading increases the processing overhead of the session tear down by 64.1 %. To put these numbers into perspective, session resumption reduces the computation overhead of a complete session lifecycle by at least 85.6 % and by 93.7 % at best.

Memory trade-offs. For state compression, both peers maintain 38 bytes of session state across connections to achieve the above improvements. This denotes a compression of an active session state (151 bytes) by 74.8 %. State-offloading lifts this storage requirement for one peers and requires the other peer to store 57 bytes of encrypted session state in addition to its own compressed state. Both states combined still denote a reduction of 37.1 % compared to an active session. Hence, session resumption with state-offloading is beneficial for *both* peers compared to active sessions over long periods of time. Session resumption thus considerably reduces the overhead of IP security protocols, especially for applications involving periodic communication patterns or keep-alive messages.

B. DoS Protection

For the evaluation of our puzzle-based DoS protection extension, we considered a network setup consisting of three IoT devices D_1, D_2, and D_3 and two Linux machines M_1 and M_2. M_1 represented a malicious Initiator that was connected to M_2 via Ethernet. M_2 was additionally connected to D_1 via USB. In combination, D_1 and M_2 represented the gateway in our network setup. The remaining two IoT devices constituted a legitimate Initiator (D_2) and Responder (D_3) that communicated wirelessly with each other and with the gateway.

We set the sliding window size to 64 s, i.e., the size of a *uint64_t*. Furthermore, we considered 5 handshakes per window period to be legitimate for a Responder, e.g., a CoAP server, and set the puzzle issuing threshold accordingly. Finally, we aimed for puzzles difficulties that prevent a single host from performing more than one handshake per window period during an attack. Thus, we measured the average computation time for puzzles with varying difficulties on an IoT device and on a Linux machine. Our results indicate a puzzle difficulty of 14 for IoT devices (64.56 s on average) and 22 for unconstrained hosts (60.44 s on average). Higher puzzle difficulties will be required as computing power increases.

With the above settings, we evaluated the impact of a DoS attack on a legitimate handshake between D_2 and D_3. Specifically, M_1 successively flooding D_3 with handshakes. During this attack, we measured the number of successful handshakes between D_2 and D_3 over 15 min. We considered three different configurations: i) puzzles with difficulty 14, ii) puzzles with difficulty 22, and iii) end-point-specific difficulties of 14 or 22 assisted by the gateway with our extension. As a baseline, we also counted successful handshakes *without* an attack.

We found that, with a fixed puzzle difficulty of 22, the number of legitimate handshakes decreased from 332 (no attack) to 7 as these puzzles were commonly insolvable for D_2. At the same time, we observed a puzzle-induced decrease of attack handshakes. More precisely, the Responder D_3 performed as few as 7 ECDH operations per window period for the adversary and the legitimate Initiator combined. Hence, the high puzzle difficulty successfully protected the Responder's scarce computation resources during the attack. A fixed puzzle difficulty of 14 decreased the number of legitimate handshakes to 8. This is the result of the undemanding puzzle difficulty for M_1 (0.21 s on average) and the corresponding high load at D_3 with a maximum of 29 ECDH operations per window period. In contrast, assisted by the gateway, the Responder selectively set a puzzle difficulty of 14 for D_2 and of 22 for M_1. As a result, the legitimate peers performed 14 handshakes. Note that this number is close to our goal of allowing only a single handshake per peer and window period during an attack. Simultaneously, the DoS attack of M_1 only caused a total of 7 ECDH operations per window period. Thus, our DoS

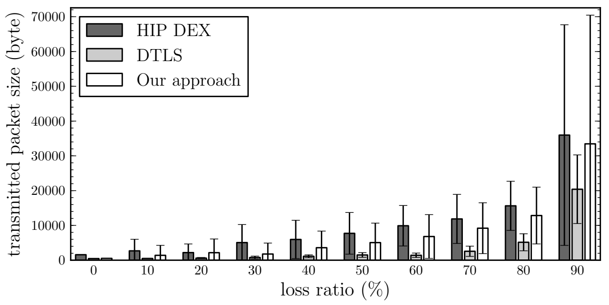

Fig. 6. Handshake transmission overhead for the different retransmission strategies and rising packet loss probability.

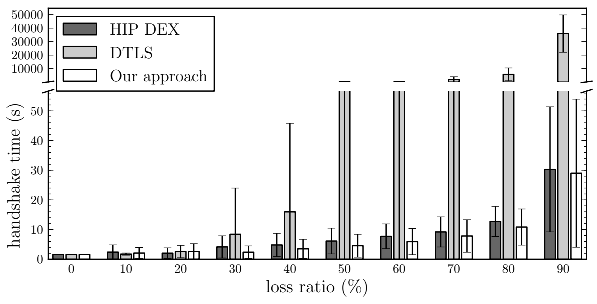

Fig. 7. Handshake run-time for the different retransmission strategies depending on the packet loss probability.

protection extension defends a constrained Responder against an unconstrained peer, while still supporting communication between legitimate peers in the same IoT domain.

Finally, we considered the case when a constrained Initiator communicates with a Responder in a foreign network domain. This Responder issues puzzles with a difficulty of 22 for the Initiator if the puzzle issuing threshold is exceeded. However, the puzzle difficulty decreases to 0 while the adversary is busy solving the puzzle. Hence, by dropping puzzles that could not be solved and by requesting a new ones, the Initiator was able to successfully establish a secure connection with the Responder. Specifically, the constrained Initiator was able to perform 16 successful handshakes within a 15 min timespan. Without our puzzle solving strategy, such high puzzle difficulties would cause the handshake to fail due to excessive computations.

C. Retransmission Refinements

We implemented the HIP DEX and the DTLS retransmission strategies along with our proposed extension and evaluated their performance in the Cooja network simulator for Contiki. Specifically, we measured the overall transmitted bytes and the handshake run-time for *end-to-end* loss probabilities between an Initiator and a Responder ranging from 0 % to 100 % for individual packet fragments. We decided for simulation over a real testbed to compare the different retransmission strategies for well-defined packet loss probabilities without side-effects on the wireless medium. Results denote the average over 5000 runs for each retransmission mechanism and loss probability combination.

We observed an average round-trip time (RTT) of about 30 ms in a simulation run without network load. Still, we set the *aggressive* timeout of the standard HIP DEX retransmission mechanism to 100 ms as lower values caused excessive premature I2 retransmissions for this strategy with high processing delays at the simulated nodes. Likewise, we configured the worst-case anticipated RTT of our proposed retransmission extension to 100 ms to account for network and processing delay under load conditions. Finally, we set the *worst-case* anticipated computation time to 750 ms according to the processing time discussed earlier (see Fig. 5).

As shown in Fig. 6, the aggressive HIP DEX retransmission strategy requires 1570 bytes on average, i.e., 333 % of a handshake without retransmissions, even without packet loss. This overhead exclusively stems from premature retransmissions. In contrast, the DTLS-based retransmission strategy largely avoids premature retransmissions and outperforms the other retransmission strategies regarding the transmission overhead. However, this comes at the cost of a quickly increasing handshake delay (see Fig. 7). Finally, our proposed retransmission mechanism shows a considerable improvement over standard HIP DEX with respect to the overall transmitted bytes and keeps retransmissions low for small loss probabilities. Transmissions are as low as 526 bytes (i.e. a full handshake and a NOTIFY message) for a loss probability of 0 %. Interestingly, the handshake run-time with our proposed extension outperforms the aggressive HIP DEX strategy with respect to the processing overhead as peers are less burdened with the processing of large handshake messages. Thus, we conclude that our proposed extension improves retransmissions in lossy IoT network scenarios compared to the considered strategies.

D. RAM and ROM Overhead

To derive RAM and ROM estimates, we analyzed the Contiki binaries of standard HIP DEX and our extensions with the *msp430-size* tool. Table I shows the marginal tradeoffs in case of our DoS protection extension and the refined retransmission mechanism. Session resumption adds a moderate overhead as a tradeoff for the significant handshake transmission and processing improvements. Overall, our proposed extensions require less than 5.6 kB of ROM and about 0.2 kB of RAM.

VII. Related Work

For related work, we distinguish three research directions: i) alternative cryptographic primitives and delegation architectures, ii) protocol mechanisms related to our extensions, and iii) further protocol optimizations in the context of the IoT.

Polynomial schemes offer an *alternative* to public-key-based primitives in DTLS and HIP [14], [15]. However, they rely on a central entity that distributes a polynomial share to each node in the network. Thus, while efficiently providing security in isolated network scenarios, non-trivial coordination between administrative domains is required to secure communication between devices of these domains. In contrast, we focus on standard primitives that afford the exchange of

Extension	ROM	RAM
Contiki with HIP DEX	58733	7042
+ Session resumption	63263 (+4530)	7198 (+156)
+ DoS protection	59369 (+636)	7066 (+24)
+ Retransmissions	59159 (+426)	7066 (+24)
All combined	64325 (+5592)	7246 (+204)

TABLE I. RAM and ROM requirements for our proposed extensions in byte. Numbers in brackets denote added overhead compared to Contiki with standard HIP DEX.

public keys for authentication purposes and propose protocol extensions to improve their applicability in the IoT.

Recently, several *delegation-based architectures* were proposed for HIP [16], [17], [18]. In each approach, the private key of the Initiator is split into multiple blocks. Proxies inside the IoT domain then perform public key-based operations on a private key block assigned by the Initiator. However, while these approaches reduce the computation cost at a constrained Initiator, they add a considerable number of network messages to the handshake. Furthermore, they require a trusted third party and pre-shared keys between the Initiator and the proxies. In contrast, our protocol extensions focus on minimizing computation as well as transmission costs and make minimal assumptions about the existing infrastructure. We believe that our protocol extensions can further improve the delegation schemes, e.g., using our session resumption mechanism.

With respect to *related protocol mechanisms*, several TLS extensions allow an Initiator to cache static Responder information and to omit this information during the next handshake [19]. However, mere caching of, e.g., the peer's public key, does not reduce expensive cryptographic operations in each handshake. Moreover, we recently highlighted session resumption as a possible solution to reduce the overhead of a certificate-based DTLS handshake in [20]. This paper extends on previous work by contributing a detailed description and evaluation of the proposed session resumption mechanism.

In [21], the authors identify the HIP puzzle as an effective DoS protection mechanism for the IoT and propose to consider the signal strength of received packets to detect an unconstrained adversary *inside* the IoT domain. Their detection mechanism is complementary to ours and could be incorporated into our attack detection strategy. Dean et al. [22] propose to extend the TLS handshake with a cryptographic puzzle and efficiently prevent attacks against the handshake. In contrast to our work, they do not consider resource heterogeneity.

Further IP security protocol optimizations for the IoT predominantly focus on header compression to reduce the transmission overhead, e.g., for HIP DEX [23] and IPsec [24]. Complementary to our work, the achieved compression further improves the applicability of HIP DEX and IPsec as the default payload protection mechanism for HIP DEX in the IoT.

VIII. Conclusion

In this paper, we analyzed the impact of public-key-based primitives in DTLS, HIP DEX, and minimal IKEv2. We identified three major challenges for IoT scenarios. First, public-key operations require significant resources from a constrained device. Hence, a full handshake should only be performed infrequently. Second, expensive public-key-based operations aggravate the risk of DoS attacks against constrained devices, even with a single, unconstrained adversary. Thus, DoS protection mechanisms for the IoT must account for resource heterogeneity. Third, retransmission strategies have to account for both, varying processing times of handshake messages and high packet loss in constrained wireless networks.

We presented three lightweight protocol extensions for HIP DEX to address these challenges. As our evaluation confirms, our proposed session resumption mechanism substantially reduces computation, memory, and transmission requirements compared to standard HIP DEX. Moreover, our collaborative puzzle-based DoS protection mechanism accounts for device and network heterogeneity and successfully protects IoT devices against more powerful adversaries. Lastly, our adaptive retransmission mechanism allows for a timely handshake conclusion despite packet loss. In combination, our extensions afford significant computation and transmission reductions and considerably improve the applicability of HIP DEX in the IoT. Notably, our proposed extensions also generalize to the wider scope of DTLS and minimal IKEv2.

References

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

[2] R. Moskowitz, "HIP Diet EXchange (DEX)," draft-moskowitz-hip-dex-00 (WiP), IETF, 2012.

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

[4] K. Hartke and O. Bergmann, "Datagram Transport Layer Security in Constrained Environments," draft-hartke-core-codtls-02 (WiP), IETF, 2012.

[5] R. Moskowitz et al., "Host Identity Protocol," RFC 5201, IETF, 2008.

[6] E. Barker et al., "Recommendation for Key Management – Part 1: General (Revision 3)," Special Publication 800-57, NIST, 2012.

[7] G. Van de Velde et al., "IPv6 Unicast Address Assignment Considerations," RFC 5375, IETF, 2008.

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

[9] Y. Sheffer and H. Tschofenig, "Internet Key Exchange Protocol Version 2 (IKEv2) Session Resumption," RFC 5723, IETF, 2010.

[10] J. Salowey et al., "Transport Layer Security (TLS) Session Resumption without Server-Side State," RFC 5077, IETF, 2008.

[11] R. Hummen and J. Gilger, "Extended DTLS Session Resumption for Constrained Network Environments," draft-hummen-dtls-extended-session-resumption-00 (WiP), IETF, 2013.

[12] R. Hummen et al., "HIP Middlebox Puzzle Offloading and End-host Notification," draft-hummen-hip-middle-puzzle-01 (WiP), IETF, 2013.

[13] R. Hummen et al., "6LoWPAN Fragmentation Attacks and Mitigation Mechanisms," in *Proc. of ACM WiSec*, 2013.

[14] A. Khurri et al., "On application of Host Identity Protocol in wireless sensor networks," in *Proc. of IEEE MASS*, 2010.

[15] O. Garcia-Morchon et al., "Securing the IP-based internet of things with HIP and DTLS," in *Proc. of ACM WiSec*, 2013.

[16] 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.

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

[18] 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.

[19] S. Santesson and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension," draft-ietf-tls-cached-info-14 (WiP), IETF, 2013.

[20] R. Hummen et al., "Towards Viable Certificate-based Authentication for the Internet of Things," in *Proc. of ACM HotWiSec*, 2013.

[21] P. Nie et al. , "Performance analysis of HIP diet exchange for WSN security establishment," in *in Proc. of ACM Q2SWinet*, 2011.

[22] D. Dean and A. Stubblefield, "Using client puzzles to protect TLS," in *Proc. of USENIX SSYM*, 2001.

[23] R. Hummen et al., "Slimfit - A HIP DEX Compression Layer for the IP-based Internet of Things," in *Proc. of IEEE WiMob Workshop IoT*, 2013.

[24] J. Granjal et al., "Enabling Network-Layer Security on IPv6 Wireless Sensor Networks," in *Proc. of IEEE GLOBECOM*, 2010.