P-HIP: A Lightweight and Privacy-Aware Host Identity Protocol for Internet of Things

Mahmud Hossain and Ragib Hasan

Abstract— The Host Identity Protocol (HIP) has emerged as the most suitable solution to uniquely identify smart devices in the mobile and distributed Internet of Things (IoT) systems, such as smart cities, homes, cars, and healthcare. The HIP provides authentication methods that enable secure communications between HIP peers. However, the authentication methods provided by the HIP cannot be adopted by the IoT devices with limited processing power because of the computation-intensive cryptographic operations involved in hash generation, signature validation, and session key establishment. Moreover, IoT devices cannot utilize the HIP as is to communicate securely in the low power and lossy networks as there is a considerable communication overhead, such as packet fragmentation and reassembly, for exchanging certificates over a lossy link. Additionally, the use of static host identifiers makes IoT devices vulnerable to cyber espionage and user-targeted attacks. In this article, we propose an authentication scheme, P-HIP, that protects the identity privacy of an IoT device by enabling the device to compute and use unique host identifiers from networks to networks and sessions to sessions. To make the HIP suitable for resource-constrained IoT devices, P-HIP provides methods that unburden IoT devices from computation-intensive operations, such as modular exponentiation, involved in authentication and session-key exchange. Additionally, P-HIP minimizes the communication overheads for exchanging certificates in lossy networks. We implement a prototype of P-HIP on Contiki enabled IoT that shows P-HIP can reduce computation costs, communication overheads, and the session-key establishment time when used by low-powered devices in a lossy network.

Index Terms—Security, Privacy, Host Identity, Authentication, and Internet of Things

1 Introduction

The Internet of Things (IoT) based systems are becoming ubiquitous in a wide range of application domains, such as smart city [1, 2], intelligent healthcare service [3, 4], connected vehicles [5, 6], and smart wearables [7], due to the recent advances in wireless communications and pervasive computing. An IoT system can have two types of smart device: mobile device (e.g., wearables and automobiles) and stationary device (e.g., smart home appliances). A user of a smart device can communicate to the device from anywhere at anytime regardless of the type (mobile or stationary) of the device. The user can control an IoT device remotely through Cloud services. The user can also interact with the IoT device locally being co-located with the device in the same network.

According to Gartner, approximately 5.8 billion IoT devices will be in use worldwide by 2020 [8]. Therefore, IoT devices require a global identification and naming scheme even more than the legacy Internet nodes. In the current Internet architecture, Internet Protocol (IP) addresses are used to identify a host and its location. As a result, it becomes challenging to locate IoT devices that travel from one network to another network using their IP addresses. Moreover, a client that sends data to an IoT device is only interested in the identity of the recipients, while the routing components in the source and intermediary nodes are only concerned with the recipient's location. Additionally, the confidentiality of the communications that take place between IoT devices needs to be protected as these devices can exchange sensitive information, such as patients' health records. The identity privacy of these devices also need to be ensured as these devices can be the target of identity spoofing, location recording, and communication tracking attacks.

• *Mahmud Hossain (mahmud@uab.edu) and Ragib Hasan (ragib@uab.edu), SECuRE and Trustworthy computing Lab (SECRETLab), Department of Computer Science, University of Alabama at Birmingham, Alabama 35294, U.S.A.*

There are some research work [9–11] that proposes methods to identify devices in mobile environments. The Mobile IP [9, 10] proposes the use of two IP address for a mobile device: one IP address is used as a locator, such as routing or forwarding messages, and the other IP address is used as the identity of the device. The Named Data Networking protocol (NDN) [11] eliminates the use of IP addresses and uses content information, such as universal resource identifiers (URIs), to locate and identify hosts and to route requests on the Internet. However, further research is required to validate the applicability of URI based addressing and routing in the Low-powered Wireless Personal Area Networks [12, 13], where IoT devices use the IPv6 for addressing and the RPL [14] for routing. Although Mobile IP and NDN provide methods to identify hosts in mobile environments, they do not consider security and identity privacy in their designs; instead these methods rely on the application layer protocols for the communication security.

The Host Identity Protocol (HIP) [15–20] can a suitable solution for IoT devices considering the security and privacy requirements of stationary and mobile IoT systems. The HIP can uniquely identify devices both in the mobile and stationary IoT environments and can be utilized to ensure identity and communication security. The HIP eliminates the dual role, such as identifier and locator, of an IP address. In the HIP, host identifiers are used to identify IoT devices, while IP addresses are used to locate these devices. An IoT device is issued a public key. The device uses its public key and a 128 bit hash of the public key (Host Identity Tag) its host identifier. An HIP host identifies its peer by using the Host Identity Tag (HIT) of the peer.

After identification, an IoT device requires to authenticate the peer before exchanging information. Moreover, the communications need be encrypted to protect the confidentiality of the exchanged messages. In this regard, the HIP provides methods for mutual authentication and key exchange. However, the authentication and key exchange methods involve computation intensive cryptographic operations, such as asymmetric encryption and decryption, signature generation and verification, and modular exportation operations. These operations are too heavy to be executed by the resource-constrained IoT devices with limited processing power and memory. It can be noted from Table 1, IoT devices [21–25] have a few megabytes of memory (8 KB to 32 KB of RAM and 48 KB to 512 KB of ROM) and low powered CPUs (8 MZ to 96 MZ clocks).

The HIP peers exchange X.509 certificates [26] to prove that a trusted party issued the host identifiers (public keys and HITs). However, the size of the certificates is considerably larger than the Maximum Transmission Unit (MTU) of the IEEE 802.15.4 link [12] that IoT devices use for communications. IoT devices operate on lossy networks, such as 6LowPAN [13], Controlled Area Network (CAN) [27], KNX [28], Z-Wave [29], and Zigbee [30]. These networks have limited bandwidth with a data rate ranging from 16 kbps to 250 kbps. The certificates are sent in multiple fragments over the lossy links. As a result, there are notable communication and energy overheads for fragment processing and packet delivery both on the sender and receiver IoT devices. The communication overheads limit an IoT device to respond to real-time requests and the energy overheads shorten the device's battery life.

Although the HIP ensures secure communications, such as authentication, integrity, and encryption, between IoT devices, it does not address the privacy issues of using the same host identifiers over and over again for authentication. An IoT device's host identifiers are static. A mobile IoT node uses the same public key and HIT to authenticate to its peers when it moves from one network to another network. Adversaries can track the locations of a mobile IoT device or the owner of the device by learning its public key and HIT. Later, this location information can be used to profile the movements of the device owner. The location information can also be used for cyber espionage. Therefore, the HIP is vulnerable to user-targeted attacks.

In this article, we propose P-HIP, a privacy-aware and lightweight authentication method for the HIP. P-HIP enables IoT devices to compute a unique public key and HIT every time it communicates to its peer or moves from one network to another network. Thus, P-HIP allows mobile devices to avoid using a static host identifier. The use of a unique identifier for protects an IoT device from the location recording and identity tracking attacks. We also propose an authentication scheme based on the Elliptic Curve Qu-Vanstone (ECQV) [34] cryptography that unburdens resource-limited devices from the computation overheads for the certificate verification, such hash computation and signature validation. Moreover, P-HIP eliminates the modular exponentiation operations involved in the asymmetric encryption and decryption and session key derivation. Unlike X.509 certificates, the ECQV-based credentials, such as public key and public key validation data, fit in an IEEE 802.15.4 frame and can be sent in a single fragment. Hence, P-HIP reduces communication overheads and energy costs for packet fragmentation, fragment delivery, and reassembly.

Contributions: The contributions of this article are as follows:

1) We propose a host identity computation scheme that enables devices to compute network or session specific host identifiers without communicating to a certificate authority. The network and session specific identifiers protect the location and identity privacy of an IoT devices and the owner of the device.
2) We present a lightweight authentication scheme that does not require IoT devices to perform hash and signature operations to validate host identifiers. Hence, the authentication method reduces computation overheads and enables devices to perform mutual authentication faster.
3) We provide a security analysis of P-HIP which shows that P-HIP is secure against various network attacks, such as spoofing and replay.
4) We implement a proof-of-concept of P-HIP and provide a performance analysis which demonstrates that P-HIP reduces communication, computation, and energy costs for authentication and key exchange.

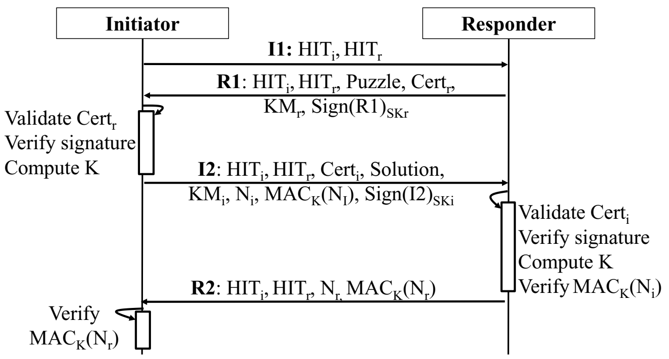

Initiator
Responder
I1: HIT$_i$, HIT$_r$
R1: HIT$_i$, HIT$_r$, Puzzle, Cert$_r$, KM$_r$, Sign(R1)$_{SKr}$
Validate Cert$_r$
Verify signature
Compute K
I2: HIT$_i$, HIT$_r$, Cert$_i$, Solution, KM$_i$, N$_i$, MAC$_K$(N$_I$), Sign(I2)$_{SKi}$
Validate Cert$_i$
Verify signature
Compute K
Verify MAC$_K$(N$_i$)
R2: HIT$_i$, HIT$_r$, N$_r$, MAC$_K$(N$_r$)
Verify MAC$_K$(N$_r$)

Fig. 1: HIP handshakes for authentication and key exchange.

Organization: The rest of this article is organized as follows. We present a background on the HIP authentication process, IoT networking, and Elliptic Curve Cryptography (ECC) in Section 2. We provide the related work in Section 3. The problem statement and motivation are provided in Section 4. Section 5 provides the details of the operational model of P-HIP. A security analysis of P-HIP is presented in Section 6. We present the experimental results in Section 7. Finally, we conclude this article in Section 8.

2 Background

2.1 HIP Authentication Process

In this section, we provide an overview of the authentication process of the HIP. In the HIP, communicating peers utilize the public key cryptography for authentication and the peers are identified by their Host Identity Tags (HITs). A host's HIT is computed from its public key. A HIT is a 128 bits hash of a public key. Figure 1 shows the HIP handshakes to establish a secure association between two HIP peers.

An Initiator sends an I1 message to a Responder. The I1 message contains the HITs of the textcolorblackInitiator (HIT_i) and Responder (HIT_r). The Responder receives the I1 message and then creates an R1 message that contains HIT_i, HIT_r, a puzzle, and its own certificate $Cert_r$. The puzzle protects the Responder from a Denial of Service (DoS) attack. The Responder's keying material (KM_r) to establish a session is also included in the R1 message. The Responder signs the R1 message using the private key SK_r of its key pair (SK_r, PK_r). The PK_r is the Responder's public key. The Initiator receives the R1 message and validates the $Cert_r$ to ensure that the PK_r included in the $Cert_r$ is issued by a trusted party. Next, the Initiator verifies the signature included in the R1 message by using the PK_r and authenticates the Responder. The

TABLE 1: Specifications of some representative devices used for IoT applications.

Device Specification	CPU		Storage		Networking		
	Arch (Bits)	Clock (MHz)	RAM (KB)	ROM (KB)	Standard	Radio Interface	Bandwidth (kbps)
Sky-Mote [22]	16	8	10	48	6LoWPAN	IEEE 802.15.4	250
Z1-Mote [21]	32	32	32	512	6LoWPAN	IEEE 802.15.4	250
Openmote [24]	32	32	32	512	6LoWPAN	IEEE 802.15.4	250
Waspmote [25]	8	16	8	128	Zigbee	IEEE 802.15.4	250
Arduino Uno [31]	8	16	2	32	6LoWPAN	IEEE 802.15.4	250
Mbed [32]	32	96	32	512	CAN	CAN Bus	320
Weptech [33]	32	32	32	512	6LoWPAN	IEEE 802.15.4	250
KNX Stacks [28]	32	32	32	512	KNX	KNX Radio	16.4

Initiator solves the puzzle and computes a session key (K) using its own session keying material KM_i and the Responder's session keying material KM_r as $K = KeyExchangeMethod(KM_i, KM_r)$. The Initiator creates an I2 message that includes HIT_i, HIT_r, it own certificate $Cert_i$, the solution of the puzzle, a signature computed using it private key SK_i of the key pair (SK_i, PK_i), and a message authentication code for a nonce N_i computed using K as $MAC_K(N_i)$. The Initiator sends the I2 message to the Responder. The Responder verifies the authenticity of the $Cert_i$, validates the signature using the PK_i included in the $Cert_i$, and authenticates the Initiator. Next, the Responder computes the session key K using KM_i and KM_r, and then validates the $MAC_K(N_i)$. The Responder creates an R2 message that includes a $MAC_K(N_r)$ for a nonce N_r. The Initiator receives the R2 message and validates $MAC_K(N_r)$. Hence, the Initiator ensures that a secure session is created successfully. After this point, the communications are encrypted using K.

2.2 Low Power and Lossy IoT Network

IoT devices operate on low-powered wireless personal area networks (LoWPANs). The IEEE 802.15.4 [12] standard defines the operation of LoWPANs. An IEEE 802.15.4 radio link has a bandwidth of 250 Kbps. The MTU of an IEEE 802.15.4 link is 127 octets which is much lower than the MTU of an IPv6 packet (1280 octets). Therefore, a full IPv6 packet does not fit in an IEEE 802.15.4 frame and requires to send in multiple fragments. The 6LoWPAN [13] protocol provides a method to send and receive IPv6 packets (or packet fragments) over IEEE 802.15.4 based networks.

In Figure 2, we present the details of an IEEE 802.15.4 frame. We consider the User Datagram Protocol (UDP) as the transport layer protocol to compute the size of the application payload for an IEEE 802.15.4 frame because UDP has fewer communication overheads than the Transmission Control Protocol (TCP). The size of the UDP headers is smaller than the size of the TCP headers. As a result, the number of IPv6 packet fragments are fewer in the UDP communications than in the TCP communications. Moreover, unlike TCP, UDP does not require communicating devices to perform a three-way handshake to send a message. Therefore, in lossy networks, a devices can exchange messages faster when UDP is used instead of TCP.

As shown in Figure 2, starting from a maximum physical layer packet size of 127 octets and the maximum header overheads of 98 octets, the resultant maximum frame size at the application layer is 29 octets. Application payloads larger than 29 bytes are sent in multiple fragments.

Two types of routing schemes are used to forward packet fragments in an IoT network [14]: mesh-under and route-over. In the mesh-under routing, a routing header (mesh-header) is added to every frame (Figure 2c). As such, application payloads larger than 29 bytes are fragmented. A forwarding node uses the mesh-header to make a routing decision on a per-fragment basis.

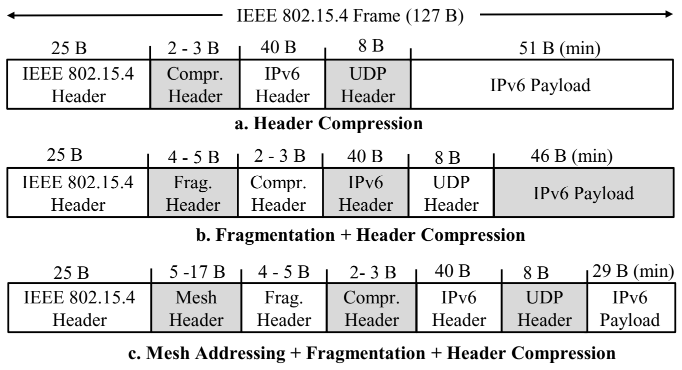


Fig. 2: IEEE 802.15.4 Frame. min = Minimum.

In contrast, in the route-over routing, application payloads larger than 46 bytes are fragmented (Figure 2b). The route-over scheme makes forwarding decisions on a per-packet basis. Every forwarding node reassembles the fragments of an IP packet to make the routing decision. Next, the on-path node re-fragments the packet and sends the fragments to the next hop based on the information available in its small-sized routing table.

2.3 Combined Public Key

In the combined public key (CPK), a new public key is computed with the addition of two or more public keys. The computation of a CPK relies on the Elliptic Curve Cryptography (ECC). Let $G = (G_x, G_y)$ is a base point on the elliptic curve and n is a prime number and has order G. An ECC key pair is represented as (d, Q) such that d is a private key and Q is a public key. The Q is a point on the elliptic curve $Q = (Q_x, Q_y)$. The d and Q are computed as $d \in [1, n-1]$ and $Q = d * G$.

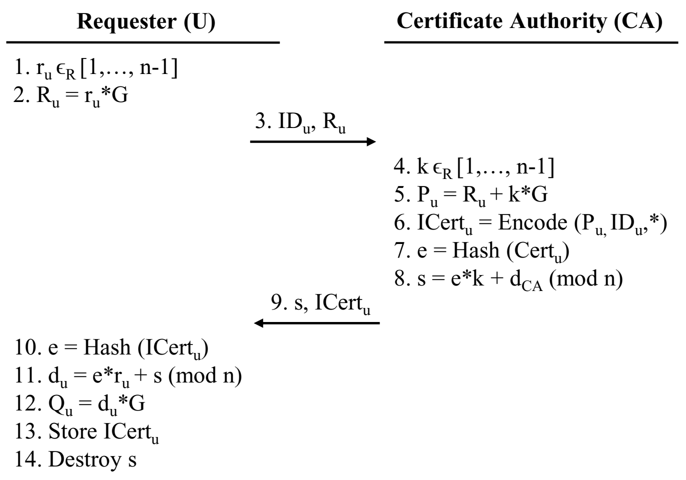
In the Elliptic Curve Cryptography (ECC),

One characteristic of ECC key pair is that the combination of private keys and corresponding public keys in ECC is still a pair of elliptic curve keys, which we denote as CPK.

For example, d_1 and d_2 are private keys in an elliptic curve, and their corresponding public keys are $Q_1 = d_1 * G$ and $Q_2 = d_2 * G$ respectively. A new ECC key pair (d_n, Q_n) can be computed as $d_n = d_1 + d_2$ and $Q_n = Q_1 + Q_2$. For completeness, we prove that Q_n corresponds to $d_n * G$ as follows: $Q_n = Q_1 + Q_2 = d_1 * G + d_2 * G = (d_1 + d_2) * G = d_n * G$. Therefore, the combination of ECC key pairs generates a new key pair (or CPK).

2.4 ECQV Based Credentials

The ECQV based credentials, such as implicit certificates [34] , public keys, and public key validation data, are computed using the



Requester (U) — **Certificate Authority (CA)**

1. $r_u \in_R [1,\ldots, n-1]$
2. $R_u = r_u{*}G$
3. ID_u, R_u
4. $k \in_R [1,\ldots, n-1]$
5. $P_u = R_u + k{*}G$
6. $ICert_u$ = Encode (P_u, ID_u,*)
7. e = Hash ($Cert_u$)
8. $s = e{*}k + d_{CA}$ (mod n)
9. s, $ICert_u$
10. e = Hash ($ICert_u$)
11. $d_u = e{*}r_u + s$ (mod n)
12. $Q_u = d_u{*}G$
13. Store $ICert_u$
14. Destroy s

Fig. 3: The process to issue an ECQV implicit certificate.

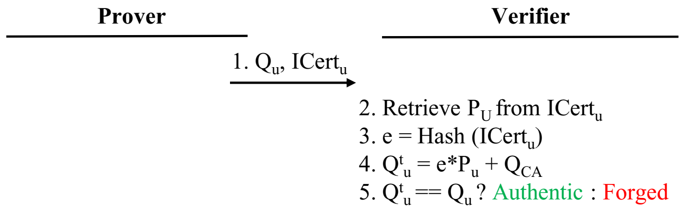
Elliptic Curve and Combined Key cryptography. The methods to issue an implicit certificate and validate the certificate are described below.

2.4.1 Implicit Certificate Issue Process

Figure 3 shows the process to issue an ECQV implicit certificate (ICert). **Step 1–3:** A requester (U) computes an ephemeral ECC key pair (r_u, R_u) such that $r \in [1, n-1]$ and $R_u = r * G$. The requester provides its identity ID_u and an ephemeral public key R_u to a Certificate Authority (CA). **Step 4–6:** The CA computes an ephemeral key pair $(k, k * G)$ by selecting a random private key k, and then computing a public key as $k * G$. The CA generates a public key construction data P_u by applying CPK as P_u = R_u+ $k * G$. Next, the CA encodes ID_u and P_u in a certificate as $ICert_u = Encode(P_u, ID_u)$. The P_u is used as a proof that the ICert is issued by the CA. A verifier uses the P_u to validate the authenticity of the $ICert_u$. **Step 7–9:** The CA computes a private key construction data s using the private key d_{CA} of its ECC key pair (d_{CA}, Q_{CA}). The CA provides s and $ICert_u$ to the requester. **Step 10–12:** The requester computes a private key d_u using s and r_u as $d_u = e * r_u \ + \ s \ (mod\ n)$, and a public key Q_u as $Q_u = d_u * G$.

2.4.2 Implicit Certificate Validation Process

Figure 4 presents the method to validate the public key Q_u. A prover presents its $ICert_u$ and Q_u to a verifier. The verifier validates that Q_u can be reconstructed by using the public key validation data P_u and the public key Q_{CA} of the CA. The verifier parses $ICert_u$ and retrieves P_u. The verifier generates a hash of the $ICert_u$ as $e = Hash(ICert_u)$. Next, it computes a public key as $e * P_u + Q_{CA}$. If this operation results in Q_u, then the verifier ensures that the Q_u is authentic.



Prover — **Verifier**

1. Q_u, $ICert_u$
2. Retrieve P_U from $ICert_u$
3. e = Hash ($ICert_u$)
4. $Q^t_u = e{*}P_u + Q_{CA}$
5. $Q^t_u == Q_u$? Authentic : Forged

Fig. 4: The process to verify an ECQV certificate.

For completeness of the public key validation method, we prove that the public key Q_u corresponds to d_u:

$$\begin{aligned} Q_u &= e * P_u + Q_{CA} \\ &= e(R_u + k * G) + d_{CA} * G \\ &= e * (r_u * G + k * G) + d_{CA} * G \\ &= e * r_u * G + (e * k + d_{CA}) * G \\ &= e * r_u * G + s * G \\ &= (e * r_u + s) * G \\ &= d_u * G \end{aligned}$$

3 Related Works

In this section, we survey HIP authentication methods and identity their limitations.

3.1 HIP-Base exchange (HIP-BEX)

In the HIP-BEX [15], two HIP peers authenticate each other by using the RSA cryptography. The HIP peers require to compute and validate RSA signatures for authentication. The peers perform Diffie-Hellman (DH) key exchange [35] to establish a session key. The HIP-BEX cannot be adopted by the resource-constrained IoT devices because of the modular exponentiation operations involved in the RSA signature generation and verification process and in the DH public and session key computations process. An RSA signature is generated as $s = m^d$ (mod n) such that s = signature, m = message, n = prime, and d = private key, and verified as $m = s^e$ (mod n) such that e = public key. An Initiator computes its DH public key A as $A = g^a$. Similarly, a Responder computes its DH public key B as $B = g^b$. The terms a and b represent the DH private keys of the Initiator and Responder respectively. The Initiator and Responder compute a DH session key k_{dh} as $k_{dh} = B^a = g^{ab}$ and $k_{dh} = A^b = g^{ab}$ respectively. IoT devices [21–25] with limited processing power (8 MHz to 48 MHz CPU without floating point unit) cannot afford these modular exponentiation operations.

3.2 Distributed HIP (D-HIP)

The D-HIP [16] delegates the modular exponentiation (g^a and g^{ab}) involved in the DH public value computation ($A = g^a$) and session key derivation ($k_{dh} = g^{ab}$) operations to the fewer resource-constrained nodes (proxies) co-located with IoT devices using a collaborative scheme. An Initiator selects a set of proxies and delegates the key exchange operation to the proxy nodes. The Initiator splits its secret exponent a into multiple blocks $a_1, a_2, \ldots, a_n$ with $\sum_1^n a_i = a$ and sends these to the proxies. Each proxy receives a unique block a_i and computes its part of the Initiator's public DH key g^{a_i} and sends it to a Responder. The Responder receives all the parts from the proxies and computes the Initiator's public DH key as $\prod_1^n g^{a_i} = g^{\sum_1^n a_i} = g^a$. The Responder sends its secret exponent b to the proxies. Each proxy computes its part of the DH key $g^{a_i b}$ and sends it to the Initiator. The Initiator computes the session key k_{dh} $= \prod_1^n g^{a_i b} = g^{\sum_1^n a_i b} = g^{b \sum_1^n a_i} = g^{ab}$.

The D-HIP assumes that only the Initiator device is resource-constrained. Therefore, the D-HIP delegates the modular exponentiation operations $A = g^a$ and $k_{dh} = g^{ab}$ performed by an Initiator to the proxy nodes. However, in device-to-device communications, a Responder can also be resource-constrained. Therefore, the Responder requires to delegate its modular exponentiation operations to the proxies as well. As such, there can be a significant increase in the time to establish a session key. Moreover, there can be a considerable communication overheads for exchanging numerous messages between the Initiator and proxies as well as between the proxies and the Responder. Moreover, the proxy selection scheme is vulnerable to DoS attacks. The D-HIP requires the initiator to select proxy nodes and split and distribute keys between them if a single proxy fails to generate its own part of the DH-key correctly. A malicious proxy can exploit this property to perform a DoS on the initiator node. For instance, a malicious proxy can keep providing an incorrect DH key to the Initiator to force the initiator to perform the proxy selection task over and over again.

3.3 HIP-Tiny Exchange (HIP-TEX)

The HIP-TEX [17] replaces the computation intensive DH key exchange with a cooperative secret key exchange using the public key cryptography. A Initiator and Responder encrypt the cryptographic materials required to establish a session key using their RSA public keys. In the HIP-TEX, nearby proxy nodes participate in the key exchange scheme through a collaborative scheme similar to the D-HIP. Although the HIP-TEX eliminates the computation overheads for the DH key agreement, it introduces the RSA encryption and decryption operations which are based on the modular exponentiation operations: $c = ENCRYPT_{rsa}(m) = m^e \ mod \ n$; $DECRYPT_{rsa}(c) = c^d \ mod \ n$ such that c = ciphertext and m = message. The HIP-TEX has the same disadvantages as the D-HIP, since both of these schemes rely on the distributed collaborations of the proxy nodes. Similar to the D-HIP, the HIP-TEX increases communication overheads and secret key setup time and is vulnerable to DoS attacks.

3.4 HIP-Diet Exchange (HIP-DEX)

The HIP-DEX [18] adopts the ECC to avoid the computation overheads for the DH key exchange. The HIP-DEX uses a long-term Elliptic Curve Diffie-Hellman (ECDH) [36] public value as a Host Identifier and the ECDH key exchange to establish a session key. The ECDH key exchange eliminates the computation cost for the two modular exponentiation operations: an ephemeral DH public key generation cost ($A = g^a$) and a DH session key computation cost ($k_{dh} = g^{ab}$). The ECC also eliminates the computation intensive cryptographic primitives involved in the RSA signature generation and verification operations. As a result, the HIP-DEX sacrifices some of the security properties, such as forward secrecy and the option for selecting cryptographic suites. The HIP-DEX uses the Elliptic Curve Digital Signature Algorithm (ECDSA) [37] to sign and validate a signature. The ECDSA signature generation and verification operations require the HIP hosts to perform inverse operations. Therefore, the ECDSA signature is still too heavy to be supported by highly resource constrained IoT devices, such as a TMote-Sky device with a CPU speed of 8 MHz [22].

3.5 HIP Header Compression

The authors in [19] propose a compression layer (Slimfit) in the protocol stack to reduce the size of the HIP headers. The Slimfit layer is introduced between the HIP and network layer. The Slimfit layer compresses the HIP headers of the outgoing packets before sending them to the network layer. For the incoming packets, the Slimfit layer decompresses their HIP headers, and then sends them to the HIP layer. The Slimfit layer reduces the transmission overheads for sending HIP headers. However, the computation costs are increased due to the header compression and decompression operations.

3.6 Lightweight HIP (LHIP)

The LHIP [20] does not consider the major security requirements to obtain simplicity. The LHIP does not perform host authentication and encryption; thus, it avoids cryptographic operations of signature computation and validation. In the LHIP, communicating peers do not perform the DH or ECDH key exchange; Therefore, the communications between the hosts are not encrypted. The LHIP achieves a minimal degree of security by authenticating succeeding messages using hash chains and has a mechanism to detect session hijacking.

4 Problem Statement and Motivation

4.1 Communication and Computation Overheads

The collaborative schemes [16, 17] are proposed to reduce the computation cost for the session key establishment. However, these schemes introduce communication overheads for proxy selection and key distribution with the cost of reducing the computation costs. The schemes increase the session key setup time as numerous messages are need to be exchanged between between the proxies, Initiator and Responder to derive a session key. Therefore, [16, 17] are not suitable for IoT devices that need to respond to real-time requests.

In [19], HIP headers are compressed to reduce the communication overheads for sending and receiving messages. However, there is an increase in the energy consumption and runtime both on the sender and receiver devices as every outgoing packet is compressed by a sender and every incoming packet is decompressed by a receiver.

None of the authentication schemes discussed in Section 3 provide solutions to minimize the computation overheads for encryption, and signature generation and verification. The schemes also do not consider the communication overheads for exchanging certificates in the I1 and R1 messages. The I1 and R1 messages are sent in numerous fragments since the size of a certificate is larger than the maximum size of the application data frame (see Figure 2). Additionally, the size of a certificate increases with an increase in the length of a public key. The increase in the certificate length results in an increase in the total number of message fragments. Moreover, the time to deliver and process fragments and establish a session key are increased because of the increase in the number of fragments.

In Figure 5, we present a correlation between the size of a certificate and the length of a public key. The certificates are self signed and created using the OpenSSL library [38]. From Figure 5, it can be noted that the increase in the key length increases the size of a certificate. in Figure 6, we also provide a correlation between the size of a certificate and the number of packet fragments. The number of fragment increases with the increase in the certificate size as shown in Figure 6. From the figure, it can also be observed that the number of fragments is larger in the Mesh-under routing (Figure 6a) than in the Route-over routing (Figure 6b). This is due to the fact that the space for an application payload in the IEEE

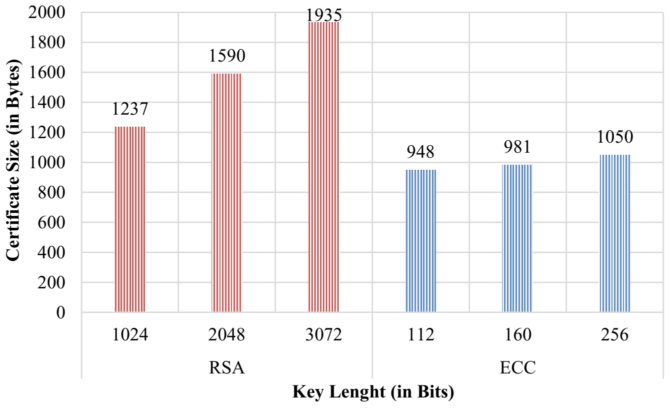
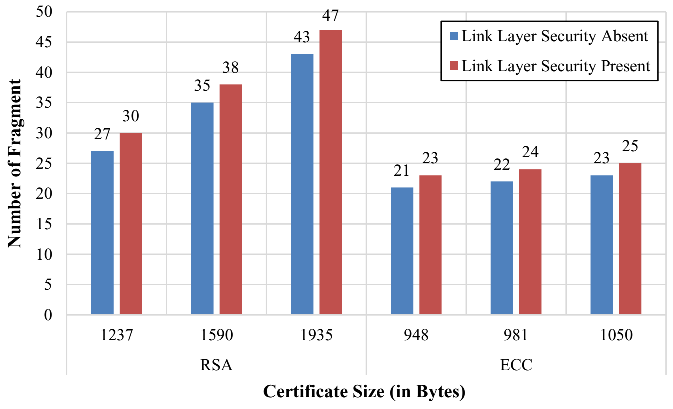
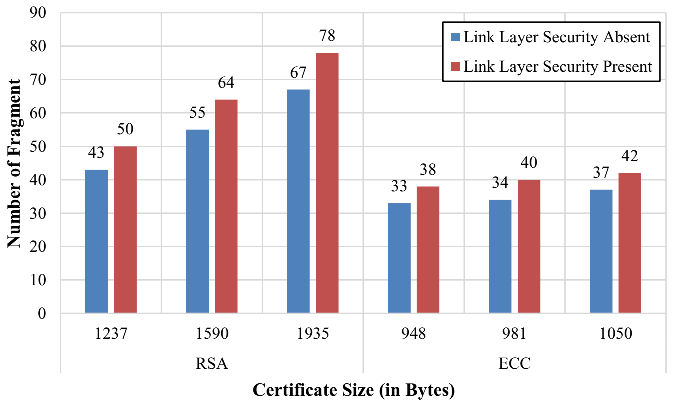


Fig. 5: X.509 certificate size Vs. Public key length.

802.15.4 is reduced by 17 Bytes if Mesh-under routing is adopted – the space for an application payload is 46 Bytes (Figure 2b) and 29 Bytes (Figure 2c) in the Mesh-under and Router-over routing respectively. Additionally, the number of packet fragment increases if the link-layer security (e.g., IPSec [39]) is used. In the link-layer security, additional header fields are added to an IEEE 802.15.4 frame. As a result, the room for an application payload is reduced and the number of fragments is increased.



(a) Certificate size Vs. Packet fragments (Route-over routing).



(b) Certificate size Vs. Packet fragments (Mesh-under routing).

Fig. 6: An analysis of packet fragments for X.509 certificates.

4.2 Threat Model

Mobile IoT devices can be a target of the location tracking attacks. Similarly, stationary devices can a subject of local DOS attacks. In this section, we provide some of the details of these types attacks.

4.2.1 Movement Profiling

An adversary can record the locations of mobile IoT devices by tracking their static host identifier. These devices can range from high-powered smart vehicles to low-powered wearable and implementable (e.g., pacemakers devices) sensors. A movement profiling attack can be performed as follows.

An adversary tracks the locations of a mobile IoT device, which travels from one network to another network, being co-located with the target device. Next, the location data of the device is used to predict its future locations for a specific day. The adversary can use the prospective location data to perform user-targeted attacks. The adversary can also use the location data to spy on a target device or the owner of the device, such as military personnel or government officials [40, 41].

In Figure 7, we present a scenario for the location tracking attack. Although we consider a connected car scenario to better understand the movement profiling attack in the mobile IoT systems, adversaries can perform such an attack on other mobile IoT devices including wearable and implantable sensors.

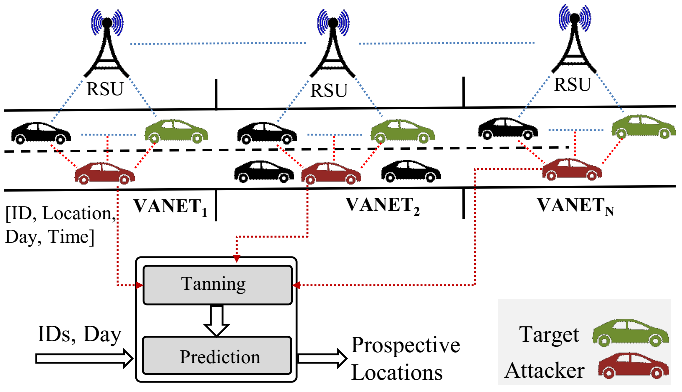


Fig. 7: Movement profiling in VANETs.

4.2.2 Communication Relation

Adversaries can perform various types of DoS attacks, such as communication jamming, resource exhaustion, and service disruption, in the LoWPANs. In this paper, we consider a type of local DoS attack where a malicious IoT device can disrupt the communications of the selective smart devices. The details of the attack scenario are as follows.

The use of static identities enables adversaries to learn the identities of communicating devices. An adversary can use this knowledge to disrupt communications when a target devices start exchanging messages with it peer. To better explain these types of attacks, we consider a smart home where the IP camera of the home is infected with a malware [42]. The compromised camera can learn the identity of the IoT-enabled pacemaker, which is implanted in the heart of the homeowner, when it presents its static identity to a cloud medical service for authentication purposes. The cloud service enables a physician to monitor the heart condition of the owner remotely. The malware can configure the camera such that it jams the communication channel between the pacemaker and the cloud service when the camera finds that the pacemaker is sending data to the cloud service. Hence, a malicious device can block critical health information from being transmitted to physicians, threatening the life of the patient. Adversaries can come up with similar attack models [43] by exploiting IoT devices' static identities. Figure 8 shows an overview of such an attack.

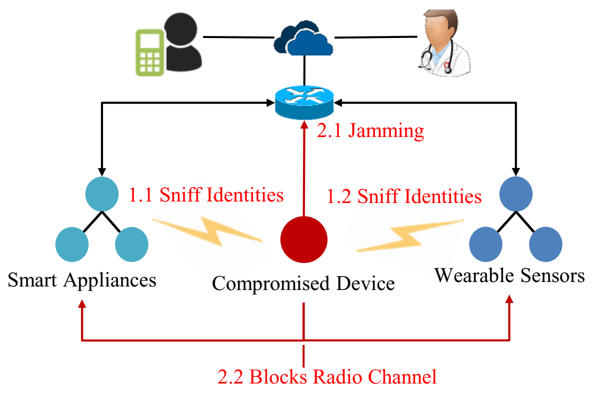


Fig. 8: Communication disruption attacks.

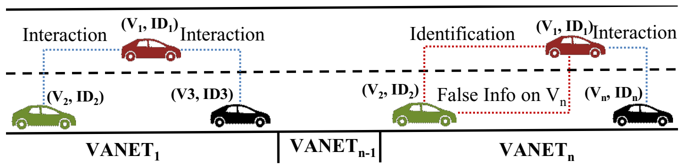


Fig. 9: An attack scenario for providing false information.

4.2.3 Falsifying Information

In the mobile ad-hoc networks, adversaries can provide false information to a target device. The false information can limit the device to make a right decision. For instance, in the VANETs, a malicious smart car can learn the identities of its nearby vehicles. Later, the malicious vehicle can provide a false information on the speeds and lane positions of nearby vehicles to a target car after recognizing the vehicle on the road by its static identifier. Such fabricated information can lead to accidents which can jeopardize the lives of the passengers (Figure 9).

5 Proposed Solution: Privacy-aware HIP (P-HIP)

In this section, first, we provide an overview of P-HIP. Next, we present the details of our proposed authentication and key exchange schemes.

We propose an authentication method based on the ECQV cryptography that does not require HIP peers to exchange large-sized certificates, such as X.509 certificates, as a proof of the authenticity of the host identities (HITs and public keys). Moreover, the authentication method allows HIP hosts to verify the authenticity and integrity of host identifiers without performing signature and hash operations. Hence, resource-limited HIP hosts are unburdened from communication and computation overheads.

In the proposed scheme, a Certificate Authority (CA) issues ECQV credentials to HIP hosts. The hosts use the ECQV credentials and ECC to compute their private (d_i) and public (Q_i) keys (see Section 5.1). During authentication, two communicating HIP peers exchange their public keys and ECQV credentials. A peer uses the public key of the CA and its counterpart's ECQV cardinal to validate the authenticity of the public key (see Section 5.2) and ensure that the counterpart possesses the corresponding private key (see Section 5.3).

After successful authentication, HIP peers use the ECDH key exchange method to establish a session key to protect communications. We provide a solution that enables HIP peers to use ephemeral public and private keys to compute the session without performing signature, encryption, and decryption operations. Section 5.3, provides the details of the computation of the session key.

To this end, we propose a scheme that enables an HIP host to compute ephemeral and unique public and private keys from sessions to sessions and networks to networks. An HIP host does not require to communicate to the CA to receive ECQV credentials to compute new public and private keys. The proposed scheme allows an HIP host to use a previously issued CA credentials to compute unique host identifiers (public and private keys). An HIP peer can still validate the authenticity of the newly computed host identifiers by using the CA's public key (see Section 5.4). The use of unique identifiers protects the session and location privacy of the HIP hosts.

5.1 Host Identity Computation

A CA issues ECQV credentials to an HIP host (an IoT device) which the host uses to compute its host identifiers. Device manufactures or service providers can maintain their own CAs or they can consider a well-known CA, such as VeriSign[1], Digicert[2], and GoDaddy[3], as the Root CA. If a CA is managed by a device manufacturer or service provider then it has to be an Intermediate CA. A well-known Root CA issues the certificate of an Intermediate CAs and the Intermediate CA issues ECAQ credentials to the HIP hosts. The Root CA has the authority to audit an Intermediate CA and revoke certificates and public keys. A host trusts the public key of a CA (a Root CA or an Intermediate CA).

In Figure 10, we present the details of the process preformed by a Host to compute its host identities, such as an ECC key pair (d_u, Q_u) and host identity tag (HIT_u). A client (an Initiator or a Responder) sends an ECC point R_u to the CA. The R_u is encrypted $(E_{Q_{ca}}(R_u))$ using the public key of the CA (Q_{CA}). The client uses the Elliptic Curve integrated encryption scheme [44] to encrypt R_u. The CA receives $E_{Q_{ca}}(R_u)$ and decrypts it using its private key d_{ca}, and issues a public key construction data s and a unique identifier construction data δ to the client. The CA encrypts s and δ using a shared key $\alpha = R_u * d_{ca} = r_u * d_{ca} * G$ before sending them to the client. On receipt of the encrypted parameters $E_\alpha(s, \delta)$, the client computes the shared key α, decrypts $E_\alpha(s, \delta)$, and verifies $MAC_\alpha(s, \delta)$. Next, the client computes a private key $d_u = r_u + s \ (mod \ n)$ and public key $Q_u = d_u * G$. The client also computes a public key validation data $P_u = R_u + \delta$ and a hash of Q_u as its Host identity tag $HIT_u = Hash(Q_u)$.

To compute s, we propose an approach (Figure 10) which is different from the conventional ECQV implicit certificate scheme (Figure 3). From Figure 10, it can be noted that the CA does not issue a certificate (ICert) to the client; therefore, we do not multiply e, which is computed as $e = Hash(ICert)$, with k, to compute s (see the differences between Step-7 of Figure 10 and Step-8 of Figure 3). We compute s as $s = k + d_{CA} \ (mod \ n)$. Hence, we skip Step 6 and 7 of Figure 3. It can be also observed that, unlike Step-11 of Figure 3, we do not multiply e with r_u to compute d_u (Step-8 of Figure 10). Moreover, in our proposed scheme, the s and δ are encrypted and then sent to the client. These parameters are stored in the client's memory and used to compute unique identities for different networks.

There are two reasons for which we do not use e to compute s and d_u. First, a verifier does not have to compute a hash and perform

1. https://www.verisign.com/
2. https://www.digicert.com/
3. https://www.godaddy.com/

a multiplication operation while it validates a public key (see Section 5.2). Thus, we reduce computation overheads for public key validation. Second, an Initiator or a Responder can compute unique public keys and host identity tags for every network it joins without communicating with the CA to receive a new ECQV-based credential s and δ (see Section 5.4). Hence, mobile IoT devices can use unique host identities when they move from one network to another network and avoid identity tracking.

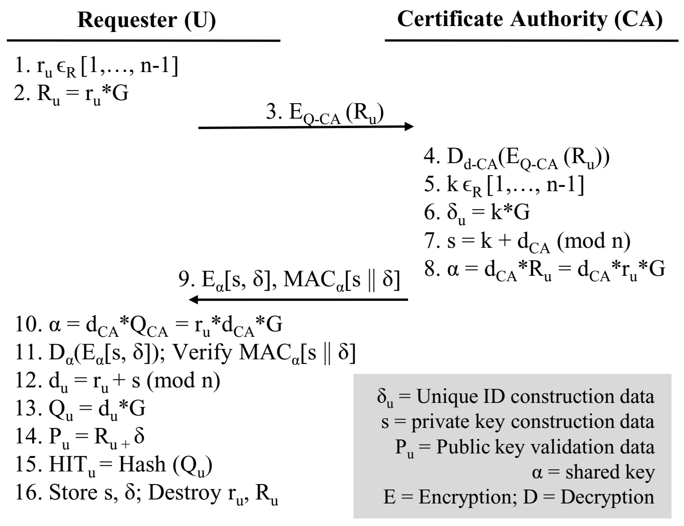

Requester (U)
Certificate Authority (CA)
1. r_u ϵ_R [1,…, n-1]
2. R_u = r_u*G
3. E_Q-CA (R_u)
4. D_d-CA(E_Q-CA (R_u))
5. k ϵ_R [1,…, n-1]
6. δ_u = k*G
7. s = k + d_CA (mod n)
8. α = d_CA*R_u = d_CA*r_u*G
9. E_α[s, δ], MAC_α[s ‖ δ]
10. α = d_CA*Q_CA = r_u*d_CA*G
11. D_α(E_α[s, δ]); Verify MAC_α[s ‖ δ]
12. d_u = r_u + s (mod n)
13. Q_u = d_u*G
14. P_u = R_u + δ
15. HIT_u = Hash (Q_u)
16. Store s, δ; Destroy r_u, R_u
δ_u = Unique ID construction data
s = private key construction data
P_u = Public key validation data
α = shared key
E = Encryption; D = Decryption

Fig. 10: Proposed ECQV implicit certificate scheme

5.2 Host Identity Validation

This section presents the procedure to validate host identifiers, such as public keys and host identity tags. Unlike the conventional ECQV public key validation scheme as shown in Figure 4, an HIP host does not perform hash and multiplication operations (see the differences between Step-3 of Figure 11 and Step-3 and Step-4 of Figure 4) to verify the authenticity of a public key and host identity tag. Our approach to validate host identities is shown in Figure 11. A prover device provides its ECQV public key (Q_p) and the public key validation parameter (P_p) to a verifier device. The verifier computes a public key Q'_P as $Q'_P = P_p + Q_{CA}$. If Q'_P matches Q_p then the verifier ensures that the public key is not forged and issued by the CA.

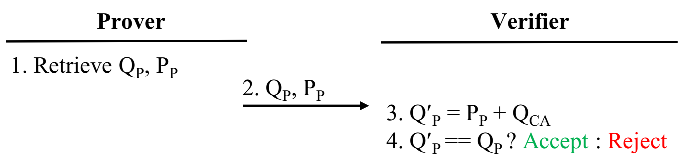


Fig. 11: Proposed approach to validate host identities.

Correctness of Host Identity Validation: A prover computes it ECC key pair (d_p, Q_p) as $Q_p = d_p * G$ (Step 13 of Figure 10). For completeness, we prove that the operation $P_p + Q_{CA}$ results in $d_p * G$, although we eliminate Step 6 and Step 7 of Figure 3, and the multiplication operation of Step 8 of Figure 3. The correctness of the proposed algorithm to validate an ECQV public key is as follows:

$$\begin{aligned}
Q_p &= P_p + Q_{CA} && (1)\\
&= R_p + \delta + Q_{CA} && \text{(from Step 14 of Figure 10)}\\
&= R_p + k * G + d_{CA} * G && \text{(from Step 6 of Figure 10)}\\
&= r_P * G + (k + d_{CA}) * G && (2)\\
&= (r_p + s) * G && \text{(from Step 7 of Figure 10)}\\
&= d_p * G && (3)
\end{aligned}$$

5.3 Mutual Authentication

We present the proposed mutual authentication scheme in Figure 12. The CA issues (s_i, δ_i) and (s_r, δ_r) to the Initiator and Responder respectively. The initiator computes its ECC pair (d_i, Q_i) as $d_i = r_i + s_i \ (mod\ n)$ and $Q_i = d_i * G$, and identity validation parameter $P_i = r_i * G + \delta_i$. Similarly, the Responder computes its ECC pair (d_r, Q_r) as $d_r = r_r + s_r \ (mod\ n)$ and $Q_r = d_r * G$, and identity validation parameter $P_r = r_r * G + \delta_r$. The details of the authentication scheme are presented below.

5.3.1 Authentication Steps

Step 1: The Initiator sends its HIT_i and the Responder's HIT_r in an I1 message.

Step 2: The Responder computes its ECDH key pair (d_{ecdh_r}, Q_{ecdh_r}). In the conventional ECDH scheme as describes in Section 3.4, a Responder selects its ECDH private key d_{ecdh_r} randomly as $d_{ecdh\_r} \in_R [1, \dots, n-1]$ and then the computes ECDH public key $Q_{ecdh\_r} = d_{ecdh\_r} * G$. Next, the Responder signs Q_{ecdh_r} and a puzzle using its private key d_r. The Responder uses the ECDSA signature algorithm to sign Q_{ecdh_r}. However, in our proposed scheme, we unburden HIP peers from the ECDSA signature computation and validation operations. The Responder computes an ECDH key pair (d_{ecdh_r}, Q_{ecdh_r}) using the ECQV public key computation scheme as shown in Algorithm 1. The Responder provides its private key d_r, HIT_r and a puzzle as inputs to Algorithm 1. The Algorithm 1 computes d_{ecdh_r}, Q_{ecdh_r}, and P_{ecdh_r}. The P_{ecdh_r} is used by the Initiator to verify that Q_{ecdh_r} is issued by the Responder. Algorithm 2 presents the verification process.

Algorithm 1: ECDH Key Pair Computation

```
function Compute_ECDH_KeyPair ($HIT_{ir}$, $d_{ir}$, PuzSol )
Input  : $HIT_{ir}$ ⟶ Initiator/responder Host Identity Tag
Input  : $d_{ir}$ ⟶ Private key of initiator or responder
Input  : PuzSol ⟶ Puzzle or Solution
Output : ($d_{ecdh\_ir}$,$Q_{ecdh\_ir}$) ⟶ ECDH key pair
Output : $P_{ecdh\_ir}$ ⟶ Parameter for $Q_{ecdh\_ir}$ validation
if $HIT_{ir}$ equals NULL or PuzSol equals NULL then
    return NULL
else
    $r_{ecdh\_ir}$ = RandBetween [1, n-1]
    $P_{ecdh\_ir} = r_{ecdh\_ir} * G$
    $e_{ecdh\_ir}$ = Hash ($HIT_{ir}$ || PuzSol || $P_{ecdh\_ir}$)        (4)
    $d_{ecdh\_ir} = e_{ecdh\_ir} * r_{ecdh\_ir} + d_{ir} \ (mod\ n)$
    $Q_{ecdh\_ir} = d_{ecdh\_ir} * G$
    return {($d_{ecdh\_ir}$,$Q_{ecdh\_ir}$), $P_{ecdh\_ir}$}
end
```

Step 3: The Responder sends an R1 message to the Initiator. The message contains HIT_i, HIT_r, a puzzle, and the Responser's ECC public key Q_r and ECDH public key Q_{ecdh_r}. The message

Algorithm 2: ECDH Public Key Validation

```
function Validate_ECDH_PublicKey ($HIT_{ir}$, PuzSol, $Q_{ir}$, $Q_{ecdh\_ir}$, $P_{ecdh\_ir}$)
Input  : $HIT_{ir}$ ⟶ Host Identity Tag of initiator or responder
Input  : PuzSol ⟶ Puzzle or Solution
Input  : $Q_{ir}$ ⟶ Public key of initiator or responder
Input  : $P_{ecdh\_ir}$ ⟶ Parameter for $Q_{ecdh\_ir}$ validation
if HIT equals NULL or PuzSol equals NULL then
    return NULL
else
    $e_{ecdh\_ir} \longleftarrow$ Hash ($HIT_{ir}$ || PuzSol || $P_{ecdh\_ir}$)
    $Q'_{ecdh\_ir} \longleftarrow e_{ecdh\_ir} * P_{ecdh\_ir} + Q_{ir}$
    if $Q'_{ecdh\_ir}$ equals $Q_{ecdh\_ir}$ then
        return Authentic
    else
        return Forged
    end
end
```

also includes a public key reconstruction data P_r and P_{ecdh_r} to validate Q_r and Q_{ecdh_r} respectively.

Step 4: The Initiator validates the Responder's public key Q_r and host identity tag HIT_r, and ensures that the identities are issued by the CA.

Step 5: In this step, the initiator authenticates the Responder. The Initiator provides the puzzle, HIT_r, Q_r, Q_{ecdh_r}, and P_{ecdh_r} as inputs to Algorithm 2 to validate that the Q_{ecdh_r} is computed using the Responder's private key d_r. Thus, the Initiator confirms that the Responder possesses the private key d_r of the ECC pair (d_r,Q_r). Section 5.3.2 presents the correctness of the ECDH public key validation method.

Step 6: The Initiator solves the puzzle. Next, the Initiator computes its ECDH key pair (d_{ecdh_i}, Q_{ecdh_i}) and the public validation parameter P_{ecdh_i} using Algorithm 2. The Initiator provides its private key d_i, host identity tag HIT_i, and the solution of the puzzle as the inputs to Algorithm 2 to compute d_{ecdh_i}, Q_{ecdh_i}, and P_{ecdh_i}.

Step 7: The Initiator computes an ECDH session key as $K_{ir} = d_{ecdh\_i} * Q_{ecdh\_r} = d_{ecdh\_i} * d_{ecdh\_r} * G$. Next, the Initiator selects a nonce $N_i \in_R [1,...,n-1]$ and computes a message authentication code (MAC) of N_i using the K_{ir} as $MAC_{K_{ir}}(N_i)$.

Step 8: The Initiator sends an I2 message to the Responder. The I2 message includes HIT_i, HIT_r, the solution of the puzzle, the Initiator's ECC public key Q_i and ECDH public key Q_{ecdh_i}, the parameters P_i and P_{ecdh_i} to validate Q_i and Q_{ecdh_i} respectively, the nonce N_i, and the $MAC_{K_{ir}}(N_i)$.

Step 9–10: The Responder, first, verifies the authenticity of Q_i and HIT_i. Next, it validates the Q_{ecdh_i}. The Responder provides the solution, HIT_i, Q_i, Q_{ecdh_i}, and P_{ecdh_i} as inputs to Algorithm 2 to verify that the Q_{ecdh_i} is computed using the initiator's private key d_i. Thus, the Responder authenticates the initiator.

Step 11: The Responder computes the ECDH session key as $K_{ir} = d_{ecdh\_r} * Q_{ecdh\_i} = d_{ecdh\_r} * d_{ecdh\_i} * G$. The Responder selects a nonce $N_r \in_R [1,...,n-1]$ and computes $MAC_{K_{ir}}(N_i||N_r)$.

Step 12–13: The Responder replies with an R2 message that contains HIT_i, HIT_r, N_b, and $MAC_{K_{ir}}(N_i||N_r)$. The initiator verifies the correctness of $MAC_{K_{ir}}(N_i||N_r)$ using K_{ir}, and confirms that both parties computed the session key correctly.

5.3.2 *Correctness of ECDH Public Key Validation*

Here, we present the completeness of Algorithm 2. We show that $d_{ecdh\_ir} * G$ can be derived from the operation $e_{ecdh\_ir} * P_{ecdh\_ir} + Q_{ir}$ (Algorithm 2). Hence, we prove that Q_{ecdh_ir} corresponds to d_{ecdh_ir} and is computed using the private key (d_{ir}) of the Initiator or Responder.

$$\begin{aligned}
Q_{ecdh\_ir} &= d_{ecdh\_ir} * G \\
&= (e_{ecdh\_ir} * r_{ecdh\_ir} + d_{ir}) * G \qquad \text{(from Equation 4)} \\
&= e_{ecdh\_ir} * (r_{ecdh\_ir} * G) + d_{ir} * G \\
&= e_{ecdh\_ir} * P_{ecdh\_ir} * + d_{ir} * G \\
&= e_{ecdh\_ir} * P_{ecdh\_ir} * + Q_{ir}
\end{aligned}$$

5.4 Unique Host Identity Computation

Let's suppose the Initiator moves from network n to network $n+1$. While the Initiator operates on network n, the Initiator's ECC pair (d_i^n, Q_i^n) and host identifiers are Q_i^n and HIT_i^n. When the Initiator moves to the network $n+1$, We propose a scheme that enables the Initiator to compute a new ECC pair (d_i^{n+1}, Q_i^{n+1}) and host identity tag (HIT_i^{n+1}) without communicating to the CA. The Initiator computes the new host identities when it moves to a different network or wants to update its host identities. The Initiator computes the new private key d_i^{n+1} and public key Q_i^{n+1} using its ECQV credentials s_i and δ_i as follows:

$$r_i^{n+1} = \text{RandomBetween}(1,\ n-1) \tag{5}$$

$$R_i^{n+1} = r_i^{n+1} * G \tag{6}$$

The new ECC pair (d_i^{n+1}, Q_i^{n+1}) is computed as below:

$$d_i^{n+1} = r_i^{n+1} + s_i \ (mod\ n) \tag{7}$$

$$Q_i^{n+1} = d_i^{n+1} * G \tag{8}$$

The new public key validation data P_i^{n+1} is computed as:

$$P_i^{n+1} = R_i^{n+1} + \delta_i \tag{9}$$

A responder validates the public key Q_i^{n+1} as follows:

$$Q_i^{(n+1)'} = P_i^{n+1} + Q_{CA} \tag{10}$$

$$Q_i^{(n+1)'} == Q_i^{n+1} \ ?\ accept\ :\ reject \tag{11}$$

For completeness of Equation 10, we now prove that Q_i^{n+1} is the public key for the private key d_i^{n+1}, which justifies that Q_i^{n+1} is issued by the CA and the Initiator can use Q_i^{n+1} as a legitimate public key:

$$\begin{aligned}
Q_i^{(n+1)} &= P_i^{n+1} + Q_{CA} \\
&= r_i^{n+1} * G + \delta_i + Q_{CA} \qquad \text{(from Equation 9)} \\
&= r_i^{n+1} * G + k * G + Q_{CA} \qquad \text{(from Step 14 Fig 10)} \\
&= r_i^{n+1} * G + k * G + d_{CA} * G \\
&= r_i^{n+1} * G + (k + d_{CA}) * G \\
&= r_i^{n+1} * G + s_i * G \qquad \text{(from Step 7 Fig 10)} \\
&= (r_i^{n+1} + s_i) * G \\
&= d_i^{n+1} * G \qquad \text{(from Equation 7)}
\end{aligned}$$

6 Security Analysis of P-HIP

In this section, we show that P-HIP is secure against various network attacks. We first consider a set of security properties, and then provide an analysis that shows P-HIP can ensure these properties under various security threats. Table 2 presents the summary of the security properties.

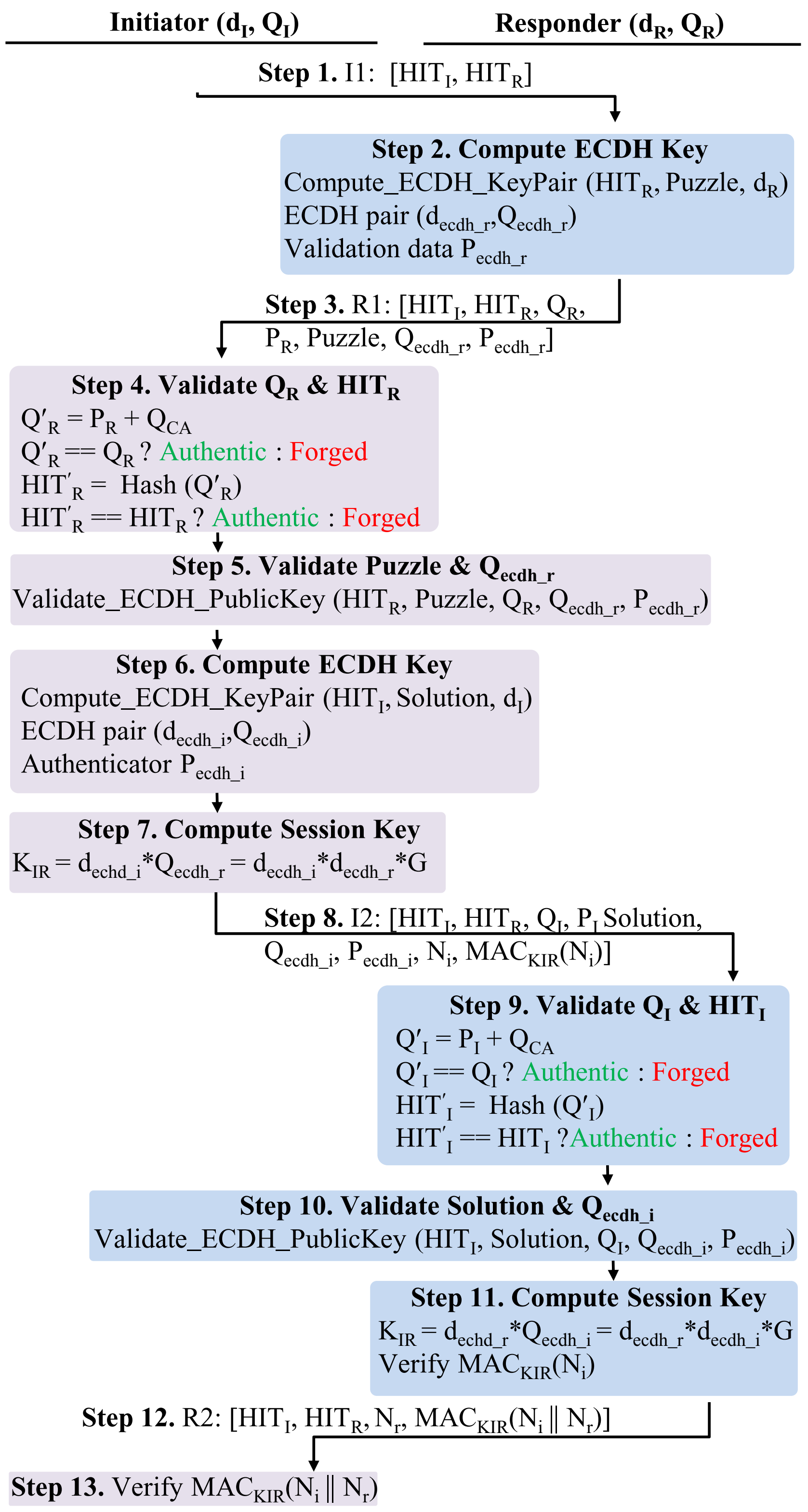

Initiator (d_I, Q_I)
Responder (d_R, Q_R)
Step 1. I1: [HIT_I, HIT_R]
Step 2. Compute ECDH Key
Compute_ECDH_KeyPair (HIT_R, Puzzle, d_R)
ECDH pair (d_{ecdh_r},Q_{ecdh_r})
Validation data P_{ecdh_r}
Step 3. R1: [HIT_I, HIT_R, Q_R, P_R, Puzzle, Q_{ecdh_r}, P_{ecdh_r}]
Step 4. Validate Q_R & HIT_R
$Q'_R = P_R + Q_{CA}$
$Q'_R == Q_R$? Authentic : Forged
$HIT'_R =$ Hash (Q'_R)
$HIT'_R == HIT_R$? Authentic : Forged
Step 5. Validate Puzzle & Q_{ecdh_r}
Validate_ECDH_PublicKey (HIT_R, Puzzle, Q_R, Q_{ecdh_r}, P_{ecdh_r})
Step 6. Compute ECDH Key
Compute_ECDH_KeyPair (HIT_I, Solution, d_I)
ECDH pair (d_{ecdh_i},Q_{ecdh_i})
Authenticator P_{ecdh_i}
Step 7. Compute Session Key
$K_{IR} = d_{echd\_i}*Q_{ecdh\_r} = d_{ecdh\_i}*d_{ecdh\_r}*G$
Step 8. I2: [HIT_I, HIT_R, Q_I, P_I Solution, Q_{ecdh_i}, P_{ecdh_i}, N_i, $MAC_{KIR}(N_i)$]
Step 9. Validate Q_I & HIT_I
$Q'_I = P_I + Q_{CA}$
$Q'_I == Q_I$? Authentic : Forged
$HIT'_I =$ Hash (Q'_I)
$HIT'_I == HIT_I$?Authentic : Forged
Step 10. Validate Solution & Q_{ecdh_i}
Validate_ECDH_PublicKey (HIT_I, Solution, Q_I, Q_{ecdh_i}, P_{ecdh_i})
Step 11. Compute Session Key
$K_{IR} = d_{echd\_r}*Q_{ecdh\_i} = d_{ecdh\_r}*d_{ecdh\_i}*G$
Verify $MAC_{KIR}(N_i)$
Step 12. R2: [HIT_I, HIT_R, N_r, $MAC_{KIR}(N_i \| N_r)$]
Step 13. Verify $MAC_{KIR}(N_i \| N_r)$

Fig. 12: Proposed ECQV authentication for HIP

6.1 Trustworthy ECDH Public Value

The HIP hosts trust the CA. The CA issues private key computation data (s) to a host. The host computes an ECC key pair using s. Next, the host proves to a verifier that it possesses the private key of the key pair that is computed using s. Hence, the verifier trusts the host. The verifier also accepts any ECC key pair as authentic that is computed using the private key of the host. For instance, the ECDH private value d_{ecdh_r} of a Responder is computed using the Responder's private key d_r (Algorithm 1). An Initiator first verifies the authenticity of the Responder's public key Q_r (Step 4 of Figure 12). Next, it validates the Responder's ECDH pubic value Q_ecdh_r (Step 5 of Figure 12). Thus, the Initiator avoids signature

TABLE 2: Security Properties.

Security Property	Attack Scenario	Requirements
Trustworthy host identifiers	An adversary uses forged identities for authentication.	HIP hosts should be able to identify forged identities.
Identity privacy	An adversary is provided host identifiers, which are used by a target host in various networks or locations, for movement profiling, communication relation, location recording, and so on.	The adversary cannot determine whether the host identifiers belong to the same host.
Credential freshness	An adversary stores messages (I1, I2, R1, and R2) exchanged between HIP hosts and then retransmit the messages to trick a target host into false identification or authentication.	The target host should be able to identify replayed messages.
Identity impersonation mitigation	An adversary can alter the credentials of I1, R1, I2 and R2 messages exchanged between two parties to impersonate a legitimate peer. Thus, the adversary tricks the peers into believing that they are directly communicating with each other.	HIP peers should be able to identify such credential fabrication.
Communication protection	An adversary can eavesdrop on the communication channel to learn the contents of exchanged messages.	Communicating peers should preserve the privacy and confidentiality of the communications.
Forward Security	An adversary can use compromised credentials, such as session keys, to learn past or future communications.	The adversary cannot use the leaked credentials to compute session keys used in the past or will be used in the future communications.

validation operations required to authenticate the Responder, and ensures that Q_{ecdh_r} corresponds to d_{ecdh_r} which is computed using d_r. Note that the Responder computes d_r using the private key computation data s_r issued by the CA (Step 12 of Figure 10).

6.2 Identity Privacy

Let's suppose, an adversary is provided with identities (P_1, Q_1), (P_2, Q_3),..., (P_{n-1}, Q_{n-1}), (P_n, Q_n) of an HIP host that were used in n number of networks. The identities were computed as follows (see Section 5.4):

$$\begin{aligned} P_1 &= R_1 + \delta & Q_1 &= R_1 + \mu \\ P_2 &= R_2 + \delta & Q_2 &= R_2 + \mu \\ &\vdots & &\vdots \\ P_{n-1} &= R_{n-1} + \delta & Q_{n-1} &= R_{n-1} + \mu \\ P_n &= R_n + \delta & Q_n &= R_n + \mu \end{aligned}$$

The goal of the adversary is to infer that the identifiers belong the same host. To find a mapping between two public keys Q_i and Q_j the adversary must know either R_i or R_j, and δ and μ, such that $1 \leq i, j \leq n \ \wedge i \neq j$, $R_j = r_j * G$, $\delta = k * G$, and $\mu = s * G$. If the adversary learns R_i then it finds δ and μ as follows:

$$\delta = P_i - R_i$$
$$\mu = Q_i - R_i$$

The adversary uses δ and μ to compute R_j and Q_j as follows.

$$R_j = P_j - \delta$$
$$Q_j = R_j + \mu$$

Hence, the adversary ensures that host identities Q_1, Q_3, ..., Q_{n-1}, Q_n belong to the same HIP host. However, an adversary cannot learn the keys $R_1, R_2, .., R_{n-1}$ because the key R_1 is encrypted using the public key of the CA (Step 3 of Figure 10) and the $R_2, .., R_{n-1}$ are destroyed right after the computation of d_k (Equation 7) and P_k (Equation 9) such that $2 \leq k \leq n$. Therefore, we can conclude that the adversary cannot learn that the identities (P_1, Q_1), (P_2, Q_3),..., (P_{n-1}, Q_{n-1}), (P_n, Q_n) correspond to the same host.

6.3 Credential Freshness

6.3.1 Replay Initiator's Credentials

An adversary replays an Initiator's host identities, such as Q_i and HIP_i, that are included in an I1 message. The Responder replies with a R2 message. The adversary replays an I2 message in response. However, the Responder terminates the connection as it finds a replayed Nonce N_i (e.g., a timestamps) included in the I2 message (Step 9 of Figure 12). The adversary can use a recent Nonce N_i^n to fool the Responder. However, the adversary cannot compute a MAC for N_i^n as $MAC_{K_{ir}}(N_i^n)$ since it does not possess the session key K_{ir}. As such, the $MAC_{K_{ir}}(N_i^n)$ validation will fail on the Responder end, and the Responder will terminate the communications.

6.3.2 Replay Responser's Credentials

An adversary replays a R1 message to an Initiator. The Initiator replies with an I2 message that contains a new Nonce N_i^n and $MAC_{K_{ir}}(N_i^n)$. The initiator waits for a R2 message which includes a MAC computed as $MAC_{K_{ir}}(N_i^n||N_r^n)$ such that N_r^n is a nonce generated by the Responder for the current session (Step 12–13 of Figure 12). However, the adversary cannot compute $MAC_{K_{ir}}(N_i^n||N_r^n)$ as it does not possess the private key of the real Responder. As such, the adversary replays a R2 message. The Initiator terminates the communication as it finds a replayed nonce N_r included in the R2 message.

6.4 Identity Impersonation Mitigation

The Man-in-the-Middle (MITM) attack is a type of identity impersonation. In the MITM, the goal of an adversary is to establish a session key with an initiator or responder impersonating a real HIP host. The adversary replaces the credentials of the target host with its forged credentials, and tricks the target host into thinking that the adversary is the legitimate host.

6.4.1 Fabrication of Responder's Keying Materials

An adversary with an ECC key pair $(d_{adversary}, Q_{adversary})$ blocks the R1 message (Step 3 of Figure 12). Next, it computes an ECDH key pair $(d_{ecdh}^{adversary}, Q_{ecdh}^{adversary})$ by providing $d_{adversary}$, $puzzle$, and HIT_r as inputs to Algorithm 1. The adversary replaces Q_{ecdh_r}, and P_{ecdh_r} with its own $Q_{ecdh}^{adversary}$ and $P_{ecdh}^{adversary}$ respectively. To this end, the adversary constructs a malicious R2 message with the forged keying materials, and sends it to the Initiator. However, the

Initiator provides HIT_r, Q_r, $Q_{ecdh}^{adversary}$, and $P_{ecdh}^{adversary}$ as the inputs to Algorithm 2 (Step 5 of Figure 12), and finds that $Q_{ecdh}^{adversary}$ is forged. Thus, the Initiator ensures that $Q_{ecdh}^{adversary}$ was not computed using the private key d_r of the Responder. The adversary can also replace the Responder's keying materials Q_r and P_r with its keying materials $Q_{adversary}$ and $P_{adversary}$ respectively. However, the Initiator identifies this forgery when it validates the HIT_r of the Responder as $HIT_r == Hash(Q_{adversary})\ ?\ authentic : forged$ (Step 4 of Figure 12). Note that the adversary can forge the host HIT_r of the Responder. However, the HIP layer of the Initiator discards this packet immediately as it finds the forged $HIT_r^{adversary}$ which was not sent in the I1 message (Section 4.4 of RFC [15]).

6.4.2 *Fabricating of Initiator's Keying Materials*

The adversary replaces the Initiator's ECDH credentials Q_{ecdh_i} and P_{ecdh_i}, that are included in the I2 message, with its forged cryptographic keys $Q_{ecdh}^{adversary}$ and $P_{ecdh}^{adversary}$ respectively. The Responder identifies this forgery as the ECDH key validation fails in Step 10 of Figure 12. The Responder computes Q'_{ecdh_r} using $P_{ecdh}^{adversary}$ and the public key of the Initiator Q_i. However, the Responder finds that Q'_{ecdh_r} does not match the forged key $Q_{ecdh}^{adversary}$. Hence, the Responder ensures that $Q_{ecdh}^{adversary}$ was not computed using the Initiator's private key d_i. If the adversary forges Q_i and P_i then the Responder identifies this forgery in Step 9 of Figure 12.

6.5 End-to-End Communication Protection

Communicating HIP peers perform the ECDH key exchange to establish a session key (Step 7 and Step 11 of Figure 12). The peers encrypt messages using the session key. Therefore, adversaries cannot learn the communications that take place between the peers.

6.6 Forward Security

The cryptographic materials that are used to compute a shared key is generated randomly for every session. An Initiator selects an ECDH key pair (d_{ecdh_i}, Q_{ecdh_i}) such that it is different from one session to another session (Step 6 of Figure 12). A Responder also computes its ECDH key pair (d_{ecdh_r}, Q_{ecdh_r}) randomly (see Step 2 of Figure 12 and Algorithm 1). As such, if the shared key of one session is leaked then the communications of that session is revealed only. An adversary cannot use the leaked session key to learn or derive the shared keys of other sessions because these keys are generated using session independent and random ECDH key pairs.

7 Experiment

7.1 Experimental Setup

We implemented a prototype of P-HIP for RE-Mote [45] IoT devices powered by the Contiki [46] operating system. Figure 13 shows the experimental setup. The hardware specification of a RE-Mote is presented in Table 3. We created an IoT network using two RE-Mote devices and a Weptech [33] border router. The Weptech gateway served as a bridge between an IPv4 and IPv6 network. The RE-Motes operated on the IPv6 network. The Weptech gateway was equipped with two network interface cards: Ethernet and IEEE 802.15.4 radio transceiver. The radio interface enabled communications between the RE-Mote devices, while the Ethernet interface connected the IoT network to the Internet. We developed a UDP server and hosted it on a RE-Mote device. The server RE-Mote exposed its resources as CoAP (Constrained Application Layer Protocol) [47] URLs, such as coap://aaaa::212:4701:101/responder/res. The other RE-Mote acted as a CoAP client and interacted with the server device using the CoAP URL. We implemented the HIP authentication methods in Contiki. We designated the server node as a Responder and the client node as an Initiator.

TABLE 3: RE-Mote [45] device specification.

Component	Description
CPU	ARM Cortex-M3
Speed	32 MHz
RAM	32 KB
Flash	512 KB
Transceiver	IEEE 802.15.4
Radio Bandwidth	250 kbps

In our experimental scenarios, we considered both the Mesh-under and Route-over routing as the packet forwarding mechanism. We used the Relic [48] library to perform cryptographic operations. In the experiment, the RE-Mote devices exchanged X.509 self-signed certificates as the proof of their hots identifiers. We created an RSA and ECC self-signed X.509 certificates using the OpenSSL library [38]. The lengths of the RSA and ECC keys were 1024 bits and 160 bits respectively.

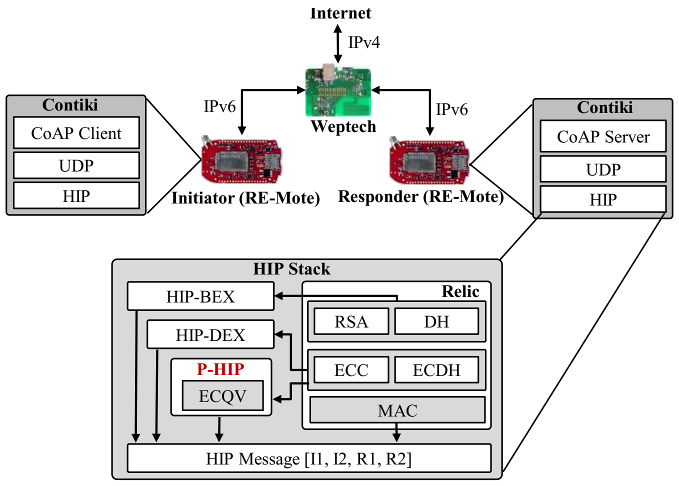


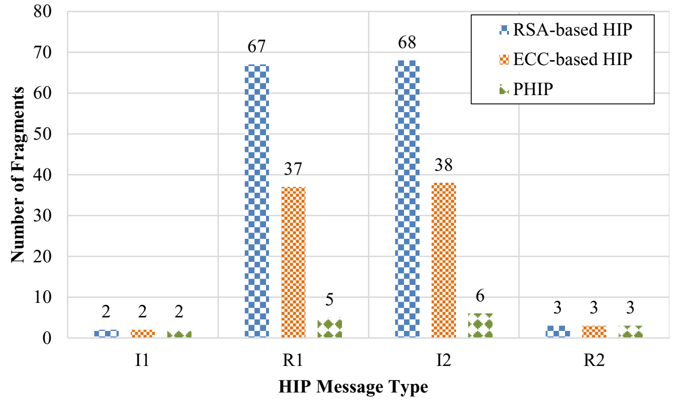
Fig. 13: Experimental Setup.

7.2 Evaluation

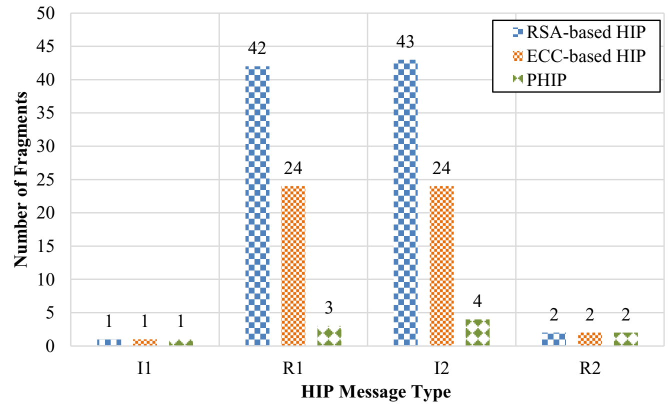
We analyzed the performance of P-HIP regarding communication, computation, and energy costs for mutual authentication and session key establishment. We compared the resource efficiency of P-HIP with the RSA based authentication used in HIP-BEX [15] and ECC-based authentication used in HIP-DEX [18]. The collaborative authentication methods proposed in D-HIP [16] and HIP-TEX [17] were not considered in our experimental scenarios because these schemes are not feasible for IoT networks that do not have resource-rich proxy nodes. Moreover, these schemes follow the HIP-BEX model in the IoT networks without proxy nodes. The LHIP [20] was not considered as it does not have supports for authentication and key exchange. The Slimfit [19] does not address the communication overhead for sending HIP credentials and computation overheads for the cryptographic operations. Instead, the Slimfit relies on the methods used in the HIP-BEX (HIP-RSA) and HIP-DEX (HIP-ECC) for the authentication. Therefore, we discarded the Slimfit from our experimental scenario. Table 4 provides the details of the HIP message (conents and sizes) used for the experimental evaluation. Two communicating peers perform the HIP handshake (I1 → R1 → I2 → R2) every time they exchange messages. The details of the evaluation of P-HIP are as follows.

Fragment Savings: We provided a comparison of the number of packet fragment both for the mesh-under and route-over routing in Figure 14. We calculated the total number of fragments (F_n) that are exchanged during the authentication as $F_n = \frac{\sum_{i=1}^{2} \text{sizeof}(I_i) + \sum_{j=1}^{2} \text{sizeof}(R_j)}{s}$. In this equation, the term s denotes the number of bytes available for an HIP payload in the IEEE 802.15.4 frame (Figure 2) – s is 46 bytes in route-over routing (Figure 2b) and 29 bytes in mesh-under routing (Figure 2c). From the Figures 14a and 14b, it can be observed that the F_n for the R1 and I2 message are significantly lower in P-HIP than in the HIP-BEX and HIP-DEX. P-HIP used ECQV based credentials (Q, P) as the proof of authenticity of public keys and host identity tags instead of X.509 certificates. The sizes of the ECQV credentials are much smaller than the certificates. Therefore, the total number of fragment was dropped significantly in P-HIP. The F_n for the I1 and R2 message are same because these messages contain information about the fixed sized host identifies (128 bits), nonces (16 bits), and MACs (128 bits).

In Figure 15, we presented the average fragment saving (F_s) in P-HIP. The F_s was calculated as $F_s = 1 - \frac{F_n^{P-HIP}}{F_n^a}$, such that $F_n^a = \frac{F_n^{HIP-BEX(rsa-based)} + F_n^{HIP-DEX(ecc-based)}}{2}$. From Figure 15, it can be observed that P-HIP could reduce the number of fragment for the R1 and I2 message by 91% and 89% respectively.



(a) Number of HIP packet fragments in Mesh-under routing.



(b) Number of HIP packet fragments in Route-over routing.

Fig. 14: A comparison of HIP packet fragments.

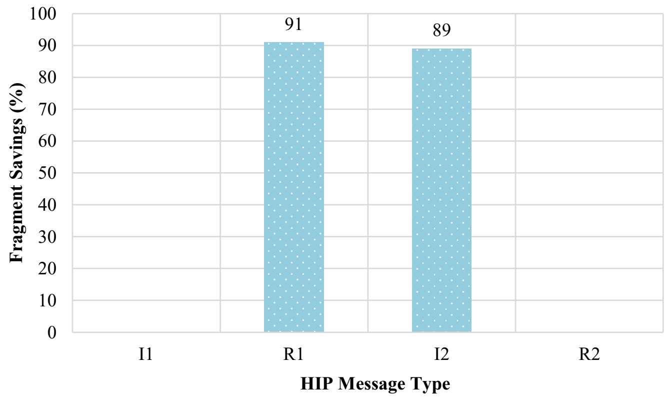
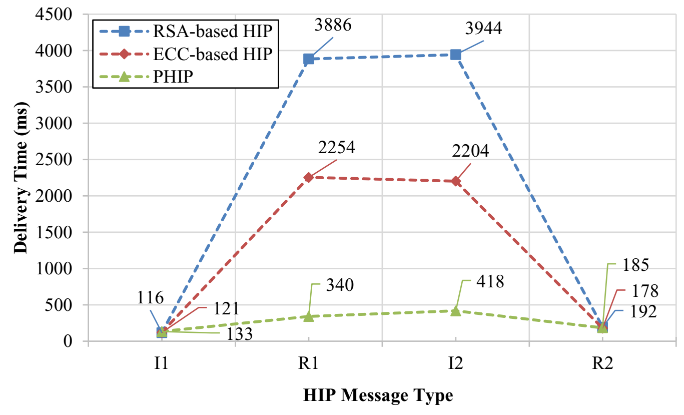
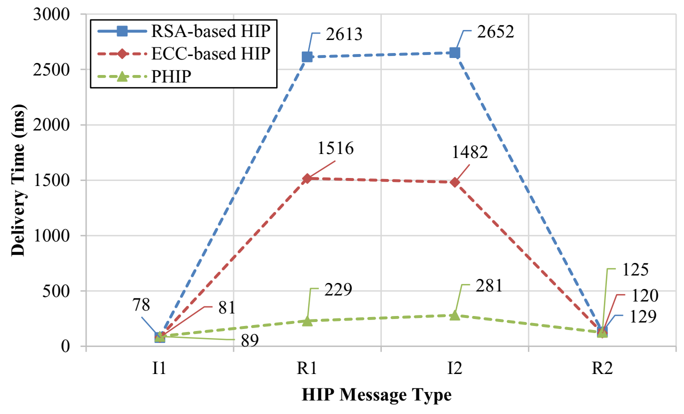


Fig. 15: Percentage of fragment savings in P-HIP.

Communication Latency: In Figure 16, we provided an analysis of message delivery delay. We recorded the time required to deliver HIP messages using the mesh-under and route-over routing separately, and showed the results in Figure 16a and 16b. We used the Contiki clock library [49] to measure the delays. From the figures, it can be observed that there was a notable reduction in the message delivery time in P-HIP. P-HIP minimized the delivery time of the R1 and I2 message by avoiding the exchange of X.509 certificates.



(a) Communication latency in Mesh-under routing.



(b) Communication latency in Route-over routing.

Fig. 16: A comparison of communication overheads.

Runtime: We provided an analysis of the runtime for cryptographic operations and fragment processing (message fragmentation and reassembly) in Figure 17. We recorded the runtime on the Initiator node. As shown in Figure 17a, P-HIP spent a significantly lower amount of time on fragment processing than HIP-BEX and HIP-DEX due to the exchange of small-sized R1

TABLE 4: The content and size of various types of message used in the HIP handshake. s = sender, r = receiver.

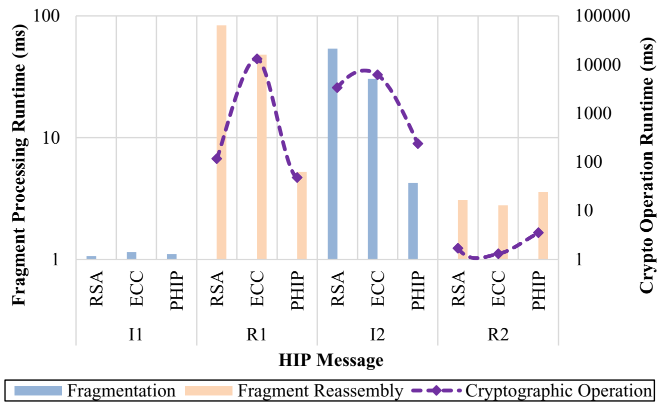
Message	Content			Size (Bits)			Total Size (Bytes)		
	HIP (RSA)	HIP (ECC)	P-HIP	HIP-RSA	HIP-ECC	P-HIP	HIP-RSA	HIP-ECC	P-HIP
I1	HIT$_s$, HIT$_r$	HIT$_s$, HIT$_r$	HIT$_s$, HIT$_r$	128 + 128	128 + 128	128 + 128	32	32	32
R1	HIT$_s$, HIT$_r$, Public Key, Puzzle, DH Key Exchange Param (discrete logarithm group), Signature (RSA), X.509 Certificate	HIT$_s$, HIT$_r$, Public Key, Puzzle, ECDH Key Exchange Param, Signature (ECDSA), X.509 Certificate	HIT$_s$, HIT$_r$, ECQV Cred (P, Q), Puzzle, ECDH Key Exchange Param (P$_{ecdh}$, Q$_{ecdh}$)	128 + 128 + 1024 + 64 + 3072 + 1024 + 9896	128 + 128 + 160 + 64 + 160 + 320 + 7584	128 + 128 + (160 + 160) + 64 + (160 + 160)	1917	1068	120
I2	HIT$_s$, HIT$_r$, Public Key, Solution, DH Key Exchange Param (discrete logarithm group), Signature (RSA), X.509 Certificate, Nonce, MAC	HIT$_s$, HIT$_r$, Public Key, Solution, ECDH Key Exchange Param, Signature (ECDSA), X.509 Certificate, Nonce, MAC	HIT$_s$, HIT$_r$, ECQV Cred (P, Q), Solution, ECDH Key Exchange Param (P$_{ecdh}$,Q$_{ecdh}$), Nonce, MAC	128 + 128 + 1024 + 64 + 3072 + 1024 + 9896 + 16 + 256	128 + 128 + 160 + 64 + 160 + 320 + 7584 + 16 + 256	128 + 128 + (160 + 160) + 64 + (160 + 160) + 16 + 256	1951	1102	154
R2	HIT$_s$, HIT$_r$, Nonce, MAC	HIT$_s$, HIT$_r$, Nonce, MAC	HIT$_s$, HIT$_r$, Nonce, MAC	128 + 128 + 16 + 256	128 + 128 + 16 + 256	128 + 128 + 16 + 256	66	66	66

and I2 messages. Moreover, there is a notable reduction in the time required to validate a host's identities (R1 message) and compute a session key (I2 message). The runtime for cryptographic operations was reduced because, unlike the HIP-BEX and HIP-DEX, P-HIP does not require HIP peers to perform modular exponentiation operations and signature verification to validate host identities and compute a session key.

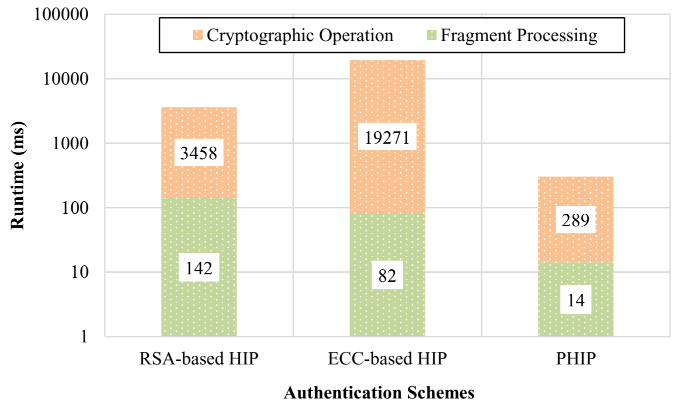
Figure 17b presents a comparison of total runtime for the authentication methods. The total runtime (r_t) was calculated as $r_t = r_i + r_c + r_r$. The terms r_i and r_r represent the time to fragment I message and reassemble R messages respectively. The term r_c is the sum of the time required to validate host identities and establish a session key. As shown in Figure 17, P-HIP is 12 and 64 times faster than the HIP-BEX and HIP-DEX respectively.

Energy Cost: We used the Contiki energy library [50] to measure the energy consumption of the CPU and radio transceiver. Figure 18 presents a comparison of the energy consumption of the radio transceiver for exchanging HIP messages. From the figure, it can be noted that, on average, P-HIP could reduce energy costs for communications by 85% and 82% in the mesh-under and route-over routing respectively.

The results of the energy consumption by a CPU for performing cryptographic operations are shown in Figure 19. From the figure, it can be noted that P-HIP is more energy efficient compared to the the HIP-BEX and HIP-DEX. On average, P-HIP could reduce the energy cost by 81%.



(**a**) Runtime analysis of HIP flights.



(**b**) Total runtime on an Initiator node.

Fig. 17: Runtime analysis.

8 Conclusion

In this article, we proposed P-HIP, a lightweight and privacy-preserving authentication method for the HIP-enabled IoT devices. P-HIP utilized the ECQV cryptography to reduce computation overheads for mutual authentication to make the HIP suitable for the resource-constrained IoT device with limited processing power. P-HIP eliminated the computation-intensive cryptographic operations, such as modular exponentiation, signature validation, and asymmetric encryption and decryption, involved in the authentication. P-HIP also eliminated the requirements for exchanging certificates, which are sent in a large number of fragments over a lossy link, as the poof of the authenticity of host identities, such as public keys and host identity tags. Hence, P-HIP unburdened IoT devices, which operate on the low data rate networks, from the communication overheads for certificate delivery and processing certificate fragments. P-HIP also provided a scheme to compute unique host identifiers without contacting a certificate authority. Thus, P-HIP enabled a mobile HIP host to compute legitimate

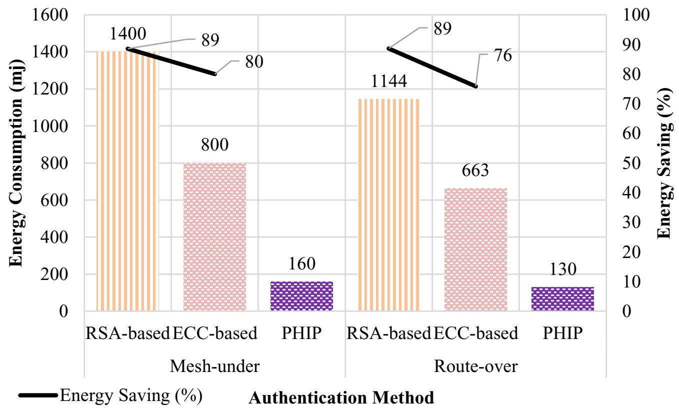
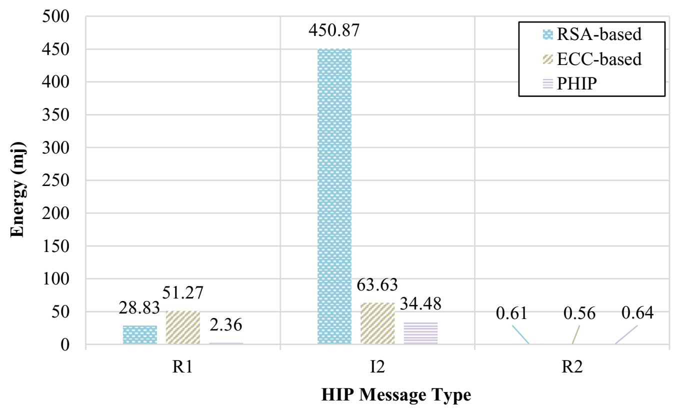
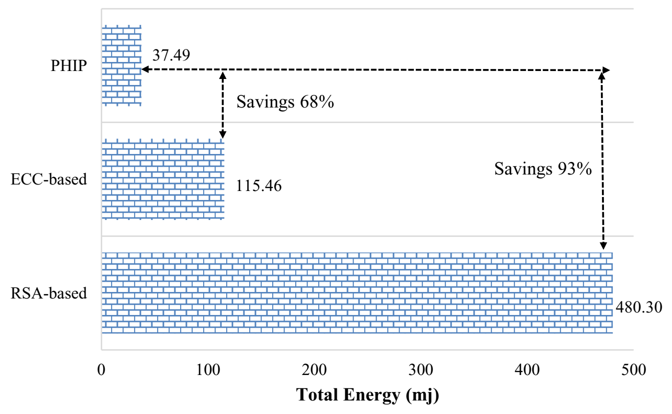


Fig. 18: Energy cost for communications



(a) Energy cost for cryptographic operations.



(b) Total energy consumption cryptographic operations.

Fig. 19: Energy cost for computations.

identifiers that are unique from networks to networks and sessions to sessions to protect its privacy. We presented a security analysis of P-HIP that showed P-HIP could provide security against identity tracking, replay, and man-in-the-middle attacks. We also provided a performance analysis of P-HIP which confirmed that P-HIP could minimize computation, communication, and energy costs for authentication.

Acknowledgment

Hasan was supported by the National Science Foundation through awards DGE-1723768, ACI-1642078, and CNS-1351038, and by the National Institutes of Health grant 1R21HD095270-01.

References

[1] R. Petrolo, V. Loscri, and N. Mitton, “Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms,” *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 1, 2017.

[2] C. Benevolo, R. P. Dameri, and B. D’Auria, “Smart mobility in smart city,” in *Empowering Organizations*. Springer, 2016, pp. 13–28.

[3] M. Hossain, S. R. Islam, F. Ali, K.-S. Kwak, and R. Hasan, “An internet of things-based health prescription assistant and its security system design,” *Future generation computer systems*, vol. 82, pp. 422–439, 2018.

[4] S. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K.-S. Kwak, “The internet of things for health care: a comprehensive survey,” *IEEE Access*, vol. 3, pp. 678–708, 2015.

[5] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of things: A survey on enabling technologies, protocols, and applications,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[6] R. Srinivasan, A. Sharmili, S. Saravanan, and D. Jayaprakash, “Smart vehicles with everything,” in *2nd International Conference on Contemporary Computing and Informatics (IC3I)*. IEEE, 2016, pp. 400–403.

[7] M. E. Berglund, J. Duvall, and L. E. Dunne, “A survey of the historical scope and current trends of wearable technology applications,” in *Proceedings of the 2016 ACM International Symposium on Wearable Computers*. ACM, 2016, pp. 40–43.

[8] Gartner, “5.8 billion enterprise and automotive iot endpoints will be in use in 2020,” Online at https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io, 2019.

[9] C. Perkins *et al.*, “Ip mobility support for ipv4,” 2002.

[10] D. Johnson, C. Perkins, J. Arkko *et al.*, “Mobility support in ipv6,” 2004.

[11] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos *et al.*, “Named data networking (ndn) project,” *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, vol. 157, p. 158, 2010.

[12] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, “Transmission of ipv6 packets over ieee 802.15.4 networks,” *IETF RFC 4944*, 2007.

[13] V. Pai, U. K. K. Shenoy *et al.*, “6lowpan—performance analysis on low power networks,” in *International Conference on Computer Networks and Communication Technologies*. Springer, 2019, pp. 145–156.

[14] V. Kumar and S. Tiwari, “Routing in ipv6 over low-power wireless personal area networks (6lowpan): A survey,” *Journal of Computer Networks and Communications*, vol. 2012, 2012.

[15] T. Heer, P. Jokela, and T. Henderson, “Host identity protocol,” *IETF RFC 7401*, 2015.

[16] Y. B. Saied and A. Olivereau, “D-hip: A distributed key exchange scheme for hip-based internet of things,” in *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2012, pp. 1–7.

[17] Y. Saied and A. Olivereau, “Hip tiny exchange (tex): A distributed key exchange scheme for hip-based internet of things,” in *Third International Conference on Communications and*

Networking, 2012, pp. 1–8.

[18] R. Moskowitz and R. Hummen, "Hip diet exchange (dex)," *draft-ietf-hip-dex-06*, 2017.

[19] R. Hummen, J. Hiller, M. Henze, and K. Wehrle, "Slimfit: A hip dex compression layer for the ip-based internet of things," in *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2013, pp. 259–266.

[20] T. Heer, "Lhip lightweight authentication extension for hip," *draft-heer-hip-lhip-00*, 2007.

[21] Zolertia, "Z1 mote iot device," 2016. [Online]. Available: http://zolertia.sourceforge.net/

[22] SkyMote, "T-Mote Sky Iot Device," 2016. [Online]. Available: http://wirelesssensornetworks.weebly.com/1/post/2013/08/tmote-sky.html

[23] APDM, "Opal sensor node," https://www.apdm.com/wearable-sensors/, 2016. [Online]. Available: http://www.net.in.tun.de/en/sandbox/wireless-sensor-networks/

[24] O. Mote, "Open hardware for the internet of things," http://openmote.com/product/openmote-b-platinum-kit/, 2016.

[25] Libelium, "Waspmote: The sensor device for internet of things developers," http://www.libelium.com/products/waspmote/, 2016.

[26] T. Heer and S. Varjonen, "Host identity protocol certificates," *IETF RFC 8002*, 2016.

[27] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[28] Weinzierl, "Kns stacks: A development board for knx applications," https://www.weinzierl.de/index.php/en/all-knx/knx-stacks-en/development-hardware-en, 2017.

[29] M. B. Yassein, W. Mardini, and A. Khalil, "Smart homes automation using z-wave protocol," in *International Conference on Engineering & MIS (ICEMIS)*. IEEE, 2016, pp. 1–6.

[30] D. Sturek, "Zigbee ip stack overview," *ZigBee Alliance*, vol. 2009, 2009.

[31] Arduino, "Arduino Uno: An IoT development board," https://store.arduino.cc/usa/arduino-uno-rev3, 2017.

[32] Arm-Mbed, "Mbed: A development board for rapid prototyping of iot applications," https://os.mbed.com/platforms/mbed-LPC1768/, 2017.

[33] Weptech, "A 6LoWPan Border Router," https://www.weptech.de/6LoWPAN_IoT_Gateway_EN.html, 2017.

[34] Certicom, "Explaining implicit certificates," Certicom, Tech. Rep., 2014, online at https://www.certicom.com/content/certicom/en/code-and-cipher/explaining-implicit-certificate.html.

[39] S. Raza, S. Duquennoy, J. Höglund, U. Roedig, and T. Voigt, "Secure communication for the internet of things—a compar-

[35] E. Rescorla, "Diffie-hellman key agreement method," *IETF RFC 2631*, 1999.

[36] R. Haakegaard and J. Lang, "The elliptic curve diffie-hellman (ecdh)," Online at https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Haakegaard+Lang.pdf, 2015.

[37] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

[38] J. Viega, M. Messier, and P. Chandra, *Network Security with OpenSSL: Cryptography for Secure Communications*. O'Reilly Media, Inc, 2002.

ison of link-layer security and ipsec for 6lowpan," *Security and Communication Networks*, vol. 7, no. 12, pp. 2654–2668, 2014.

[40] C. Lin, K. Liu, B. Xu, J. Deng, C. W. Yu, and G. Wu, "Vclt: An accurate trajectory tracking attack based on crowdsourcing in vanets," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2015, pp. 297–310.

[41] J. Hua, Z. Shen, and S. Zhong, "We can track you if you take the metro: Tracking metro riders using accelerometers on smartphones," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 2, pp. 286–297, 2017.

[42] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *USENIX Security Symposium*, 2017.

[43] A. Di Maio, M. R. Palattella, R. Soua, L. Lamorte, X. Vilajosana, J. Alonso-Zarate, and T. Engel, "Enabling sdn in vanets: What is the impact on security?" *Sensors*, vol. 16, no. 12, p. 2077, 2016.

[44] M. Campagna, "Sec 4: Elliptic curve qu-vanstone implicit certificate scheme (ecqv)," *vol*, vol. 4, p. 32, 2013.

[45] ReMote, "A 6LoWPAN IoT device," http://zolertia.io/z1, 2017.

[46] Contiki, "Contiki os: An open source operating system for the internet of things," Online at http://www.contiki-os.org/, 2016.

[47] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," *RFC 7959*, 2014.

[48] D. F. Aranha and C. P. L. Gouvêa, "RELIC is an Efficient LIbrary for Cryptography," https://github.com/relic-toolkit/relic.

[49] C. Library, "Contiki apis for accessing realtime clock," 2017. [Online]. Available: http://www.eistec.se/docs/contiki/a02184.html

[50] Contik, "Contiki apis for measuring energy consumption," http://contiki.sourceforge.net/docs/2.6/a00452_source.html, 2017.