EAP-Swift: An Efficient Authentication and Key Generation Mechanism for Resource Constrained WSNs

Daniel Granlund, Christer Åhlund, and Patrik Holmlund

Division of Pervasive and Mobile Computing, Luleå University of Technology, 931 87 Skellefteå, Sweden

Correspondence should be addressed to Daniel Granlund; daniel.granlund@ltu.se

Received 21 November 2014; Revised 29 January 2015; Accepted 18 March 2015

Copyright © 2015 Daniel Granlund et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Technological advances within the area of wireless sensor technology allow WSNs to be used in a increasing number of measurement scenarios. As new application areas are emerging, such as infrastructure monitoring and smart cities, the need for sensor mobility handling requires efficient and secure authentication protocols. This paper presents EAP-Swift, a novel EAP based authentication protocol with a focus on lightweight processing and faster response. It supports end-to-end session encryption key generation and mutual authentication. By utilizing lightweight hashing algorithms, the challenge-response authentication mechanism uses only two round trips to the AAA server for the complete authentication procedure leading to the reduction of latency by 33% compared to the baseline protocols. Further, using extensive experimentation, we validate that the authentication time can be kept below 250 ms and the power consumption can be kept below 15 mJ. Furthermore, we show that a battery lifetime of more than four years can be achieved when running the system on a regular button cell battery. Finally, the protocol was verified in terms of security using the AVISPA tool.

1. Introduction

Wireless sensor networks (WSNs) are becoming more and more prevalent in our everyday lives. We are witnessing a trend where WSN application areas tend to expand from a traditional industrial environment to home and public service applications. Sensors can be used not only to monitor industrial processes but also to monitor and control appliances in the home as well as important public service functions such as seasonal road and bridge maintenance [1, 2]. However, in most cases, it is not feasible to have hard wired sensors in all locations due to increased cost and reduced flexibility. Therefore, WSNs are widely used for industrial and home applications. Wireless sensor nodes are usually battery powered and can support a battery lifetime of several years. A typical wireless sensor node has a communication range of approximately 100 metres and, with multihop capabilities, can span large areas.

An obvious disadvantage, however, is the added security features that are involved with sending data over a shared medium. In an unprotected WSN, a malicious node can easily pick up traffic between nodes, inject false data, and affect the system functionality [3]. Data encryption between wireless sensor nodes is typically supported by several wireless sensor platforms such as that in [4]. Encryption algorithms such as advanced encryption standard (AES) [5] or elliptic curve cryptography (ECC) [6] are used to perform robust encryption of the wireless data.

Some modern sensor platforms, such as that in [4], have a hardware implementation of cryptographic functions to speed up the encryption process by avoiding CPU intensive operations on the main CPU. The encryption procedure typically involves the use of an encryption key to make the crypto understandable to a receiving party that has the corresponding key.

In symmetric key cryptography [7], the key is the same at both the sender and the receiver sides, whereas, in asymmetric key cryptography [6], there is one key to encrypt and another key to decrypt a message. Symmetric key cryptography is generally considered to be less demanding on CPU and power consumption, so it is usually preferred in sensor implementations where resources can be scarce. A common key can be distributed in advance to all nodes in the WSN; this method is called preshared key (PSK).
and may be used in, for example, ZigBee [8] and Bluetooth networks [9]. Another alternative is to exchange individual keys dynamically between communicating parties when initializing the communication. This process, therefore, is more secure since it also protects against over hearing between nodes within the network. Exchanging keys dynamically also allows sensors to move between different networks and adds the ability to renegotiate keys at any time [8].

As part of the Sense Smart City project [10], a general wireless sensor network (WSN) platform is developed which can support a wide variety of sensor devices such as sewage water level sensors [1] and structural motion sensors [2]. A scalable backend server system, along with a flexible application programming interface (API), makes the system easy to extend and implements new functionality such as adding more sensors and applications. To tackle these challenges, we developed the Sense Smart City sensor authentication model as shown in Figure 1. This model is motivated by the EduRoam [11] architecture. It supports sensor mobility on a global scale by enabling the interconnection of trusted WSN gateway providers in different administrative domains. An administrative domain in this context consists of one or many WSNs that are owned and managed by a single administrative entity (e.g., a municipality). The administrative domains are interconnected in a tree-like structure with a preestablished trust agreement formed with the closest node (AAA server) above in the tree, thereby eliminating the need to form trust agreements between all nodes. The root node (root server) is typically managed by some entity that has high level privileges, for example, a government or a third party provider having control over the AAA system. Using this architecture, it is possible to form a larger network of trusted nodes that are able to route authentication messages between them. Sensors roaming between different administrative domains need to be preconfigured with a sensor ID along with its authentication credentials. By identifying the sensor with a network access identifier (NAI) in the form uniqueID@realm (e.g., sensor543@administrative-domain.example.net), the sensor gets a globally unique name. The realm part identifies the administrative domain and thereby enables the AAA server hierarchy to route the request to the appropriate destination. Using that information, the sensor will be able to authenticate and connect to any trusted WSN and establish a secure communication channel to the home administrative domain (the administrative to which the sensor belongs). In this paper, we do not address the problem of mobility detection and management in WSNs. Instead, we focus on developing an authentication protocol that supports sensor mobility.

Secure authentication for sensor mobility is a key aspect that needs consideration while sensors are moving between

![Figure 1: Sense smart city sensor authentication model using tree-like interconnection of AAA servers.](image-url)
WSNs belonging to different administrative domains (ADs) [12]. In a smart city environment, due to the heterogeneity, large coverage area, and wide range of applications, there will be a need to support sensors moving between different WSNs and ADs [13, 14]. For example, in a smart city environment, sensors might be placed on cars in order to measure air quality. Such highly mobile sensors might span a large area and are expected to attach to networks belonging to different ADs. To support secure sensor mobility, a dynamic authentication and key management scheme is required since it is not feasible to have preshared keys distributed to all networks to which the sensor might move. In order to build an efficient sensor mobility system, there is a need for an authentication protocol that allows sensors to change their point of attachment without causing excessive service disruption.

The key contribution of this paper is to propose, implement, and validate a novel, lightweight authentication solution, EAP-Swift, an authentication protocol that includes the creation of a symmetric encryption key. This results in reduced energy consumption and makes mobility of sensors less resource demanding. EAP-Swift provides two major functionalities:

(i) lightweight and power efficient mutual authentication, including session encryption key generation,
(ii) enabling sensor mobility in resource constrained sensor networks.

The proposed protocol, EAP-Swift, is unique in the sense that it is specifically designed to support sensor mobility and interdomain roaming by reducing authentication delay and carrying out key generation as part of the authentication phase. In particular, EAP-Swift is designed to operate end-to-end between the sensor node and the AAA server. To support AAA request proxying without additional signaling, EAP-Swift includes the sensor NAI in the first message sent to the AAA server. Session key generation for end-to-end encryption, while still providing a reduction of 33% in message exchange, is achieved as compared to baseline protocols [15, 16].

The reminder of the paper is organized as follows. Section 2 presents related work. Section 3 presents our contribution, EAP-Swift, and its verification. Section 4 presents the prototype implementation and results analysis. Finally, Section 5 presents the conclusion and future work.

2. Related Work

Current research in the area of sensor mobility and inter-WSN mobility support is generally centered around an IP-based WSN. Proposed mobility solutions for IP-based sensor networks typically implement functionality based on 6LoWPAN [17] and Proxy Mobile IPv6 [18] such as 6LoMSNS [19] and SPMIPv6 [20]. Mobile IPv6 enabled sensors are globally addressable on the Internet using IPv6 addressing; and mobility management mechanisms allow IP addresses to be maintained while roaming between networks. The aforementioned solutions, however, typically adds overhead in the form of IP headers and extensive signalling as well as adding requirements on protocol specific implementations in many parts of the network infrastructure. However, this approach is not optimal from a power efficiency and implementability point of view [21].

For the past few years, significant effort has been put into developing security protocols for WSNs [6, 7, 22, 23]. For example, [22] developed security mechanisms including elliptic curve cryptography for WSNs. In [23], a comprehensive, standards compliant framework is presented for securing IEEE 802.15.4 networks which targets IP-based sensors. However, the support for AAA infrastructure as mentioned in Figure 1 necessitates the use of a compliant authentication protocol transported by an AAA protocol such as RADIUS [24] or DIAMETER [25]. Protocols based on the extensible authentication protocol (EAP) [26] framework can also be designed to fit this purpose as it acts as a transport mechanism that encapsulates messages used during authentication.

There are several EAP based protocols proposed in the literature, such as EAP-MD5 [26], EAP-TLS [27], and EAP-TTLS [28], which are intended for authentication in IEEE 802.11 wireless LAN. EAP-SIM [29] and EAP-AKA [30] are examples of EAP based authentication protocols designed for cellular networks, used by mobile phones. There are also a number of lightweight EAP based protocols developed for sensor authentication over the past few years. For example, EAP-Sens [15] supports node authentication and session key generation based on the EAP-GPSK [31] protocol with hierarchical keys. The authentication mechanism assumes that both the sensor node and the AAA server have a secret preshared key (PSK). In addition to performing mutual authentication, EAP-Sens provides functionality to generate a master session key (MSK) after the authentication procedure which can be used by the WSN gateway to create keys for data encryption.

EAP-EHash [16] is an authentication and session key generation protocol based on hashing algorithms that support mutual authentication as well as a high degree of security. EAP-EHash uses one-way hashes to perform mutual authentication. The EAP-EHash authentication procedure is divided into three phases. The first phase is the negotiation phase which allows communicating parties to negotiate the usage of ciphersuite. Second phase carries out the actual authentication. Authentication of both parties is carried out by each party producing a random value, so called a nonce, which is communicated to the opposite party. The opposite party then forms a response by calculating the one-way hash of the PSK along with the received nonce. The response is then communicated and can be verified at the receiving end. In the third and final phase, a session key is generated between the node and the gateway.

3. EAP-Swift: A Lightweight Protocol for Sensor Authentication

In line with the EAP based protocols mentioned above, in this paper, we propose a sensor authentication protocol called EAP-Swift. In EAP-Swift, the session key is generated between the sensor node and the AAA server that the
sensor node belongs to, rather than between the sensor and the gateway (where the sensor node moves to). Therefore, no keying information is sent to any intermediate entity (e.g., gateway or AAA server) and enables encryption to be carried out end-to-end. For example, in Figure 1, a sensor node belonging to the administrative domain A (called the home domain) moves temporarily to administrative domain B (called the visited domain). As previously mentioned, EAP-Sens is an authentication protocol specifically designed to be run on resource constrained platforms such as sensors [15]. EAP-Sens is an extension to the EAP-GPSK protocol [31] that aims to minimize computation and communication overhead. The described optimized solution uses three round trips between the sensor and server (six messages) for a full authentication [30]. In EAP-EHash, a full authentication can be carried out with only two round trips, if the recommended ciphersuite negotiation phase is omitted. However, using the default ciphersuite will require computationally demanding cryptographic operations [16]. Compared to [15, 16], EAP-Swift does not utilize encryption of authentication messages. Instead, it considers session unique nonces that are included in calculated hashes to prevent replay attacks. This is a critical functionality in the case of sensor mobility between different administrative domains. Section 5 will show how EAP-Swift reduces authentication latency and power consumption by approximately 33% compared to [15, 16].

3.1. Protocol Design. In this section, we present EAP-Swift, a lightweight sensor authentication protocol based on hashing functions such as MD5 [32] and SHA1 [33] that are used for both authentication and end-to-end encryption key generation between sensor node and home AAA server (see Figure 1). The hashing functions take a variable size data block and produce a fixed size hash that is a product of a number of calculations on the input data. For MD5 and SHA1, the hash sizes are 128 and 160 bits, respectively. The produced hash has the following properties: (i) it is relatively easy to calculate; (ii) it will change if the input data is changed; and (iii) it is very hard to find a block of data that will produce the same hash. Traditional hash based authentication algorithms, or so called digest authentication algorithms like EAP-MD5 [26], are typically implemented in the following fashion: the requesting party sends a message that contains a nonce, which is a randomly generated number that is preferably changed for every authentication attempt. The requested party then creates a hash of its authentication credentials along with the received nonce and returns the message. If the requesting side has the same credentials, it will be able to produce the same hash and verify the authenticity of the requested party without the transmission of any authentication credentials. EAP-MD5 supports only single sided authentication since there is no mechanism that allows the sensor node to authenticate the AAA server.

In the proposed protocol, as shown in Figure 2, mutual authentication is provided by carrying out the challenge-response authentication procedure in both directions. In the first step (Step 1), an EAP-Start – ID Request handshake is performed between the sensor (or peer) and the WSN gateway (or authenticator). In this step, the sensor sends requests to connect to the gateway using an EAP-based protocol. If this is supported by the gateway, it will respond with an EAP ID Request message requesting the sensor identity. In the second step (Step 2), the sensor will then respond with an EAP ID Response message containing the EAP Session ID (ID) and the sensor network access identifier (NAI). The authenticator or the WSN gateway will then relay the message to the RADIUS enabled AAA server either directly or proxied via a number of servers to its destination. A preconfigured entry at the destination AAA (Home AAA or AAA_H) server includes the information about the sensor NAI and the preshared key. The AAA_H server will generate a 128 bit nonce $\eta$ and return it as a challenge to the sensor node via the WSN gateway. A 128 bit nonce is considered sufficient in WSN applications since the sensor battery life would never

**Figure 2: EAP-Swift authentication protocol.**
support the high number of authentications that have to be recorded to conduct a replay attack [15]. In the third and final authentication step (Step 3), the sensor node will then create a 128-bit nonce $\eta$ and send it along with a hash created from the ID, the preshared key (psk), $\eta n$, and $\eta s$. The NodeResponse can be written as follows:

$$\text{NodeResponse} = \text{Hash}(\text{ID} \| \text{psk} \| \eta n \| \eta s). \quad (1)$$

By adding $\eta n$ to the computed hash, the $\eta n$ value is signed and therefore it is not possible for an attacker to change during the transition between the sensor and the AAA-H. Also, it will not be possible for an attacker to directly query the node for the expected hash result for a given nonce. Upon reception, the server can verify that the node side and the $\eta n$ value are authentic. If the sensor node is authenticated, the AAA-H server generates an EAP Success message including a third 128-bit nonce $\eta k$ which is then signed with a hash (ServerResponse) consisting of ID, psk, $\eta n$, and $\eta k$. This way, it is possible for the node to authenticate the server side by verifying the hash. The server response can be written as follows:

$$\text{ServerResponse} = \text{Hash}(\text{ID} \| \text{psk} \| \eta n \| \eta k). \quad (2)$$

Also, the $\eta k$ value is signed and can be verified in the same way. To generate the session key, a key generation function (KDF) is used as shown in (3). The KDF is a simplified version of the generalized key derivation function, GKDF [31]. It truncates the hash function output to the desired key length (KeyLength) assuming that the length of the produced hash is always larger than or equal to KeyLength. For the purpose described in this paper, this is a safe assumption since, among the supported hashing functions, MD5 produces the shortest hash (128 bit) which is equal to the key length supported by hardware used in the experiments section. The nonce $\eta k$ is then used together with psk and $\eta n$ to create a session encryption key (SessionEncryptionKey) to use during the communication session. The exchange of $\eta k$ can be carried out as often as desired in order to regenerate the session key to increase security:

$$\text{KDF} = \text{Truncate}(\text{Hash}(\text{InputData}), \text{KeyLength}), \quad (3)$$

$$\text{SessionEncryptionKey} = \text{KDF}(\eta k \| \text{psk} \| \eta n). \quad (4)$$

If the sensor authentication fails, an EAP Fail message is sent to the node which will inform the node about the failed authentication and return it to an initial state.

3.2. Security Analysis. Similar to [16, 34, 35], we validate our protocol using the automated validation of Internet security protocols and applications (AVISPA) tool [36]. AVISPA is an analysis and validation tool for security protocols and applications. EAP-Swift was validated by implementing it in the high level protocol specification language (HLPSL) which is a high-level scripting language specifically designed for use with AVISPA as shown in Figure 3(a). EAP-Swift was then evaluated against a certain number of specified goals using different methods, or so called back-ends. The back-ends used in this evaluation were OFMC [37], CL-ATSE [38], SATMC [39], and TA4SP [40]. We defined the following goals for the successful validation of EAP-Swift:

1. mutual authentication of the sensor and the AAA server,
2. maintained secrecy of psk as well as the session key.

Figure 3(a) shows the protocol specific part of EAP-Swift in HLPSL for the AVISPA tool. Figure 3(b) shows the output from the AVISPA tool. EAP-Swift was implemented in HLPSL with two roles, server and sensor. Each role was implemented with a set of states and events generated by the peer triggering a state transition. The first requirement to verify was that the protocol supported mutual authentication. In the HLPSL code shown in Figure 3(a), the sensor node authenticity is verified on line 32 where the server verifies the received NodeResponse message hash (produced on line 12). Similarly, the server side authenticity is verified by the sensor node on line 15 where the received ServerResponse message hash (produced on line 33) is verified. The witness and request primitives used by both peers ensure that the nonces generated by remote peers are authenticated which protects against replay and man-in-the-middle attacks [16]. Finally, the secret primitives are used to verify that the secrecy of psk as well as the generated session key Sk is maintained. The output shows that the goals (marked with solid boxes) were met for all tested back-ends and that no attacks were found which would compromise the security. Results (marked with dashed boxes) verify that the proposed protocol fulfilled all desired security goals according to the aforementioned back-end methods.

4. Results Analysis

The previous section proposed EAP-Swift, a light-weight sensor authentication protocol designed to support sensor mobility. The proposed protocol was also verified using the AVISPA tool. This section further validates EAP-Swift using a prototype implementation and extensive experimentation. For experimentation, we considered the Sense Smart City platform [10] as mentioned in Section 2. The Sense Smart City platform supports sensor authentication using the RADIUS protocol while sensors are moving between different administrative domains (see Figure 1). The RADIUS protocol is a well-known AAA protocol that was originally designed for dial-up connections. It inherently supports AAA request proxying and tree-like interconnection of servers which makes it suitable for AAA support in mobility applications [11]. In our testbed, we considered FreeRadius [41] as a highly configurable and flexible RADIUS server platform. The FreeRadius server supports a comprehensive range of EAP-based authentication methods and modular construction. Further, it allows for the integration of new protocols in the form of customizable plugins. Therefore, in our testbed, we implemented EAP-Swift as a FreeRadius plugin. Furthermore, results from a scalability study of the FreeRadius server [42] showed that the server is suitable for building highly scalable systems.
For the sensor-side implementation, we considered the Mulle sensor platform (Mulle v6.2) developed by Eistec [4]. The Mulle platform (shown in Figure 4) is based on a 16-bit Renesas microcontroller and is equipped with an 868 MHz, IEEE 802.15.4 compatible radio and a 3-axis accelerometer. The Mulle sensor runs the TinyOS operating system [43] designed specifically to run on resource constrained devices. Further, TinyOS provides powerful APIs for creating sensor applications and is based on NesC, which is an extension to C programming language that supports event-driven software design. The software running on top of TinyOS is built using several components that are linked or wired together via a well-defined interface. This kind of software architecture provides a truly modular and scalable software that is easy to extend and shortens application development time.

4.1. Analysis of Hashing Algorithms. As a first step in implementing and evaluating the EAP-Swift protocol, two common hashing algorithms were implemented in TinyOS, namely, MD5 [32] and SHA1 [33]. Both algorithms were ported to the Mulle platform from open source implementations, MD5 [44] and SHA1 [45]. To support a direct comparison, the EAP-EHash [16] cryptographic mechanisms were implemented based on the aforementioned SHA1 source code and the open source implementation of the 3DES encryption algorithm [46]. Execution time and energy consumption

---

(a) HLPSL:
(1) role Swift_Sensor (A, B : agent, Psk : symmetric_key,
   H : hash_func, Snd, Rcv : channel(dy))
(2) played_by A
(3) def=
(4) local State : nat,
(5) Nn, Ns, Nk, Sk : text
(6) const sec_psk1, sec_sk1 : protocol_id
(7) init State := 0
(8) transition
(9) 1. State = 0 \land Rcv(start) =\rightarrow
(10) State' := 1 \land Snd(A)
(11) 2. State = 1 \land Rcv(Nn') =\rightarrow
(12) State' := 2 \land Nn' := new() \land Snd(Nn'.H(Psk.Nn'.Nn'.A))
(13) \land witness(A, B, ns, Nn')
(14) \land secret(Psk, sec_psk1, [A, B])
(15) 3. State = 2 \land Rcv(Nk'.H(Psk.Nn'.Nk')) =\rightarrow
(16) State' := 3 \land request(A, B, ns, Ns)
(17) \land sk' := H(Nn'.Psk.Nk')
(18) \land secret(sk', sec_sk1, [A, B])
(19) end role
(20) role Swift_Server (B : agent, Psk : symmetric_key, H : hash_func,
(21) Snd, Rcv : channel(dy))
(22) played_by B
(23) def=
(24) local State : nat,
(25) Nn, Ns, Nk, Sk : text
(26) const sec_psk2, sec_sk2 : protocol_id
(27) init State := 0
(28) transition
(29) 1. State = 0 \land Rcv(A') =\rightarrow
(30) State' := 1 \land Ns' := new() \land Snd(Ns')
(31) \land witness(B, A, ns, Nn')
(32) 2. State = 1 \land Rcv(Nn'.H(Psk.Nn'.Nn'.A)) =\rightarrow
(33) State' := 2 \land Nk' := new() \land Snd(Nk'.H(Psk.Nn'.Nk'))
(34) \land request(B, A, nn, Nn')
(35) \land secret(Psk, sec_psk2, [A, B])
(36) \land sk' := H(Nn'.Psk.Nk')
(37) \land secret(sk', sec_sk2, [A, B])
(38) end role

(b) AVISPA validation tool output.

Figure 3: (a) HLPSL implementation of EAP-Swift in AVISPA tool; (b) AVISPA validation tool output.
were evaluated for both implementations using a PicoScope MSO 2205 mixed signal digital oscilloscope [47] sampling at 49.08 kS/s. The Mulle sensor was connected through a 1 \( \Omega \) shunt resistor, and the voltage drop over the resistor was measured to calculate the current. Digital outputs on the Mulle were connected to the oscilloscope in order to signal the states of the program execution. As the first experiment, we compared the hashing operations required on the sensor side for a full EAP-Swift authentication to the hashing and cryptographic operations required by EAP-EHash [16] using the default ciphersuite as described in Section 2.

EAP-Swift uses a total of three hashing calculations for a complete authentication and key generation procedure (one for generating the response to the server, one for verifying the server response, and one for producing the session key). EAP-EHash, in omitting the ciphersuite negotiation phase, requires a minimum of five SHA1 hashing operations: one 3DES encryption and one 3DES decryption operation [16]. Figure 5(a) depicts the average measured execution times for the three implementations and Figure 5(b) shows the energy consumed during the operations. The results were based on 80 independent measurements carried out for each protocol version using randomly generated data as input. During experimentation, we observed that the CPU usage caused an average current consumption of 14 mA with a standard deviation value of <0.01 ms using the disabled radio chip. From this experiment, we conclude that the CPU time and the energy consumption can be significantly reduced by decreasing the number of cryptographic operations required to complete an authentication procedure. Using the MD5 based version of EAP-Swift would be rendered in a reduced CPU time from 378 ms to 6.5 ms and an 81% reduction in energy consumption compared to using EAP-EHash with default ciphersuite on the Mulle platform.

4.2. EAP-Swift Protocol Evaluation. In this experiment, we evaluated the EAP-Swift protocol in terms of power consumption and authentication delay. In particular, we measured the total time taken to authenticate and generate the session encryption key (Steps 1 to 3 and session encryption key generation, as shown in Figure 2). Further, time and power consumption measurements for evaluating the EAP-Swift protocol were carried out using the same method as explained in the previous experiment (see Section 4.1).

For the evaluation of the proposed protocol, a prototype system was developed (see Figure 6) with a Mulle sensor node which is connected via IEEE 802.15.4 radio to a gateway node composed of a second Mulle running base station software that acts as a transparent bridge to the USB interface on a PC. The PC was configured with Fedora 14 Linux operating system, with software developed in Python 2.7 to perform the gateway functionality including EAP initiation and RADIUS server relay. The relay function in the gateway identifies incoming EAP-response messages from the WSN, encapsulates them in a RADIUS message, and forwards them to the local AAA server. Similarly, it receives RADIUS-messages from the AAA server, extracts the EAP message contained, and sends it over the WSN to the intended destination. On the sensor side (Mulle sensor), the implementation was carried out in both NesC and C.

The evaluation software can assume three states as shown in Figure 7. GWDISCOVERY is the initial startup state of the sensor node. In this state, the node listens for periodic beacon signals sent from gateways. When a beacon signal is picked up from a gateway and the node wants to connect, it will initiate the EAP conversation by issuing EAP-Start as a unicast message to the gateway. If the EAP-Start is responded to with an ID Request, the node will enter the EAPAUTH state. If the node authentication is completed successfully, the node will enter the CONNECTED state and start its normal operation. If the communication is dropped during either the EAPAUTH or the CONNECTED state a timeout function is implemented. After a 5-second timeout, the node will be returned to the GWDISCOVERY state and try to reconnect.

On the AAA server side, EAP-Swift was implemented as an additional EAP method plugin. A mutual trust relationship was established between the gateway PC and the AAA_H server by preconfigured fixed IP addresses and a shared secret. Furthermore, the sensor ID and credentials were entered in the AAA user database as a NAI and a preshared key which also are hard coded into the sensor device. The gateway software uses an application called radclient [41] which is provided as a utility along with the FreeRadius server software package. Radclient is responsible for encapsulating EAP packets and communicating with the AAA server. It is also responsible for verifying the incoming RADIUS packets using the Message Authenticator value in the RADIUS message.

A total of 30 measurements were carried out for each of the EAP-Swift implementations based on MD5 and SHA1 hashing algorithms, respectively. Mean and standard deviation were calculated for the measurements. Time was measured from the creation of the EAP ID Response message on the sensor side until the authentication is complete, and the session encryption key is generated. In total, this involves three hash calculations at the sensor node and two round trips to the AAA_H server. Figure 8 shows a comparison of mean authentication times for both implementations.

The communication link between the gateway PC and the AAA_H server had a round-trip time of 2.2 ms in average...
Figure 5: (a) Execution times for cryptographic operations; (b) measured energy consumption during cryptographic operations.

Figure 6: Experimental setup for sensor power consumption measurement.

Figure 7: Sensor node software connection states.

Figure 8: Measured total authentication time.

Figure 9 shows a sample of the measured Mulle current consumption during a typical authentication procedure. Since the radio was active during the authentication process, the average idle current was 11 mA. One can see an initial spike at approximately 7 ms which corresponds to the transmission of the EAP ID Response message. Since it is a unicast message, there is a time period immediately after...
transmission where the radio is awaiting an ACK from the receiver [48]. At approximately 20 ms, an ACK for the EAP ID Response message was received by the sensor. The short spike at approximately 78 ms corresponds to the reception of the AUTH Request; however, it is not possible to conclusively verify this since this occurs on the driver layer in TinyOS. At 85 ms, an event was generated, indicating that a packet has been successfully received by the radio. Immediately, the first hash calculation started as part of building the AUTH Response message. At 108 ms, we can note the transmission of the aforementioned message followed by a wait period for the returning ACK. The EAP Success/Failure message is received at 185 ms and the event that indicates successful reception is generated at 195 ms. This is immediately followed by the hash calculations which verifies the authenticity of the AAA server. Upon successful authentication, the corresponding session encryption key is generated at approximately 200 ms.

The total energy consumed during the aforementioned procedure was approximated using the trapezoidal rule stated in (5) where \( N \) is the number of discrete measurement points. It can be written as follows:

\[
\int_{a}^{b} f(x) \approx \frac{1}{2} \sum_{k=1}^{N} (x_{k+1} - x_{k}) (f(x_{k+1}) + f(x_{k})).
\]

In our case, in (5), energy consumption during the authentication procedure starting at time = \( a \) and ending at time = \( b \) is approximated using \( N \) samples during that time period where \( f(x_{k}) \) corresponds to the measured power consumption at time \( x_{k} \). The results were calculated for a dataset of 30 measurements for both MD5 and SHA1 versions of the EAP-Swift protocol. We observed that the energy consumption for the MD5-based version was on average 12.1 mJ with a standard deviation of 0.5 mJ. For the SHA1 based version, the average energy consumption was calculated to be 12.9 mJ with a standard deviation of 0.5 mJ. From these results, we conclude that using a typical CR2032 button cell lithium battery with a supply voltage of 3 V and a capacity of 225 mAh contains 2430 J. Such a battery could, in theory, be used to power over 200,000 authentication procedures. This assumes that only authentication is carried out, but it gives an indication on how the authentication mechanism will affect the overall energy usage.

Consider a more realistic case where a sensor is used to authenticate, carry out a simple measurement, and transmit it to the gateway once per hour; we can estimate the battery life of the sensor node. Assume that the measurement involves reading an external temperature sensor and that task will fully utilize the CPU for 20 ms. Furthermore, we need to authenticate the sensor node, transmit the data, and wait for acknowledgment before the sensor node can return to a low-power sleep state.

For the case above, using the measurement results shown in Figure 9, we can estimate that the sensor node would require 12.1 mJ for authentication and 1.5 mJ for data acquisition and transmission. Further, it consumes 4 \( \mu \)A in low-power sleep state [4]. This would allow for a theoretical total energy consumption of 56.8 mJ per cycle and a battery life of approximately 4.9 years using the aforementioned CR2032 lithium battery. As can be observed in Figure 8, approx. 96% of the authentication time using the MD5 based implementation of EAP-Swift is due to the communication latency (approx. 92% for SHA1 version). Since we did not have access to the implementations of protocols [15, 16, 31], we were not able make a direct comparison of the full authentication procedure. Rather we concluded that, in order to achieve the same functionality for protocols [15, 31], one additional round trip would be required to support mutual authentication and end-to-end key generation. Compared to the authentication protocols [15, 31] that may require at least three round trips to complete an authentication procedure and generate an end-to-end key, EAP-Swift reduces the communication latency by 33%, thereby leading to 9% battery gain using the Mulle platform for the application mentioned above. For the protocol [16], we concluded that by omitting the recommended cipher-suite negotiation, two round trips are enough to complete the authentication procedure. However, since the authentication mechanism relies on more computationally demanding operations as proven by experiment 1, EAP-Swift offers significant advantages in a resource constrained application.

Considering a case where an application requires a high data update frequency (e.g., car monitoring applications), the number of round-trips for sensor authentication and key generation can have a significant impact on the maximum number of nodes in a single WSN. Equation (6) calculates the theoretical maximum number of sensor nodes that can be supported in a WSN:

\[
N_{\text{max}} = \frac{\text{ABW} \times f_{\text{pkt}}}{L_{\text{frame}} \times n_{\text{pkt}} \times 8},
\]

where \( N_{\text{max}} \) represents the maximum number of sensors in WSN. ABW represents the maximum bit rate (kbps) supported by the radio technology, \( L_{\text{frame}} \) is the frame length supported by the sensor node, \( n_{\text{pkt}} \) is to total number of packets sent in the wireless network during one sensor cycle, and \( f_{\text{pkt}} \) is the frequency at which the measurement cycle is carried out. Since TinyOS uses fixed message sizes, the bandwidth consumption is directly related to the packet frequency and the frame length, \( L_{\text{frame}} \) used [49]. Using the same sensor behavior as in the previous example where the sensor authenticates itself and then transmits the data,
the theoretical maximum number of sensor nodes can be calculated with (6). Figure 10 shows how the maximum number of Mule sensors with 1 sec update interval ($f_{pk} = 1$) in a WSN depends on the available bandwidth. The Mule IEEE 802.15.4 radio has a theoretical maximum throughput of 250 kbps and a maximum frame size of 127 bytes. It can be appreciated that EAP-Swift that only uses 2 RTTs instead of 3 RTTs in order to complete the authentication procedure and key generation would, in this case, is rendered in a 29% increase in the maximum number of sensors in the network.

4.3. Discussion. From the results presented in this section, it can be concluded that an authentication protocol based on hashing algorithms can be made light weight in terms of CPU resource and power consumption. We can also see that the largest impact on power consumption is caused by the sensor node staying awake during communication with the AAA server. This delay is reduced in EAP-Swift by eliminating one round trip to the AAA server as compared to baseline protocols. When possible, care should be taken to keep the round-trip time to the server as low as possible in locations where the sensor is commonly attached. Furthermore, we can see that operations and scheduling carried out by the sensor operating system may have an impact on the power consumption. For power constrained applications (e.g., battery powered sensors) where authentication is carried out frequently, the power consumption can be further reduced by optimizing how this is handled by the operating system. The reduced amount of messages sent between entities during sensor node authentication will decrease flexibility when it, for instance, comes to the ability to negotiate certain configuration parameters such as which hashing algorithm to use. It is, for instance, always assumed that a session key is desired. Therefore, the $pk$ nonce is always generated and included in the Access-Accept message. By implementation and extensive evaluation on real hardware, we show that our proposed protocol is implementable and operates on industry standard platforms (Tiny OS and FreeRadius).

5. Conclusions and Future Work

This paper presents EAP-Swift, a novel, EAP based authentication protocol, for use in sensor mobility applications. Using an EAP-based authentication protocol allows for compatibility with preexisting AAA protocols. In particular, an AAA infrastructure based on the RADIUS protocol allows for interconnection of different domains on a global scale to support global roaming between domains. EAP-Swift protocol supports mutual authentication of sensor node and back-end AAA server. It is based on lightweight hashing algorithms and relies on only two round trips of communication between the sensor node and the AAA server. This allows for fast performance and low energy consumption. During the final step of authentication, a session key is generated between the sensor and AAA server which enables encryption of data traffic end-to-end. This is an improvement by 33% or 1 round trip compared to the baseline protocols in order to achieve the same functionality while reducing power consumption. Scalability is an important aspect when creating large WSN infrastructures. The EduRoam collaboration [11] is a good example that proves that the AAA infrastructure can scale globally. Since EAP-Swift supports EduRoam-like infrastructure, it can be used to form a scalable system.

Future work will involve evaluating EAP-Swift in a real-world case where handover mechanisms will be implemented in order to study how the architecture performs in a true sensor mobility scenario.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

The work presented in this paper is part of the SenseSmartCity [14] project supported by EU Structural Funds, Skellefteå Municipality and the Regional Council of Västerbotten.

References

[1] D. Granlund and R. Brännström, “Smart city: the smart sewerage,” in Proceedings of the 37th IEEE Conference on Local Computer Networks Workshops (LCN Workshops ’12), pp. 856–859, IEEE, Clearwater, Fla, USA, October 2012.

[2] R. Brännström and D. Granlund, “Sensor monitoring of bridge movement: a system architecture,” in Proceedings of the 36th IEEE Conference on Local Computer Networks (LCN): 5th IEEE Workshop on User Mobility and Vehicular Networks. IEEE Communications Society (LCN ’11), pp. 797–800, Bonn, Germany, October 2011.

[3] H. K. Kalinta and A. Kar, “Wireless sensor network security analysis,” International Journal of Next-Generation Networks, vol. 1, no. 1, 2009.

[4] Eistec AB, http://www.eistec.se.
[5] J. Daemen and V. Rijmen, The Design of Rijndael: AES—The Advanced Encryption Standard, Springer, 2002.

[6] I. Blake, G. Seroussi, and N. Smart, Elliptic Curves in Cryptography, vol. 265 of London Mathematical Society Lecture Note Series, Cambridge University Press, Cambridge, UK, 1999.

[7] N. Daswani, C. Kern, and A. Kesavan, Symmetric Key Cryptography, Foundations of Security, 2007.

[8] ZigBee Alliance, ZigBee Specifications, Version 1.0, ZigBee Alliance, 2005.

[9] Bluetooth SIG, Bluetooth Baseband Specification Version 1.0B, http://www.bluetooth.com.

[10] Sense Smart City project, http://www.sensesmartcity.org.

[11] EduRoam, http://www.eduroam.org.

[12] M. Zorzi, A. Guluak, S. Lange, and A. Bassi, “From today’s INTRAnet of things to a future INTERnet of things: a wireless and mobility-related view,” IEEE Wireless Communications, vol. 17, no. 6, pp. 44–51, 2010.

[13] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, “Internet of Things for smart cities,” IEEE Internet of Things Journal, vol. 1, no. 1, pp. 32–42, 2014.

[14] P. Bellavista, G. Cardone, A. Corradi, and L. Foschini, “Convergence of MANET and WSN in IoT urban scenarios,” IEEE Sensors Journal, vol. 13, no. 10, pp. 3558–3567, 2013.

[15] M. A. Alim and B. Sarikaya, “EAP-Sens: a security architecture for wireless sensor networks,” in Proceedings of the 4th International ICST Conference on Wireless Internet (WICON ’08), Maui, Hawaii, USA, November 2008.

[16] O. Cheikhrouhou, M. Laurent, A. B. Abdallah, and M. B. Jemaa, “An EAP-EHash authentication method adapted to resource constrained terminals,” Annals of Telecommunications, vol. 65, no. 5-6, pp. 271–284, 2010.

[17] J. Hui, Ed., Compression Format for IPv6 Datagrams over IEEE 802.15.4—Based Networks, RFC6282, IETF, 2011.

[18] K. Leung, V. Devarapalli, K. Chowdhury, and B. Patil, Proxy Mobile IPv6, IETF, RFC 5213, edited by S. Gundavelli, 2008.

[19] J. Kim, R. Haw, E. J. Cho, C. S. Hong, and S. Lee, “A 6LoWPAN sensor node mobility scheme based on proxy mobile IPv6,” IEEE Transactions on Mobile Computing, vol. 11, no. 12, pp. 2060–2072, 2012.

[20] M. M. Islam and E.-N. Huh, “Sensor proxy mobile IPv6 (SPMIPv6)-a novel scheme for mobility supported IP-WSNs,” Sensors, vol. 11, no. 2, pp. 1865–1887, 2011.

[21] R. Silva, J. S. Silva, and F. Boavida, “A proposal for proxy-based mobility in WSNs,” Computer Communications, vol. 35, no. 10, pp. 1200–1216, 2012.

[22] H.-L. Yeh, T.-H. Chen, P.-C. Liu, T.-H. Kim, and H.-W. Wei, “A secured authentication protocol for wireless sensor networks using Elliptic Curves Cryptography,” Sensors, vol. 11, no. 5, pp. 4767–4779, 2011.

[23] S. Sciancalepore, G. Piro, E. Vogli, G. Boggia, and L. A. Grieco, “On securing IEEE 802.15.4 networks through a standard compliant framework,” in Proceedings of the Euro Med Telco Conference (EMTC ’14), pp. 1–6, Naples, Italy, November 2014.

[24] C. Rigney, Ed., Remote Authentication Dial in User Service (RADIUS), IETF, RFC2865, 2000.

[25] P. Calhoun, Ed., Diameter Base Protocol, RFC3588, IETF, 2003.

[26] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowitz, Extensible Authentication Protocol (EAP), RFC3748, IETF, 2004.

[27] D. Simon, B. Aboba, and R. Hurst, The EAP-TLS Authentication Protocol, IETF, RFC5216, 2008.

[28] P. Funk and S. Blake-Wilson, Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0), RFC 5281, IETF, 2008.

[29] H. Haverinen and J. Salowey, Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM), RFC 4186, IETF, 2006.

[30] J. Arkko, V. Lehtovirta, and P. Eronen, Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA), RFC 5448, IETF, 2009.

[31] T. Clancy and H. Tschofenig, EAP Generalized Pre-Shared Key, RFC 5433, IETF, 2009.

[32] R. Rivest, The MD5 Message-Digest Algorithm, RFC1321, IETF, 1992.

[33] D. Eastlake and P. Jones, US Secure Hashing Algorithm 1 (SHA1), RFC 3174, IETF, 2001.

[34] A. K. Rai, V. Kumar, and S. Mishra, “An efficient password authenticated key exchange protocol for WLAN and WiMAX,” in Proceedings of the International Conference & Workshop on Emerging Trends in Technology, pp. 881–885, ACM, February 2011.

[35] S. U. Khan, C. Pastrone, L. Lavagno, and M. A. Spirito, “A secure online key establishment scheme for mobile heterogeneous sensor networks,” International Journal of Distributed Sensor Networks, vol. 2014, Article ID 925479, 12 pages, 2014.

[36] Automated Validation of Internet Security Protocols and Applications, AVISPA, http://www.avispa-project.org/.

[37] D. Basin, S. Mödersheim, and L. Viganò, “OFMC: a symbolic model checker for security protocols,” International Journal of Information Security, vol. 4, no. 3, pp. 181–208, 2005.

[38] M. Turunni, “The Cl-Ate protocol analyser,” in Proceedings of the 17th International Conference on Term Rewriting and Applications, Seattle, Wash, USA, August 2006.

[39] A. Armando and L. Compagna, “SATMC: a SAT-based model checker for security protocols,” in Proceedings of the 9th International Conference on Logics in Artificial Intelligence (JELIA ’04), pp. 730–733, Lisbon, Portugal, September 2004.

[40] Y. Boichut, P.-C. Héam, and O. Kouchnerenko, “Tree automata for detecting attacks on protocols with algebraic cryptographic primitives,” in Proceedings of the 9th International Workshop on Verification of Infinite-State Systems, Lisbon, Portugal, September 2007.

[41] FreeRadius, http://www.freeradius.org.

[42] D. Granlund and C. Ahlund, “A protocol implementation of AAA in heterogeneous networks using global roaming support,” in Proceedings of the 10th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom ’11), pp. 488–493, IEEE, Changsha, China, November 2011.

[43] P. Reis, A. Maddon, J. Polastre et al., “TinyOS: an operating system for sensor networks,” in Environment and Energy, pp. 115–148, Springer, Berlin, Germany, 2005.

[44] L. P. Deutsch, Independent Implementation of MD5 (RFC3132), Aladdin Enterprises, 2002, http://fedoraproject.org/w/uploads/2/22/Deutsch-md5.c.

[45] P. E. Jones, Secure Hashing Algorithm (SHA1), A C and C++ implementation, http://wwwpacketizer.com/security/sha1/.

[46] DES and Triple-DES encryption/decryption Algorithm, http://www.mrtc.mdh.se/projects/wcet/wcet_bench/des/des.c.

[47] Pico Technology, http://www.picotech.com.

[48] Atmel Corp, http://www.atmel.com/.
[49] J. Chen, L. Zhou, B. Zheng, and J. Cui, “Application-dependent frame design for the internet of things,” in Proceedings of the IEEE International Conference on Communications (ICC '13), pp. 3694–3698, June 2013.
Submit your manuscripts at
http://www.hindawi.com