Lightweight Group-Key Establishment Protocol for IoT Devices: Implementation and Performance Analyses

Nico Ferrari, Teklay Gebremichael, Ulf Jennehag, and Mikael Gidlund
Department of Information Systems and Technology
Mid Sweden University, Sweden
Email: nife1600@student.miun.se, {teklay.gebremichael , ulf.jennehag , mikael.gidlund}@miun.se

***Abstract*— In the context of Internet of Things (IoT), group communication is an efficient and fast way of broadcasting group messages. The message needs to be sent securely to maintain confidentiality of data and privacy of users. Establishing cryptographically secure communication links between a group of transceivers requires the pre-agreement upon some key, unknown to an external attacker. Complex and resource-intensive security schemes are infeasible for devices with limited computational capabilities. In this paper, we implement a lightweight and computationally secure group key establishment scheme suitable for resource constrained IoT networks. The proposed scheme is based on elliptic curve cryptography and cryptographic one-way accumulators. We analyze its feasibility by implementing it in the Contiki operating system and simulating it with the Cooja simulator. The simulation results demonstrate the feasibility of the scheme and its computational and communication costs are also comparable with other existing approaches, with an energy consumption of only 109 mJ per node for group key establishment.**

***Index Terms*— IoT security, Group communication security, ECC, Cryptographic key establishment, Lightweight cryptography, Contiki, one-way accumulators**

I. Introduction

The Internet of Things (IoT) denotes the interconnection of highly heterogeneous networked entities (nodes) [1]. There will be about 24 billion of connected devices by 2020 [2], thereby creating an extremely powerful element of next generation networking technologies. IoT has become immersive and pervasive in everyday life by empowering easy access and interaction with a wide variety of devices, presenting a growth of group-based applications [3]. Group communication occurs when a message is broadcasted to a set of nodes, possibly in a multicast or broadcast group. Group communication takes place in many different sectors, from smart lighting, where a central gateway controls lighting level and related functions of the connected lighting devices, to e-health where sensors implanted in a patient's body can send data to a server and/or readers used by doctors and nurses.

However, the nodes that will constitute these networks are often weak, both in terms of computational power and security measures, as they are very constrained in hardware and CPU clock. Thus security services that reside in the higher layers of the OSI model, such as traditional cryptographic protocols, that require key distributions or digital signature generation, cause significant overhead regarding computation and communication [4]. Finding a balance between security solutions and the associated resources utilization overhead is needed to mitigate security threats. The goal of securing a node-group communication is to guarantee authenticity, confidentiality (providing forward and backward secrecy when nodes leave or join the network, respectively) and integrity of the exchanged messages [5]. Public key cryptography (PKC) could be good choice for ensuring a satisfactory level of security for data transmissions within the network. This is because one does not need to transmit the private key in the channel, in addition to providing digital signatures that can not be repudiated. However, the constrained energy, computation, and memory budgets of sensors make the implementation of PKC challenging.

In this paper we implement a secure scheme proposed in [6] showing that the protocol is suitable for constrained devices making comparisons with other similar schemes. To reach our goals, we want to minimize the number of messages exchanged between the gateway and nodes and the nodes' computation overload in order to minimize the energy consumption, without compromising the level of security. The technique described in this work is based on scalar elliptic curve point multiplication used as a one-way accumulator. This technique has been presented in [6] and permits to initiate a shared secret between the legitimate nodes participating in the group. We show that a node can easily create a secret to prove its membership to the group, generating a security layer against attackers which want to fake the group membership. In fact, the security of this secret, lies in the difficulty of solving the Elliptic Curve Discrete Log Problem (ECDLP) to generate a fake membership. We describe a re-keying operation after every node's join or leave so that a former group member has no access to current communications and a new member has no access to previous communications[7].

Our contributions are summarized below. We implement on the Contiki operating system our scheme using elliptic curve multiplication as a cryptographic one-way accumulator function which permits the secure establishment of a group key. We show the feasibility and efficiency of our protocol simulating a network of IoT devices with the Cooja simulator, analyzing the results in terms of energy consumption, memory utilization and security features.

This work is supported by grant ERUF ID20201010 SMART - Smarta system och tjänster för ett effektivt och innovativt samhälle.

The rest of the paper is organized as follows: In Sect. II, we review some related work on group key distribution and management. In Sect. III we describe the setup of the devices in the network and some concepts used during the construction of our protocol. In Sect. IV we present our scheme, showing the different phases covered by the protocol. In Sect. V we describe the setup for the performed tests. In Sect. VI our scheme has been analyzed in terms of security and performance to show its impact on the network's performances and compared with others existing protocols with similar approach. Finally, Sect. VII provides conclusions.

II. Related Work

In multicast group communications, the energy consumption, bandwidth and processing overhead at the nodes are minimized compared to a point-to-point communication system [8]. The multicast communication protocol has to generate and distribute a secret group key that can be used to encrypt data sent from one source to all destinations that are member of the same group. Multicast groups are often very dynamic due to the join and leave of the members, and for this reason the protocol has to handle such group membership changes by re-generating and re-distributing new group keys in a secure and efficient matter.

This paper is based on a previous work presented in [6]. There are well studied group key establishment protocols in use today. As described in [9]and [10], it is possible to extend the Diffie-Hellman protocol to a group of nodes and generate a one-time session key. However, it is not ideal for power and resources constrained devices due to the intensive use of computational power. Moreover,it has been demonstrated that [10] do not meet some security requirements [11].

A conference-key distribution system is proposed in [12]. However it is demonstrated that the system is insecure because the information exchanged by the users makes it possible for a passive eavesdropper to compute the key. The approach used for [12] needs a high number of messages exchanged and executes an high number of integer exponentiation operations.

Some other protocols like μTESLA [13] are some of the earliest proposed protocols in the category symmetric key based protocols and they are based on hash function to reduce the energy consumption but they do not check for data integrity. Other symmetric key schemes based on key ring like [11] are not scalable, and therefore, these are unsuitable for dynamic environments such in our scenario.

In [14] the authors propose two lightweight group-key establishment protocols using an approach similar to ours. The first protocol allows only the legitimate members of the multicast group as eligible to continue the rest of the process of key derivation. This one is more appropriate for distributed IoT applications, which require nodes to contribute hightly to the key computation and need greater randomness. The second one allows to establish a shared secret key among the multicast group. This one is more suitable for centralized IoT applications, where a central entity performs the majority of the cryptographic functions.

III. Preliminaries and Setup

A. Elliptic Curves and One-Way Accumulators

Elliptic Curve Cryptography (ECC) is a preferred choice among various PKC options due to its fast computation, small key size, and compact signatures[15]. Experiments proved that Elliptic Curve Cryptography (ECC) is more suitable for resource constraint motes compared with RSA [16][17]. For example, to provide equivalent security to 1024-bit RSA, an ECC scheme only needs 160 bits on various parameters, such as 160-bit finite field field operations and 160-bit key size[15]. EC parameters are denoted by *a, b, q, p and* $\mathbb{P}$ and they are embedded in all the entities that participate in the communication scenario. The parameter q is the prime which indicates the finite field $\mathbb{F}_q$. The variables a and b are the coefficients of the elliptic curve. $\mathbb{P}$ is the base point generator with order p which is a prime number.

We build a secure group membership operation by using one-way accumulators [18] using point multiplication on elliptic curve as a one-way asymmetric accumulator. Here the basic function f is defined as point multiplication as described in [6]:

$$f(s, \mathbb{P}) = s \times \mathbb{P} = \mathbb{Q}$$

where $\mathbb{P}$ and $\mathbb{Q}$ are points on the curve and s is a large scalar value.

B. Network Setup

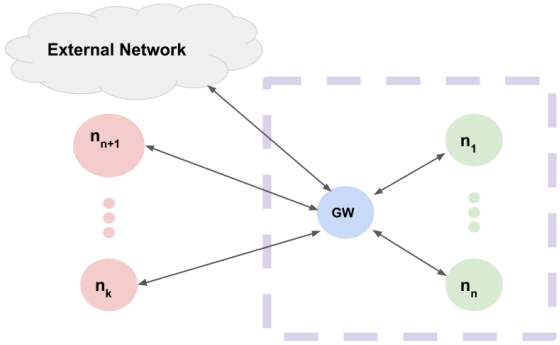

Fig. 1. Network Model. The network consists of a gateway and a set of nodes, supported by a communication infrastructure. All or a part of the nodes may be members of a group as shown in the figure (m nodes are in the group).

It is defined as multicast group a particular group of nodes, which are entitled to receive the common set of information. In this paper, we consider a network composed by *k* nodes connected to a gateway *G* as shown in Fig. 1. Formally, the network can be modeled as a graph $Gr = \langle G, N, E \rangle$ where G is the gateway that acts as a group initiator and manager and N is a set of nodes $n_1, n_2, \ldots, n_k$, and *E* is the set of edges from G to n_i representing bi-directional communication links.

We assume that G is capable to do as many scalar elliptic curve point multiplications as the number of nodes. G is a trusted entity which acts as the group-manager which includes creating new groups, adding new nodes to the groups a maintaining keys and the members of nodes of each group.

It is also assumed that this communication technology and sensor nodes support the transaction of multicast messages and is considered that all the entities in the network possess the same security associations and perform identical cryptographic functions. In our specific tests the entities support the following security functionality:

- digital signatures using ECC, specifically Elliptic Curve Digital Signature Standard (ECDSA), by using the F_p curves defined by the NIST;
- establishing shared secret using ECC, specifically Elliptic Curve Diffie-Hellman (ECDH);
- Symmetric encryption scheme, specifically AES;
- hash functions, specifically SHA3.

A symmetric secret key $SymKey_i$ is used for securing the communication between a node n_i and the gateway G during the group-key initialization. The gateway loads $SymKey_i$ and its own public key $PubKey$ in each node n_i that will be part of the group before the execution of the protocol in order to encrypt the channel with a symmetric algorithm like AES and sign the messages. Moreover, in order to reduce the node's workload during the protocol execution, each node n_i generate a pair of public and private keys $PubKey_i$ and $PrivKey_i$.

TABLE I
LIST OF NOTATIONS FOR THE PROPOSED SCHEME

Notation	Description
G	Gateway
n_i	ith node
k	number of nodes
m	number of legitimate nodes
$PrivKey$	node's private key
$PubKey$	node's public key
$SymKey$	node's symmetric secret key
$\mathbb{F}_q$	finite field
$\mathbb{E}\|\mathbb{F}_q$	elliptic curve on the finite field $\mathbb{F}_q$
s_i	secret shared between ith node and the gateway
E	generic symmetric encryption/decryption function
MSK	final group shared secret (Master Shared Key)
PRF	Pseudo Random Function

IV. PROTOCOL DESCRIPTION

The proposed protocol focuses on different problems related to group-key establishment and management.

First, we consider the problem of establishing a group shared secret among a group of end nodes. We then show how a new node can be added to the group in such a way that the newly added node does not learn the previous group key or group communication. We also show how to remove a node from the group in such a way that it does not learn anything about the communications after it left the group. Finally, we analyze the problem of generating new session keys from the established group secret.

In this paper, we adopt a slightly loose definition of forward and backward secrecy. By forward secrecy, we mean that an attacker will not able to learn future session keys or group communications even if the attacker managed to learn the current session key. This also applies to which was a member of the group but that was later removed. Even that node had access to the session key when it was a member, it shall not be able to learn or derive session keys generated after it left the group. By backward secrecy, we mean that a newly added node will not be able to learn previous session keys and/or group communication before it joined the group.

A. Establishing Group Secret

In this phase, a device that wants to start a secure group, usually the gateway, broadcasts a "*new_group*" message including the new $group\_ID$, signing the message with its private key to guarantee authenticity. Now a specified timeout starts and the nodes that want to join the group reply to the broadcast message with a secret s_i generated with Elliptic-curve Diffie-Hellman (ECDH) protocol. To guarantee confidentiality and authenticity, the *join* message is encrypted with a symmetric key and signed with the node's private key. The gateway *G* collects the *join* replies until the timeout expires and then it decrypts the messages (indicating the willingness to join the group) and checks the authenticity. *G* computes the $group\_secret = \prod_{i=1}^{m} s_i$ where m is the number of nodes which sent the secret value s_i. *G* finally sends to each node n_i the value $partial\_secret = group\_secret/s_i$, encrypted with the symmetric key and signed with its $PrivKey$. This procedure is shown in Algorithm 1.

Algorithm 1 Group-key initialization's algorithm

Input: parameter $\mathbb{P}$
Output: *group_secret*
Initialization: chose $group\_id, group\_secret = 1$,
Broadcast group join request with *group_id*,
Set *timeout*

while $timeout$ not expired **do**
  for for each s_i received **do**
    $group\_secret = group\_secret * s_i$
for for each s_i received **do**
  $G \rightarrow n_i : \quad E_{SymKey}(group\_secret/s_i)$
return $group\_secret$

After receiving the message, each node n_i decrypts the message obtaining its own $partial\_secret$ and computes

$$MSK = partial\_secret * s_i * \mathbb{P}$$

where MSK represents the group shared key.

B. Adding a New Node

After establishing a secure $group\_secret$, the addition of a new node in the group may occur. When a new node new_node wants to join the group, it sends a message to G containing the $group\_ID$ and its shared secret s

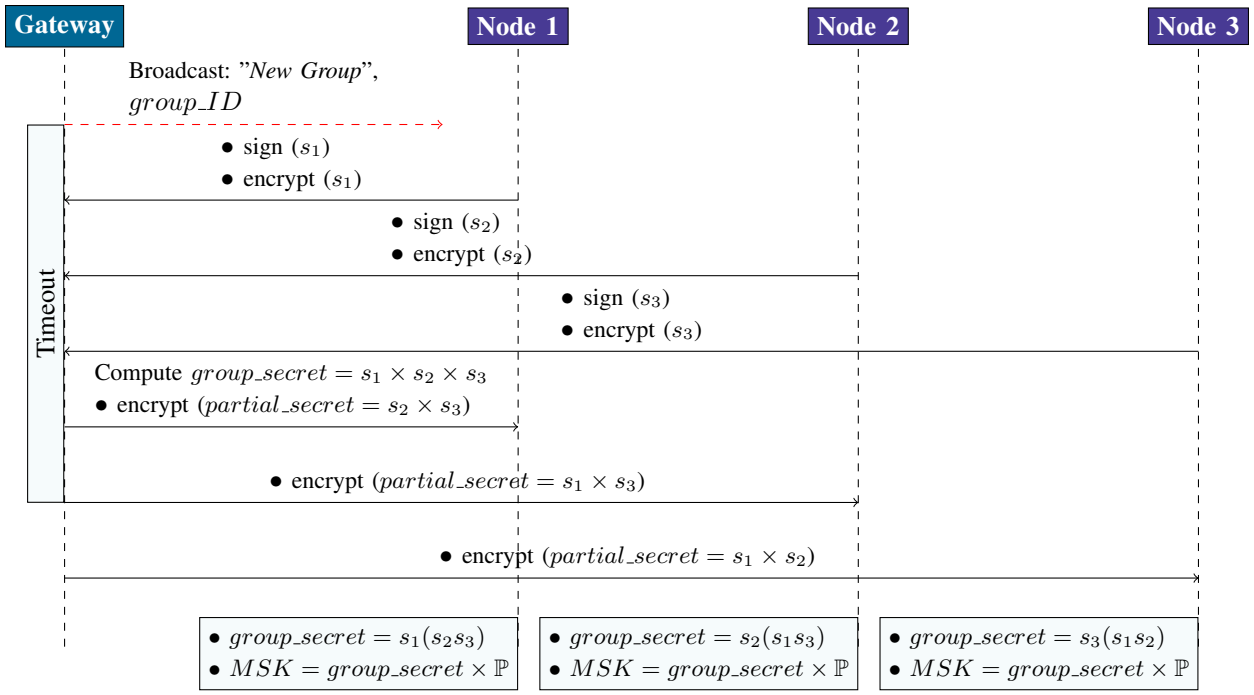

Fig. 2. Group Key Initialization: The figure shows the messages exchanged between G and three end nodes which replied to the join request before the specified timeout.

encrypted with a symmetric key $SymKey$. G recomputes the $group\_secret$ as before including the secrets from the new nodes and broadcasting the new_group_secret encrypting the message with the previous $group\_secret$.

Algorithm 2 New node addition's algorithm

Input: parameter $\mathbb{P}$
Output: *group_secret*
Initialization: chose the *group_id* of the group to join, the new node computes the secret s,

1: $new\_node \rightarrow G: \quad E_{SymKey}(group\_ID, s)$
2: $old\_group\_secret = group\_secret$
3: $group\_secret = s * old\_group\_secret$
4: **for** for each node n_i already in the group **do**
5: $\quad G \rightarrow n_i: \quad E_{old\_group\_secret}(group\_secret/s_i)$
6: G picks a random s_i of a node already in the group
7: $partial\_secret = group\_secret/s_i$
8: $G \rightarrow new\_node: \quad E_{SymKey}(s_i\mathbb{P}, partial\_secret)$
9: new_node calculates $MSK = s_i\mathbb{P} * partial\_secret$
10: **return** $group\_secret$

In this phase we have to ensure that the new node isn't able to easily recover the old MSK. To solve this problem, G picks a random secret s_i from the nodes already present in the group and sends to the new_node a message, encrypted with a symmetric key, containing $s_i\mathbb{P}$ and $partial\_secret = new\_group\_secret/s_i$.

To obtain the MSK, the new node only has to multiply $s_i\mathbb{P}$ by $partial\_secret$ and it will not be able to recover the old MSK due to difficulty to obtain s_i from $s_i\mathbb{P}$. This procedure is shown in Algorithm 3.

C. Removing Node

Algorithm 3 Node removal's algorithm

Input: parameter $\mathbb{P}$
Output: *group_secret*
Initialization: remove the node n from the group $group\_id$

1: G picks a random s of a node in the group
2: $partial\_secret_i = group\_secret/(s_i s)$
3: **for** for each node n_i already in the group **do**
4: $\quad G \rightarrow n_i: \quad partial\_secret_i, s\mathbb{P}$
5: n_i calculates $MSK = s\mathbb{P} * partial\_secret_i$
6: **return** $group\_secret$

It can happen that one or more nodes have to be removed from the group for various reasons. In such an an event, the protocol has to ensure forward secrecy of the MSK to avoid the possibility that the removed node is able to get the new group secret. The gateway proceeds as described before picking up a random node's shared secret s and sends to each node n_i which belongs to the group a message containing a new partial secret $partial\_secret_i = group\_secret/(s_i s)$ and $s\mathbb{P}$. Each node can than compute MSK by multiplying $partial\_secret_i$ by $s\mathbb{P}$ and its shared secret s_i. The nodes that have left the group will not be able to compute the new

MSK due to the same reason described during the *node additions phase*.

D. Session Key Generation

After the MSK generation, it is possible to use symmetric key encryption to encrypt the group messages. The choice of using symmetric key encryption instead of public key encryption is motivated by the fact that public key encryption is slower and requires more computation power, a critical point in this class of constrained IoT devices.

Using the same key for more than one session makes the protocol vulnerable to many attacks [19], so we generate new and pseudorandom keys from the MSK. The session key can be produced and managed in different ways depending on whether the encryption scheme is block or stream cipher. In this section we consider those two methods separately. MSK with block ciphers: Suppose the block cipher of our choice takes n-bit key size. Therefore, every session key must be n-bit long. We assume we have a k-bit output hash function h, and a PRF that maps an input of l-bit key size to an output of size n. Note that the output of the PRF is computationally indistinguishable from an output of a random function from l to n. An l-bit input seed to our PRF is first formed by concatenating the results of a repeated application of a hash function applied to a nonce and MSK, as shown below.

$$seed = h(nonce_1 \parallel MSK) \parallel h(nonce_2 \parallel MSK) \parallel \cdots$$

A session key is then generated by feeding the seed into a PRF.

$$session\_key = PRF(seed)$$

And finally MSK is updated to the new session key. Note that the node sending a group message randomly selects as many nonce values as needed to create an l-bit output. The nonce values are sent in the clear to all nodes so that every nodes updates its session key similarly.

we produce a session key as long as the length of the text to be encrypted. This can be achieved by concatenating the results of the PRF on many seed values as follows.

$$\begin{aligned} seed1 &= h(nonce_1 \parallel MSK) \\ seed_2 &= h(nonce_2 \parallel MSK) \\ &\vdots \\ seed_n &= h(nonce_n \parallel MSK) \end{aligned}$$

We then apply the PRF to each seed seed and simply concatenate the results to generate a session key as long as required.

$$session\_key = PRF(seed_1)||PRF(seed_2)||\cdot$$

In both cases, encryption is done as follows :

$$Encrypt(M) = f(M, Sessionkey)$$

Where f could by any block cipher, such as AES, or stream cipher, such as RC4.

V. Experimental Setup

To check the feasibility and analyze the performances of our protocol, the scheme was implemented on the Contiki OS and simulated with the COOJA simulator [20]. COOJA is a network simulator which allows us to simulate a network of IoT resource-constrained devices. In our study we simulate Tmote Sky node, an MSP430 microcontroller based board whose details are shown in Table II.

TABLE II
TMOTE SKY

Resource	
Operating Voltage	3 V
Microcontroller	(16 bit)8 MHz
RAM	10 KB
ROM	48 KB
Low Power Mode (LPM)	0.0545 mA
Current consumption TX mode	19.5 mA
Current consumption RX mode	21.8 mA
Ticks/second	327680

COOJA gives us an interface where we can analyze the messages exchanged in the network during our simulation and tools for the energy consumption calculation and timing.

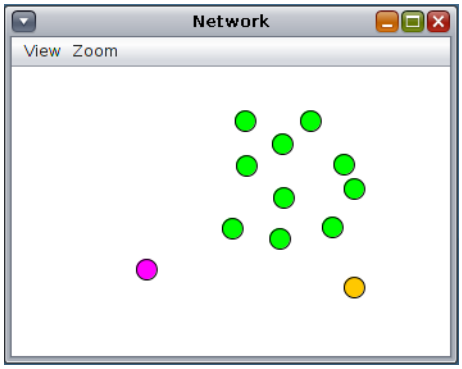

Fig. 3. Network of Sky Mote devices simulated with COOJA, where the green ones are the nodes, the purple one is a node that will be added after the first execution of the protocol and the orange one is the gateway.

To implement the ECC operations, we have used nano-ECC [1], a C library which provides all the necessary Elliptic Curve primitives and functions for our protocol. It has built in features that allows us to run ECC even on very constrained environments (i.e. 8-bit processors) and tiny architectures which do not support a heap. This ECC implementation is based on the prime finite field, where basic arithmetic operations can be effectively optimized if pseudo-Mersenne primes are used [21].

TABLE III
NODE MEMORY USAGE [BYTES]

Text	**Data**	**bss**	**dec**	**hex**
30763	332	5552	36647	8f27

[1]https://github.com/iSECPartners/nano-ecc

In Table III we show the memory usage of the node in bytes. *text* shows the size of the code section in bytes (this will typically be in ROM). *data* and *bss* show sections that contain variables, stored in RAM.

VI. RESULTS

In this section we present the performance results of our scheme comparing it with another scheme [14] previously proposed for IoT applications. The storage overhead analysis explains the memory utilized to store the parameters used by the protocol. The energy analysis is based on the estimate energy consumption of the computation and communication energy cost of the protocol.

Security analysis explains how we mitigate the commons security threats.

A. Storage Overhead

The amount of memory required to store security parameters and key space is considered the storage cost. The total number of devices in the network is $d = m + 1$, which is the number of nodes m plus the gateway G. Calculations are performed for the *secp192r1* curve ECC operations, estimating the dimension of an EC point $P = ((P_x), (P_y))$ of 48 byte. In our scheme, every node in the group is preloaded with a symmetric key $SymKey$ used for a symmetric key encryption during the group-key establishment and the gateway's $PubKey$ to verify the signature of each message sent by it. Each node n_i will store its public key $PubKey_i$, private key $PrivKey_i$ and the secret s_i shared with the Gateway. Finally, every node has to store also the master shared key MSK, which is represented only by the x coordinate of an EC point [22]. We further applied the secure hash standard SHA-3, hence the hash digest length is 192 bits.

TABLE V
SPACE UTILIZED BY EACH PARAMETER USING *secp192r1* CURVE

Resource	Dimension (Bytes)
$PubKey$	48
$PrivKey$	24
$GroupID$	1
$GatewayPubKey$	48
MSK	24

B. Energy Consumption Analysis

The performance analysis represents a major concern at this time and it is based on the estimated energy consumption of the computation and communication energy cost of the protocols. For the potential enormous number of IoT devices, this will help saving the energy significantly in a long term. For disposable-battery-powered devices, using efficient protocol means changing battery less often. This helps reducing battery consumption which should be recycled in certain way. The energy consumption and duration for each step of the protocol using 192 bit long keys is shown in Table VI.

TABLE VI
ENERGY CONSUMPTION AND DURATION FOR EACH STEP OF THE PROTOCOL USING *secp192r1* CURVE ECC OPERATIONS .

	Energy (mJ)	**Duration (s)**
Generate signature (ECDSA)	2.197	19.68
Verify signature	2.006	17.112
ECDH	2.197	19.206
EC Point Multiplication	2.085	19.199
AES Encrypt	≈ 0	0.010
AES Dencrypt	≈ 0	0.014

Comparing to hashing and AES operations, EC point operations (i.e., point addition and multiplication) and radio activities (transmitting and receiving data) are considered the most expensive operations. By reducing the number of messages, the power usage can be decreased, letting the devices in the network operate longer.

The computational overhead for each phase of our protocol and protocol 2 [14] is shown in Table IV, where AES_E and AES_D, which represent encryption and decryption using AES protocol, result not relevant for the analysis. In Table IV, PM is the Point multiplication , V and S are the verification and generation of a signature respectively and PA is the point addition.

In the Protocol 2 proposed in [14], to establish a group key it uses four point multiplications and three point additions, which represent a higher overhead compared to our protocol. In fact, assuming that a signature or a signature verification is comparable to point multiplication operation, our scheme needs only 3 point multiplications. Moreover, our protocol covers the different scenarios which can happen in a dynamic network, like the node addition or the node removal. This means an higher scalability than re-executing the protocol every time.

Instead, the number of exchanged messages between a node and the gateway results higher in our protocol, with one more received message. The total amount of bits exchanged between a node and the gateway is 1736, justified by a higher level of security offered by the chosen curve (in [14] has been used $secp160r1$ which has only 160 bit length keys).

The complete energy consumption for the group key establishment phase is approximately 109 mJ, where 49 mJ are spent on the messages transmission. Our protocol consumes around 8.4% less energy than the protocol 2 in [14]. Taking into account that with two Zinc-carbon AA batteries of 1.5 V nominal voltage and 800 mAh average capacity,the available energy is 8640 J. Consequently, this value correspond to 0.0013% of the total available energy for one complete execution of our protocol. Taking only the execution of the protocols into account, it implies that the group key establishment phase can be executed around 79225 times, which is the 30% more than [14] .

C. Security Analysis

The proposed group-key generation protocol is developed using a one-way accumulator based on one of the lightest PKC schemes, ECC. Thought it is comparatively more

TABLE IV

COMPUTATIONAL OVERHEAD DURING EACH STEP.

	Node Computation Overhead	
Phase	**Our Protocol**	**Protocol 2 [14]**
MSK establishment	$Overhead = V + S + (AES_E \approx 0) + (AES_D \approx 0) + PM$	$Overhead = 4PM + 3PA$
Adding new node	$Overhead = V + S + (AES_D \approx 0) + PM$	$Overhead = 4PM + 3PA$
Removing nodes	$Overhead = V + S + (AES_D \approx 0) + PM$	$Overhead = 4PM + 3PA$

expensive than symmetric key algorithms, it is inherently secure due to the PKC characteristics. The advantage of using ECC is that it provides an equal security as RSA, however, with less overhead (e.g., 160-bit ECC equals RSA with 1024 key size).

In our work, MITM attack resistance is provided by including digital signatures. An attacker can try to impersonate a legitimate node but that requires stealing the privately stored information, i.e. private key and s by physically examining the node. This is a feasible attack for which we do not a propose a solution in our paper. Moreover, another type of attack which this paper does not cover protection is Denial of Service attack (DOS).

VII. CONCLUSIONS

Group communication in IoT should be done securely, maintaining security requirements such as confidentiality and privacy. Secure group communication requires the sharing of common cryptographic parameters such as keys used for encryption, authentication and integrity checking. This paper designs and analyzes a lightweight secure group key establishment mechanism for multicasting in the context of IoT applications. Our scheme leverages the notion of one-way cryptographic accumulators to enable end nodes connected to a common gateway to establish and manage a secure group channel which guarantees forward and backward secrecy. According to the performance evaluations results, computation and communication energy consumption, the protocol proposed in this paper shows the capacity to be tolerated by resource-constrained devices.

Our designed scheme has showed the feasibility of using the arithmetic of elliptic curves as one-way accumulators, allowing us to negotiate shared secrets and generate membership witnesses with a limited number of computations at the nodes , which have limited computational capabilities. Moreover it results easily scalable when new nodes are added or removed to the network, making the scheme suitable for dynamic environments like IoT networks.

REFERENCES

[1] T. Heer, O. Garcia-Morchon, R. Hummen, S. L. Keoh, S. S. Kumar, and K. Wehrle, "Security challenges in the IP-based Internet of Things," *Wireless Personal Communications*, vol. 61, no. 3, pp. 527–542, 2011.

[2] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A vision of IoT: Applications, challenges, and opportunities with China Perspective," pp. 349–359, 2014.

[3] I. Jawhar, N. Mohamed, and D. P. Agrawal, "Linear wireless sensor networks: Classification and applications," *Journal of Network and Computer Applications*, vol. 34, no. 5, pp. 1671–1682, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2011.05.006

[4] G. Margelis, X. Fafoutis, G. Oikonomou, R. J. Piechocki, T. Tryfonas, and P. Thomas, "Practical Limits of the Secret Key-Capacity for IoT Physical Layer Security," *Communications (ICC) 2017 IEEE International Conference on*, pp. 311–316, 2016.

[5] L. Veltri, S. Cirani, S. Busanelli, and G. Ferrari, "A novel batch-based group key management protocol applied to the Internet of Things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2724–2737, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.adhoc.2013.05.009

[6] T. Gebremichael, U. Jennehag, and M. Gidlund, "Lightweight IoT Group Key Establishment Scheme Using One-way Accumulator," *IEEE The International Symposium on Networks, Compunters and Communications (ISNCC'18)*, 2018.

[7] C. A. Kumar, R. S. Lakshmi, and M. Preethi, "Implementing secure group communications using key graphs," *Defence Science Journal*, vol. 57, no. 2, pp. 279–286, 2007.

[8] S. L. K. Rahman, S. Kumar, O. Garcia-Morchon, E. Dijk, and Akbar, "DTLS-based Multicast Security in Constrained Environments," 2015.

[9] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-Hellman key distribution extended to group communication," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 31–37, 1996.

[10] E. Bresson, O. Chevassut, and D. Pointcheval, "Provably secure authenticated group Diffie-Hellman key exchange," *ACM Transactions on Information and System Security*, 2007.

[11] J.-M. Bohli, M. Isabel González Vasco, R. Steinwandt, J.-m. Bohli, M. I. González Vasco, and R. Steinwandt, "Secure group key establishment revisited," *Int. J. Inf. Secur*, vol. 6, pp. 243–254, 2007.

[12] I. Ingemarsson, D. T. Tang, and C. K. Wong, "A Conference Key Distribution System," *IEEE Transactions on Information Theory*, 1982.

[13] A. Perrig, R. Szewczyk, V. Wen, D. Culler, J. D. Tygar, A. Perrig, and R. Szewczyk, "SPINS: Security Protocols for Sensor Networks SPINS : Security Protocols for Sensor Networks," *Wireless Networks*, vol. 8, no. September, pp. 521–534, 2002.

[14] P. Porambage, A. Braeken, C. Schmitt, A. Gurtov, M. Ylianttila, and B. Stiller, "Group key establishment for enabling secure multicast communication in wireless sensor networks deployed for IoT applications," *IEEE Access*, 2015.

[15] C. Research, "STANDARDS FOR EFFICIENT CRYPTOGRAPHY SEC 1: Elliptic Curve Cryptography," 2000.

[16] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing elliptic curve cryptography and RSA on 8-bit CPUs," *Lecture notes in computer science*, pp. 119–132, 2004.

[17] "Energy analysis of public-key cryptography for wireless sensor networks," no. PerCom, pp. 1–5, 2005.

[18] J. Benaloh and M. de Mare, "One-Way Accumulators: A Decentralized Alternative to Digital Signatures," in *Advances in Cryptology — EUROCRYPT '93*, T. Helleseth, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 274–285.

[19] P. Syverson, "A taxonomy of replay attacks [cryptographic protocols]," *Proceedings The Computer Security Foundations Workshop VII*, pp. 187–191. [Online]. Available: http://ieeexplore.ieee.org/document/315935/

[20] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with COOJA," *Proceedings - Conference on Local Computer Networks, LCN*, pp. 641–648, 2006.

[21] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab, "NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks."

[22] V. S. Miller, "Uses of elliptic curves in cryptography," *Advances in Cryptology CRYPTO 85*, no. January 1985, pp. 417–426, 1986.