IOT-1-PASS-SECURITY: 1(ONE)-PASS AUTHENTICATED KEY AGREEMENT PROTOCOL FOR ENERGY CONSTRAINT IOT APPLICATIONS

Mehrdad Aliasgari, Garrett Chan, and Mohammad Mozumdar
Department of Computer Engineering and Computer Science,
Department of Electrical Engineering
California State University, Long Beach
Emails: mehrdad.aliasgari@csulb.edu, gchan310@gmail.com,
mohammad.mozumdar@csulb.edu

Submitted: Nov. 30, 2015 ***Accepted: Mar. 22, 2016*** ***Published: June 1, 2016***

Abstract- IoT data security is one of the core unresolved challenges in IoT community. Lack of resource-efficient authenticated secure key exchange methods among resource- constrained IoT devices makes man-in-the-middle attacks a serious vulnerability. In this regard, we propose 1(One) pass Authenticated Key Agreement (AKA) protocol for IoT applications. This protocol requires only one round of communication among the sender and receiver to establish a secure session, providing a balance between security (data confidentiality with integrity) and performance. We implemented and performed comprehensive power consumption and timing analysis of our implementation on Contiki platform to demonstrate the efficiency of the proposed protocol.

Index terms*: IoT, Sensor Networks, Security, Identity Based Encryption, Elliptic Curve Cryptography.*

I. INTRODUCTION

Uses of sensing based Internet of Things (IoT) applications have been expanding rapidly in many fields such as factory and building automation, environmental monitoring, health care, and in a wide variety of consumer and military application areas. Advancements in micro electro-mechanical systems and wireless communication have motivated the development of small and low power sensors and radio equipped modules that are replacing traditional wired sensor systems. These modules can communicate with each other by radio to receive and transmit data, and form networked embedded systems. Hence sensing based IoT applications have shown much promise to become a driver of current and future cyber physical system design. Currently, researchers are developing various types of software applications at different protocol level of IoT. Implementation of sensing applications is usually done using mostly battery powered micro sensing platform. In academia, researchers use operating systems, such as TinyOS [14], Contiki [6], ZigBee [3] stack etc., whereas industry uses their proprietary stack to develop applications in IoT devices.

Developing IoT applications in micro-sensing device is a challenging task, because most of the available sensor nodes (also called “motes”) on the market (such as MTM-CM5000-MSP [17] ,TelosB[26] and others) have tight constraints on computation and communication resources. Moreover, data sensitive applications like bio- medical sensing, industrial automation, military applications and others require stringent commitment of data and key security. It is fundamental that traditional sensor networks should have the capability to avoid eavesdropping, injection and modifications of packets [11]. If a suitable security capability is not available the efficiency and reliability of the network will be decreased [27]. Hence, security services like authentication and key management are crucial to IoT applications. To enforce security in IoT applications, security services and protocol based on PKC (such as SSL, IPsec, etc.) are widely used. For example, PKC is employed to bootstrap symmetric session keys and also to authenticate messages between sender and receiver. Traditionally, protocols and algorithms based on public key require extensive computation resources which are not typically available in IoT sensing platforms.

Traditional key agreement protocols allow parties to agree on a secret key at beginning of each communication session. The communication channel is assumed to be insecure. To this end, the parties exchange messages and use local secret values. However, the adversary is able to read and modify the messages that are transmitted through the channel. There are two different types of adversary:

1. Eavesdropper: An adversary that observers the communication between two nodes and tries to learn information about underlying messages. This adversary does not modify/delete/cancel any message exchanged among honest parties.
2. Malicious: This adversary is allowed to modify/delete and drop any communication between two honest parties at any time. Also, this adversary can initiate a session with an honest party. This often happens after compromising an honest party and trying to connect to an uncorrupted party with the network. This type of adversary can perform a strong class of attacks known as Man-In-The-Middle (MITM) attacks. In MITM, the adversary places in between the two honest parties and intercepts the communication. In key agreement protocols, the adversary forges the honest receiver to the honest sender and also forges the honest sender to the honest receiver. Therefore, each honest party falsely assumes that it has established a secret key to the other.

Note that using a shared "network secret key" fails to provide security. If one node is corrupted by the adversary then the adversary will have access to this shared secret and the security of the entire network will be compromised [30]. We can't rely on a solution with fixed shared pairwise secret keys since this approach is not efficient, dynamic or scalable. Therefore, we need a scalable and efficient solution that enables any node to establish a secure channel to another entity without a pre-shared key or interaction with a third trusted center. In addition and to prevent MITM attacks, each party needs to authenticate the other and establish a secret key that is private from all other nodes (with possible exception of a root/master node). Authenticated Key Agreement (AKA) protocols are designed to address this issue. However, the computational and communication complexity of such protocols is considerable especially in constraint environments. Communication complexity is often measured in the number of messages exchange between parties. On the other hand, computation complexity is measured in the most time consuming operations. AKA protocols usually rely on public key cryptographic operations to be scalable. Therefore the number of exponentiation or similar operations is the measure of computation complexity. However, research has shown that communication complexity has a more prominent impact on battery consumption than computation. Therefore, the fewer message exchanges the better a protocol performs.

A one-pass AKA protocol is a key agreement protocol where the initiator only sends one message (however long) to the responder and both parties are able to compute a session locally using the transmitted message and internal secret values. This type of AKA protocols finishes only in one round. If there are two rounds involved we call such a protocol a two-pass protocol.

Three major and different notions of security in Authenticated Key Agreement (AKA) protocols are as follows:

1. Known-Key Secrecy (K-KS): In this notion, an adversary has gained access to session keys of other communication sessions. A protocol under this notion should provide security for the current session which is under attack. This is considered as the minimum level of a secure AKA protocol.
2. Perfect Forward Security (PFS): In this case, if an adversary can learn the long-term private key of a party then the adversary should not be able to recover the previous agreed session keys. It was shown by Krawczyk ([13]) that no protocol achieves PFS with two or fewer messages.
3. Key-Compromise Impersonation (K-CI) Security: In this case, an adversary has learned the long-term private key of a party, A. If the adversary cannot masquerade as another party (say, B) to A then the underlying AKA is K-CI secure. The work in [13] mentions that protocols which rely on using long-term static keys based on the difficulty of the discrete log problem to achieve a session key are insecure against K-CI attacks.

In this work, we consider only one-pass AKA solutions. This is due to the computational and communication restrictions on the IoT devices. Therefore, we can only achieve MITM and K-KS security against an active adversary. We argue that this is a valid assumption since in IoT applications often the devices are too limited in their computation and communication power and expensive solutions that provide higher security are too demanding. Thus, it is considered the best to protect a party's long-term private key than implementing costly protocols that offer security in situations where the long-term private key is compromised. In fact, in IoT applications, MITM attacks are more serious issues since an attacker can easily impersonate a valid mote of a network and recover all messages sent and received from that node. Thus, our goal is to design and implement a secure AKA protocol with low power consumption footprint for resource-limited sensor nodes. We then measure execution time and power consumption of our proposed method. We argue the feasibility of adapting our AKA protocol in various applications that rely on constraint devices. Elliptic curve cryptography (ECC) is chosen since among PKC solutions for the same level of security, it requires smaller key sizes. Hence, ECC solutions can reduce memory and power consumption while providing an acceptable level of security.

II. RELATED WORK

Contiki ([7]) is an open source, portable and event-driven Real-Time OS(RTOS) for memory-constrained embedded systems with a focus on low-power communication. Thus it is a good platform candidate for Internet of Things devices. Contiki is able to support various hardware platforms such as the Tmote Sky and MICAz. These platforms are built using low-power micro-controllers such as the TI MSP430 and the Atmel AVR. Contiki kernel is not protected from applications. This allows an application to access and/or corrupt global or event data that belongs to the underlying operating system.

The work of Casado et al. in ContikiSec ([4]) provides symmetric key security functionality on network layer for Contiki OS. Their work only considers offering three security mode options: confidentiality-only (ContikiSec- Enc), message authentication-only (ContikiSec-Auth), and authentication with encryption (ContikiSec-AE). Note that ContikiSec does not address the issue of key agreement (let alone AKA) and is only limited to providing symmetric key security functionalities. Karlof et al. in [12] had done a similar work for TinyOS in link layer and [8] considered network layer symmetric key functionality for IPv6 in WSN. The authors are not aware of any efficient key agreement tool designed for Contiki.

Identity Based Encryption (IBE) is a public key encryption scheme where the public key is generated using the identity of the entity ([2]). IBE requires a master public/private key. This pair of keys is generated by a master entity known as Private Key Generator (PKG). Anyone can compute the public key of an entity given their ID and the master public key. However, only PKG can compute any entity's private key. IBE is used to prevent MITM attacks. However, the idea of using IBE in WSN isn't new (e.g., [20, 28, 29, 5, 24]). Oliveria et al. in [20] implement ECC pairing that leads to establishment of long term secrete keys. In contrast, this work examines authenticated secure communication session establishment in Contiki with timing as well as power consumption experiments. In this work we demonstrate the computational and power consumption impact of our IBE-ECC-based protocol in Contiki sensors with TinyECC as the main library. We demonstrate the performance of our protocol in this setting. To the best of our knowledge, there is no efficient IBE-based work for authenticated session key agreement in WSN. In this work, not only we propose an efficient and secure IBE based AKA protocol but also we perform time and power consumption analysis of our implementation.

III. PROPOSED METHOD

We have an elliptic curve with parameters *P, a, b, p, g, h* where *p* is the prime modulus and *g* is the order of *P* which is a point on the elliptic curve constructed by *a, b*. For more on elliptic curve cryptography (ECC) refer to [9]. *h* is any secure hashing function of appropriate digest size.

To provide dual authentication in our protocol we will rely on a specific class of encryption, Identity Based Encryption (IBE). In IBE the public key of any user is tied to their identity. Therefore, any node can compute the public key of another node locally as long as the receiver's ID is known and the sender has access to a master public key. In this setting, there is a dedicated entity (e.g., master node or PKG) that generates a pair of master public and private key. Any node's public/private key is computed using the master keys and the node's identity. A simple IBE-ECC encryption scheme is demonstrated below. We assume that the master entity has generated an array of random integers with appropriate size (X_i for $1 \leq i \leq n$). This array is the private key of the master entity. Consequently, the master public key will be an array *Y* where for each element we have $Y_i = X_iP$ using ECC point multiplication. The master public key is published public key or pre-programmed in any node's memory.

For a node of ID "str", the Master node can calculate its private key (random integer X_{str}) and public key (random point of the curve, Y_{str}) as follow:

$X_{str} = \sum h_i(str).X_i$ where h_i represents the i^{th} bit value of the output of function *h* and $Y_{str} = \sum h_i(str).Y_i$.

Note that to compute any node's public key, only its identity ("str") and the master public key is required. In our approach we use the work of Tan et.al. from [25].

If party A wants to talk to party *B* then *A* reaches out to obtain *B*'s public key (Y_{strB}). *A* sends to *B* in plaintext a random number, *r*. Since we use an IBE-ECC as the underlying public key scheme then parties need not to exchange their public key to establish a secure session. Each party computes the public key of the other party locally. Therefore, MITM attacks are prevented. The use of a random r ensures that the generated key belongs only to the current session. Below we present our 1-pass AKA protocol. We assume that the parties can compute each other's public key locally and store them temporarily in memory for the sake of AKA execution. This means that all parties have the master public key stored in their memory.

1-passAKA

1. Party A generate a fresh random number r of fixed length. A sends to B in plaintext the message: $ID(A), r$
   where $ID(A)$ represents the ID of party A.
2. Both parties locally compute Elliptic curve point $Q = XstrAYstrB = XstrBYstrA = (Qx, Qy)$ using their secret keys and other party's public keys.
3. Compute locally Kr = HMAC(Hash(Qx)||Hash(Qy), r||$ID(A)$||$ID(B)$) where || denotes concatenation. HMAC uses Hash function which is SHA256 in our approach.
4. Compute the encryption key, Ke = HMAC(Kr, r||$salte$) where $salte$ is publicly known. We can assume that for a pair A, B we have $salte$ = $ID(A)||ID(B)||string(^tEncKey^t)$.
5. Compute the MAC key as Km = HMAC(Kr, r||$saltm$) where $saltm$ is publicly known. We can assume that for a pair A, B we have $saltm$ = $ID(A)||ID(B)||string(^tMacKey^t)$.

The security of our AKA solution can be stated as follows:

Theorem 1 *Assuming security of prior building blocks, the above protocol is secure against an eavesdropper.*

Proof 1 (Proof sketch) *We sketch the security of the above protocol using the work of [1]. In [1], Bellare showed that if the underlying hash function of an* HMAC *is a Pseudo Random Function (PRF) then the* HMAC *is PRF as well. A PRF is a keyed function where a polynomial time adversary, A without the key cannot distinguish the output of the function from a truly random value unless with negligible probability. If we replace our* SHA256 *building block with a true PRF (i.e., in a random oracle model) then the output of* HMAC *in line* 3 *of the above protocol is indistinguishable from a true random value if Q is kept private. This confirms eavesdropper and K-KS security of our approach.*

As for complexity analysis of the above protocol, the dominant computational operation is elliptic curve multiplication (in line 2). However, we note that this computation can be carried out in advance and the result (Q) can be stored in memory to be used for future session establishment. As for communication complexity, we observe that only one message is sent (in line 1). The size of this message depends on the size of ID and size of r. We advise against re-using r and therefore the size of r should be proportional to the logarithm of the

total number of sessions that will be established between a pair of nodes. In our implementation we used 32 bits for size of r.

a. Data Integrity Tag Segmentation

Data integrity and message authentication is essential to data security. For any message before being sent in an insecure channel, we compute HMAC (hash message authentication code) and append it to the original message. We use the approach of Encrypt-then-MAC (Etm) as outlined in ISO/IEC 19772:2009 [10]. In this approach, given a message m, the sender first encrypts m under the encryption key (k_e) and produces $c = \text{Enc}k_e(m)$. Then the sender computes $t = \text{HMAC}k_m(c)$ where k_m is integrity key and is independent of k_e. The underlying hash function in HMAC is a secure hash function. In our experiments we used either SHA1 (160 bits of message digest) or SHA256 (256 bits of message digest). The sender sends buffer = $c||t$. The receiver computes a new tag (t^t) and accepts the message only if $t = t^t$. The length of sent message is extended by the size of t (message digest). However, we note that in constrained devices, the exchanged data samples are small (as few as two bytes).

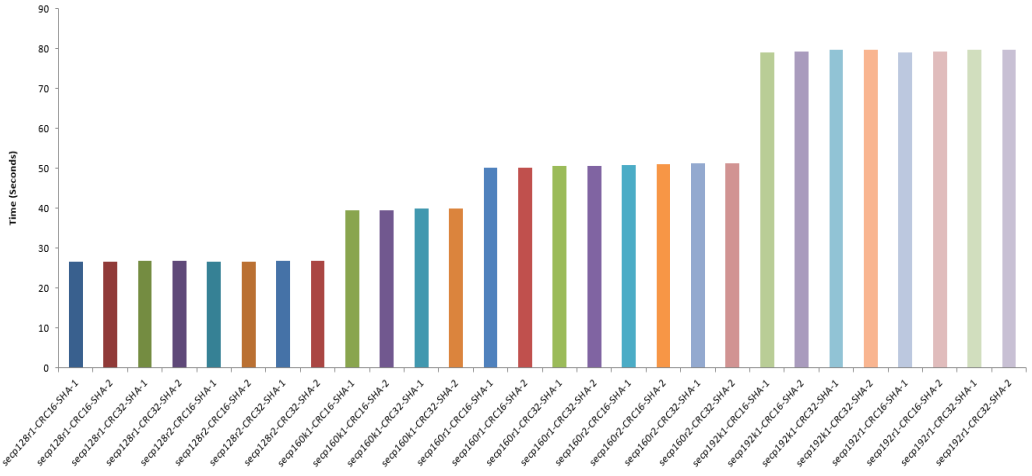

Figure 1: ECC AKA Total Time

Communication consumes considerable amount of energy, thus the fewer packages sent the longer the battery can last. One solution would be to postpone message integrity check. In this approach, for every M messages sent, one tag is computed and sent. It requires to maintain all M pack ages in memory at both sender and receiver. In addition, the integrity of a message is checked at a later time. This carries a risk that the receiver may act on a corrupt message only to discover later that the package was modified. This is another trade-off between security and cost (power consumption). The value of M is, therefore, critical. This value depends on the size of payload that is exchanged between two nodes in an IoT network.

We note that the payload size is often smaller than the encryption block size (AES's block size is 16 bytes). If the payload size (*PL*) is less than the block size (*BL* = 16 in bytes) then we set *M* = *BL/PL* Otherwise we set *M* = 1. We then define the message expansion rate (MER) as the ratio of total sent message including integrity tag versus message size with no integrity. This ratio indicates the overhead associated with message integrity. If we follow the segmentation rule above regarding *M* and indicate hash digest size with *HD* with encryption in counter mode then we have:

$$\mathrm{MER} = (MPL + HD)/(MPL) = 1 + HD/(BL) \quad (1)$$

Note that the overhead shown above is less than the overhead with no integrity segmentation (MER = 1 + *HD/PL*). In fact, the reduction is $100(M - 1)/M$ percent. This reduced overhead also depends only the hash digest size since encryption block size is constant. In our experiment, we used payloads of size *PL* = 4 bytes and therefore *M* = 4 and MER was reduced by 75 percent.

IV. EXPERIMENTAL RESULTS

We implement the proposed authenticated session key agreement protocol on Contiki platform. We use TinyECC ([15]) as a software package for ECC-based operations. TinyECC is intended for TinyOS operating system and is implemented in nesC, but was converted to be used with Contiki in C. TinyECC supports SECP 128-bit, 160- bit, and 192-bit elliptic curve domain parameters. It supports all elliptic curve operations over Fp, such as point addition, doubling, and scalar point multiplication. We conducted our experiments with TMote Sky, ultra-low power wireless sensor nodes. TMote Sky is equipped with The TI MSP430, a family of ultra-low power micro- controllers. This device features a 16-bit RISC CPU, 2 16-bit timers, and constant generators that contribute to maximum code efficiency. We utilized some additional features in this work such as the Universal Asynchronous Receiver/Transmitter (UART), Watchdog Timer, and two 16-Bit Timers. We also utilized Cooja, a Contiki simulator ([21]). We tested our protocol on a simple application where if the sender's light sensor value passes a predefined threshold, the light value will be encrypted with a tag and sent to the receiver, in which the receiver will turn on or off its LEDs. The authors are not aware of any previous work that implements a one-pass AKA protocol in the above setting. Therefore, a comparison of experimental evaluations with related work is not possible currently.

a. Setup

Each node contains variables NodeID, PrivateKey, PublicKey, and MasterPublicKey are programmed by the base station (Master Node). The size of the MasterPublicKey array is dependent on the digest size of the hash function that is used in IBE-ECC-hash. We used two

cases of CRC-16 and CRC-32 in our experiments. All nodes have the same MasterPublcKey and each node has their own unique NodeID, PrivateKey, and PublicKey. NodeIDs all have same size. For message integrity we implemented two cases of SHA-1 (160 bit digest) and SHA-256 (256 bit digest). We used secp128r1, secp128r2, secp160k1, secp160r1, secp160r2, secp192k1 and secp192r1 for ECC parameters in our experiments. Therefore, there are total of 28 different cases in our experiments (seven cases of elliptic curves, two cases of IBE-ECC-hash and two cases of SHA). We set the AES key size to 128 bits as it provides 64 bits of security (due to birthday attack) and is a common choice for a lot of devices including IoT devices. We use counter mode for our encryption as it is a fast, secure and a common mode for IoT encryption applications. We also assumed node discovery was done and each sender node knows the RIME address of the receiver.

b. Execution Time Method

There are two methods used to measure the time consumed by software to perform various operations. The first method is to use MSPSim to execute the program and perform a profile dump which will output the amount of clock cycles that was spent in a particular function. This method relies on a simulator to be as accurate as the node itself. The second method is to utilize the MSP430 real-time timers, which is setup by Contiki. These real-time timers are highly accurate and can be utilize to capture execution time. There are two types of timers in Contiki which are utilized: CTimer and RTimer. These timers are hardware dependent which utilizes ACLK, with the Tmote Sky ACLK is a 32,768Hz clock. RTimer is a 16-bit counter that increments approximately every 30.517 micro-second and will overflow after approximately 2 seconds. In Contiki, CTimer is also a 16-bit counter that is configurable to increment based on CLOCK CONF SECOND, which must be a power of two. Using the Tmote Sky, CLOCK CONF SECOND is setup to be 128, which means the CTimer counter will increment every 7.8125 milli-seconds and the counter will overflow after approximately 8.53 minutes.

Depending on the function, using RTimer may not be the best option to capture the execution time due to chances of the timer counter overflowing. Instead, both CTimer and RTimer were used to see the difference. Using the compiler directive DEBUG TIME, it will compile the code that will output the execution time for a particular function. The compiler directive is used because of the additional code size and is only useful when gathering statistics. DEBUG TIME will compile two additional functions start statTimer() and stop statTimer(). statTimer() will store the initial timer values of CTimer and RTimer by using Contiki built-in functions clock time() and RTIMER NOW(). After the initial timer values are stored, the

function that we want to gather execution time will be executed, at the end stop statTimer() will be called which will compute the difference between the initial timer values with the current timer values, and print out the statistics.

Execution time for each function was measured using CTimer and RTimer. As an example, the function for computing the receiver's public key took 177/128 = 1.3828125 seconds by using CTimer, but with RTimer it calculated to be 45287/32768 = 1.3820495 seconds, which has more precision. In this case it is more useful to use RTimer. But, when looking at the function for generating the common secret in 1passAKA), it is useful to use CTimer because RTimer will wrap around, while CTimer will maintain the correct elapsed time. Table b. shows approximate execution time for each method of our 1passAKA handshake protocol using the above measurement method. Figure 1, shows the total amount of execution time it took for each combination of parameters. We can see that the overall time for the protocol when an ECC curve of 192 bits is used is approximately 80 seconds. This is compared to a ECC curve of 128 bits which is approximately under 30 seconds.

c. Elliptic curve addition and multiplication

As it can be seen from table b., elliptic curve addition (in function RECV PUBLIC KEY which computes the public key of other node) and multiplication (in function COMMON SECRET which computes point Q on the curve) from our 1pass AKA protocol take the longest time. In fact, the overall execution time of our protocol is mainly dominated by COMMON SECRET function. Any improvement in these two underlying building blocks will improve the overall execution time significantly. We decided to run stress tests on these two operations and study their execution time further. We note that in our experiments we ported TinyECC library for all elliptic curve operations in Contiki. There are other libraries such as mbed TLS (formerly known as PolarSSL [19]), MoTE-ECC ([16]), NanoECC ([23]) and TinyIBE ([22]). The authors intend to implement 1passAKA using various ECC libraries and provide a complete comparison of 1passAKA's performance in future. Nevertheless, our ECC operation stress tests using TinyECC library are of independent interest.

Figures 2 and 3 show the statistics gathered when doing a stress test of these operations (occurrences). Each operation was tested 1,000 times with the various curve parameters, each iteration was given random input values. Table 2 indicates a summary of our tests. The results of these stress tests are consistent with the values collected during execution time test of our 5 protocol shown in Table 1.

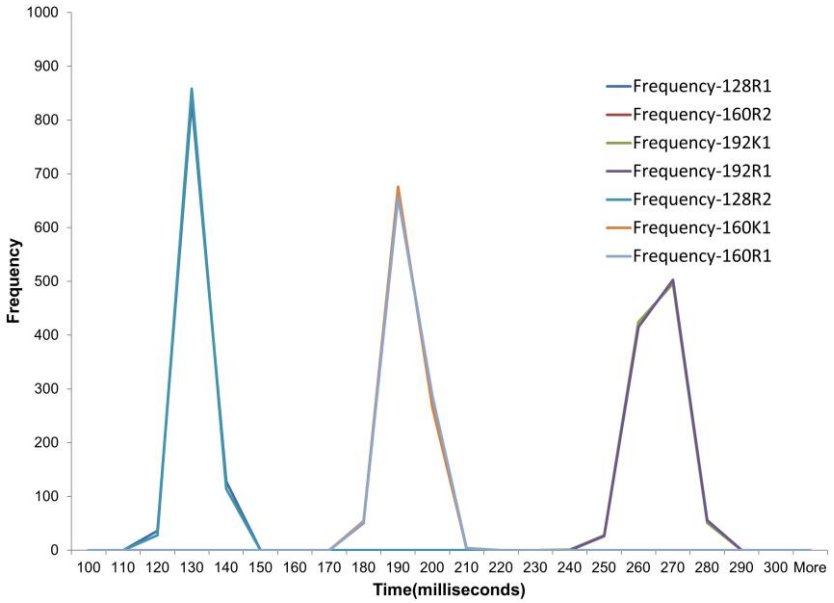

Figure 2: Elliptic curve addition test

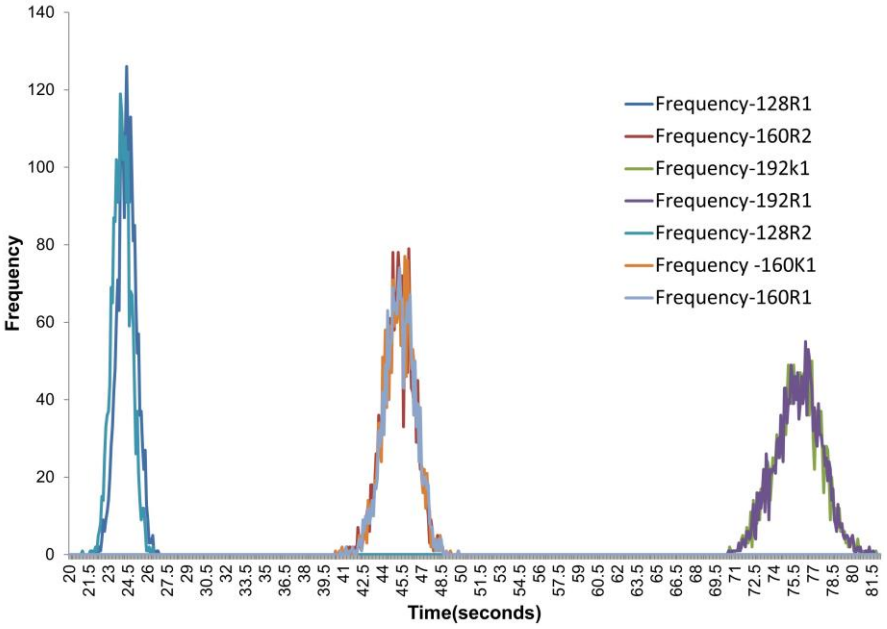

Figure 3: Elliptic curve multiplication test

	Curves	ECC Hash Function	Tag Hash Function	Init Params (ms)	RECV ID Hasing (ms)	Computing RECV Public Key (s)	Computing Common Secret (Q in sec)	Hasing of Q (ms)	Preparing KR Text for HMAC (ms)	KR HMAC (ms)	Preparing MACKey (KM) for HMAC (ms)	KM HMAC (ms)	Preparing KE TEXT for HMAC (ms)	KE HMAC (ms)	TOTAL TIME (s)
1	secp128r1	CRC16	SHA-1	0.122	0.092	0.89	25.59	0.549	0.183	22.46	0.214	25.54	0.214	25.54	26.66
2	secp128r1	CRC16	SHA-2	0.122	0.092	0.89	25.59	0.549	0.183	45.38	0.214	27.25	0.244	54.5	26.71
3	secp128r1	CRC32	SHA-1	0.122	0.061	1.11	25.61	0.275	0.183	25.45	0.214	25.54	0.244	25.63	26.9
4	secp128r1	CRC32	SHA-2	0.122	0.061	1.11	25.61	0.305	0.183	45.38	0.214	27.31	0.244	45.41	26.94
5	secp128r2	CRC16	SHA-1	0.122	0.092	0.89	25.59	0.549	0.183	25.51	0.214	25.54	0.244	25.54	26.66
6	secp128r2	CRC16	SHA-2	0.122	0.092	0.89	25.59	0.549	0.183	47.06	0.214	25.57	0.244	47.15	26.71
7	secp128r2	CRC32	SHA-1	0.122	0.061	1.11	25.61	0.305	0.183	25.45	0.214	25.63	0.244	25.63	26.9
8	secp128r2	CRC32	SHA-2	0.122	0.061	1.11	25.61	0.305	0.183	45.38	0.214	27.34	0.244	45.41	26.94
9	secp160k1	CRC16	SHA-1	0.122	0.092	1.39	38.13	0.671	0.183	25.45	0.214	25.54	0.336	25.57	39.62
10	secp160k1	CRC16	SHA-2	0.122	0.092	1.39	38.12	0.671	0.183	45.35	0.183	27.4	0.244	45.35	39.74
11	secp160k1	CRC32	SHA-1	0.122	0.061	1.8	38.13	0.366	0.183	27.19	0.214	25.54	0.244	25.57	40.11
12	secp160k1	CRC32	SHA-2	0.122	0.061	1.8	38.13	0.336	0.183	47.3	0.214	25.6	0.214	45.32	42.27
13	secp160r1	CRC16	SHA-1	0.122	0.092	1.34	48.77	0.671	0.183	27.16	0.214	25.57	0.336	25.54	50.3
14	secp160r1	CRC16	SHA-2	0.122	0.092	1.34	48.77	0.671	0.153	45.38	0.214	25.57	0.244	47.27	50.34
15	secp160r1	CRC32	SHA-1	0.122	0.061	1.68	48.83	0.366	0.183	25.51	0.214	25.57	0.244	25.54	50.38
16	secp160r1	CRC32	SHA-2	0.122	0.061	1.68	48.83	0.336	0.183	45.38	0.214	27.4	0.244	45.35	50.73
17	secp160r2	CRC16	SHA-1	0.122	0.092	1.37	49.52	0.671	0.275	25.45	0.214	25.57	0.244	25.54	51.07
18	secp160r2	CRC16	SHA-2	0.122	0.092	1.37	49.52	0.671	0.275	45.38	0.214	25.57	0.244	47.24	51.11
19	secp160r2	CRC32	SHA-1	0.122	0.061	1.73	49.6	0.366	0.153	25.45	0.214	27.19	0.244	25.63	51.52
20	secp160r2	CRC32	SHA-2	0.122	0.061	1.73	49.6	0.366	0.183	45.35	0.183	25.67	0.244	45.41	51.55
21	secp192k1	CRC16	SHA-1	0.366	0.122	1.91	77.2	0.763	0.153	25.51	0.214	27.19	0.244	25.54	79.3
22	secp192k1	CRC16	SHA-2	0.366	0.122	1.91	77.2	0.763	0.153	44.92	0.214	25.57	0.214	44.98	79.34
23	secp192k1	CRC32	SHA-1	0.366	0.092	2.46	77.13	0.397	0.153	25.45	0.214	27.31	0.244	25.57	79.77
24	secp192k1	CRC32	SHA-2	0.366	0.092	2.44	77.13	0.397	0.153	45.38	0.214	27.34	0.214	45.32	79.81
25	secp192r1	CRC16	SHA-1	0.366	0.122	1.92	77.2	0.763	0.153	27.19	0.183	25.54	0.244	25.54	79.3
26	secp192r1	CRC16	SHA-2	0.366	0.122	1.91	77.2	0.763	0.153	44.92	0.214	25.57	0.244	44.98	79.34
27	secp192r1	CRC32	SHA-1	0.366	0.092	2.46	77.13	0.397	0.153	25.45	0.214	25.63	0.244	27.31	79.77
28	secp192r1	CRC32	SHA-2	0.366	0.092	2.45	77.13	0.397	0.153	45.29	0.214	27.34	0.305	45.38	79.81

Table 2: 1Pass AKA Execution Time in seconds (s) or milliseconds (ms)

d. Power consumption analysis

In order to accurately measure the power consumption, we utilized the 2400 series SourceMeter. It allows a maximum of 1700 readings/second with a current range reading from 10A to 10pA. It can be programmed to be either a CCS (Controlled-Current Source) or a CVS (Controlled-Voltage Source) along with measuring the current, voltage, and/or resistance that the circuit is experiencing. We used LabTracer 2.0 program in our experiments, which is made specifically for the SourceMeter. The SourceMeter was set to be a CVS (at 3 volts) and to measure the current and power consumed during operation. Power was calculated as the product of measured current and voltage. Our TMote Sky was supplied with power directly by the SourceMeter. The measurements were performed on both the sending node and the receiving node. As for the sender, there are two possible communication mode: blocking mode (keeps sending till it receives an acknowledgment from the receiver) or non-blocking (sends the message only once). In addition, three elliptic curve parameters of 128r1, 160r1 and 192r1 were tested in our power experiments. CRC16 and SHA-1 were used in all cases. In all modes and parameters, the nodes started at idle, followed by the execution of our 1passAKA protocol. Then the nodes were idle for

a few seconds and finally the green LED was turned on to indicate the successful termination of the protocol.

Curve	MIN	MAX	AVG	STDEV
secp128r1	114.72	138.52	126.05	3.40
secp128r2	115.60	135.44	125.95	3.24
secp160k1	171.75	201.93	187.28	4.56
secp160r1	175.20	202.79	187.57	4.61
secp160r2	172.73	203.77	187.43	4.72
secp192k1	239.75	279.39	260.57	5.69
secp192r1	240.63	279.39	260.85	5.69

Table 3: Elliptic curve addition (milliseconds)

Curve	MIN	MAX	AVG	STDEV
secp128r1	21.66	26.77	24.35	0.72
secp128r2	20.92	26.31	23.96	0.72
secp160k1	40.38	49.71	45.24	1.20
secp160r1	40.80	49.78	45.24	1.20
secp160r2	40.85	49.73	45.17	1.20
secp192k1	70.53	81.73	75.79	1.76
secp192r1	70.48	81.52	75.81	1.77

Table 4: Elliptic curve multiplication (seconds)

	T_{128r1}	T_{160r1}	T_{192r1}	Power
Blocking Sender	26.64	50.31	79.32	5.4
Non-Blocking Sender	25.93	49.25	-	5.3
Receiver	25.78	49.63	79.25	5.3
Mean	26.11	49.73	79.28	5.3

Table 5: Results of the power tests in seconds and mill-watt

Figures 4, 5 and 7 show the power consumption of our 1passAKA protocol for the blocking sender, non-blocking sender and the receiver respectively. Table 5 summarizes execution time and power consumption results. We noticed that the power consumption in all communication modes and all elliptic curve parameters was about 5.3mW. This power remained constant throughout the execution of our protocol. Therefore, the energy consumption is estimated as

5.3mW times the execution time. The longer the execution time, the more energy is consumed. Thus 192r1 elliptic curve setting imposes the biggest energy consumption yet provides the highest level of security. These experiments show the trade-off between security and energy consumption.

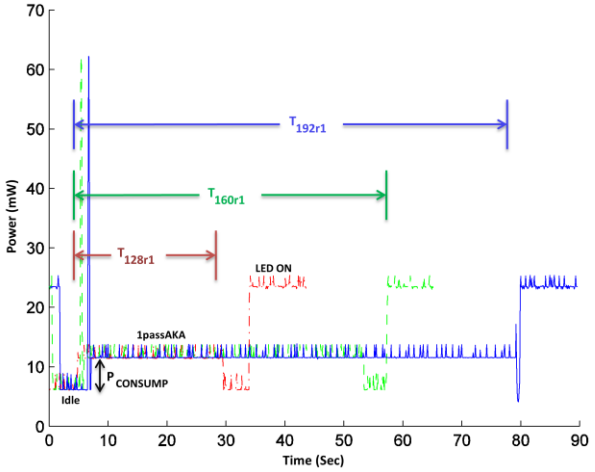

Figure 4: Power Graph for Blocking Sender

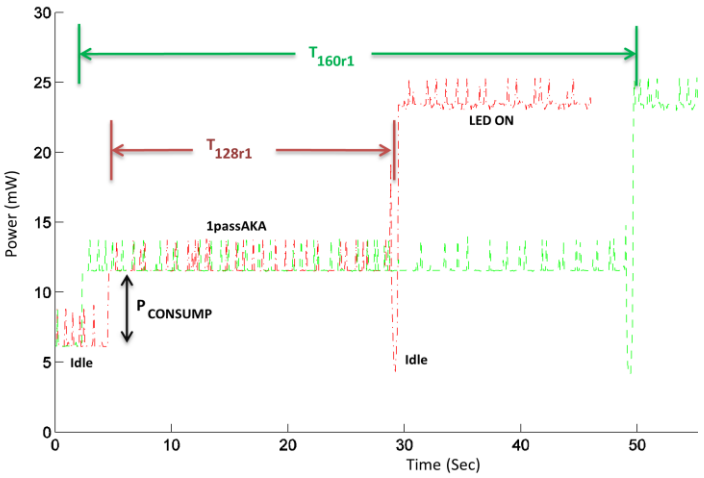

Figure 5: Power Graph for Non-Blocking Sender

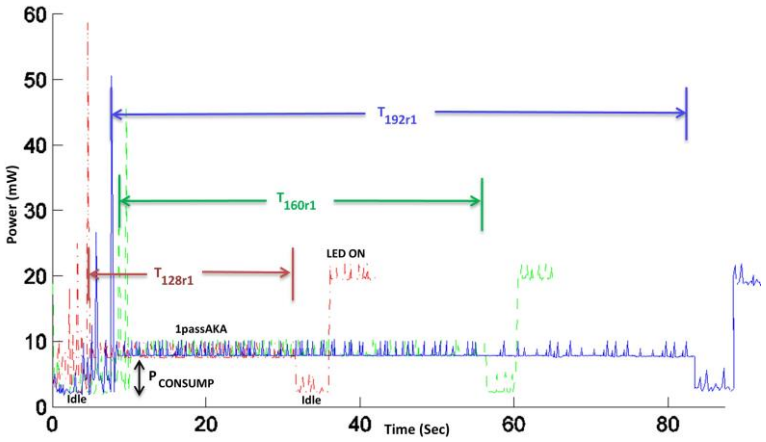

Figure 6: Power Graph for Receiver

V. CONCLUSIONS

In this work, we designed an efficient one-message-round authenticated session key agreement protocol to provide data confidentiality and integrity in IoT applications. Execution time and power consumption of our protocol in various settings were measured. We implemented our procotol for Contiki OS with TinyECC as the ECC library. We noticed that the main time consuming task in our protocol is the elliptic curve computation. Any optimization in this area is left as future work. In particular, studying the efficiency of different ECC libraries such as

mbed TLS [18] is an interesting future work. In addition, we noticed that our protocol consumes approximately 5.3mW which makes it suitable for IoT applications.

VI. ACKNOWLEDGMENT

This project was supported through a grant from Raytheon. We would like to expresses our gratitude to Mark Skidmore (Raytheon) for his support and valuable comments. We also would like to thank Seth Drake and Rafi Koutoby for helping us in this project.

REFERENCES

[1] Mihir Bellare. New proofs for nmac and hmac: Security without collision-resistance. In Advances in Cryptology-CRYPTO 2006, pages 602–619. Springer, 2006.

[2] Dan Boneh and Matthew Franklin. Identity-based encryption from the Weil pairing. SIAM Journal on Computing, 32(3):586–615, 2003.

[3] Zigbee Wireless Semiconductor Solutions by Ember. http://www.ember.com.

[4] Lander Casado and Philippas Tsigas. Contikisec: A secure network layer for wireless sensor networks under the contiki operating system. In Identity and Privacy in the Internet Age, pages 133–147. Springer, 2009.

[5] Barry Doyle, Stuart Bell, Alan F Smeaton, Kealan McCusker, and Noel E O'Connor. Security considerations and key negotiation techniques for power constrained sensor networks. The Computer Journal, 49(4):443–453, 2006.

[6] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, LCN '04, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.

[7] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In Local Computer Networks, 2004. 29th Annual IEEE Inter- national Conference on, pages 455–462. IEEE, 2004.

[8] Jorge Granjal, Edmundo Monteiro, and J Sa´ Silva. Enabling network-layer security on ipv6 wireless sensor networks. In Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE, pages 1–6, IEEE, 2010.

[9] Darrel Hankerson, Scott Vanstone, and Alfred J Menezes. Guide to elliptic curve cryptography, Springer Science & Business Media, 2004.

[10] Switzerland International Organization for Standardization, Gen‘eve. Iso/iec 19772, information technology security techniques authenticated encryption mechanisms. In Advances in Cryptology- CRYPTO 2006, pages 602–619. 2009.

[11] Yu-Ming Hsu Jiann-Lian Chen and I-Cheng Chang. Adaptive routing protocol for reliable sensor network applications. International Journal on Smart Sensing and Intelligent Systems, 2(4):515, 2009.

[12] Chris Karlof, Naveen Sastry, and David Wagner. Tinysec: a link layer security architecture for wire- less sensor networks. In Proceedings of the 2nd international conference on Embedded networked sensor systems, pages 162–175. ACM, 2004.

[13] Hugo Krawczyk. Hmqv: A high-performance secure Diffie-Hellman protocol. In Advances in Cryptology–CRYPTO 2005, pages 546–566. Springer, 2005.

[14] Philip Levis, Sam Madden, David Gay, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler. The emergence of networking abstractions and techniques in tinyos. In Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1, NSDI’04, pages 1–1, Berkeley, CA, USA, 2004. USENIX Association.

[15] An Liu and Peng Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In Information Processing in Sensor Networks, 2008. IPSN’08. International Conference on, pages 245–256. IEEE, 2008.

[16] Zhe Liu, Erich Wenger, and Johann Großscha¨dl. Mote-ecc: Energy-scalable elliptic curve cryptography for wireless sensor networks. In Applied Cryptography and Network Security, pages 361–379. Springer, 2014.

[17] MTM-CM5000-MSP MAXFOR Technology INC. http://maxfor.co.kr/eng/en_sub5_1_1.html

[18] ARM mbed IoT Device Platform. https://www.mbed.com/en/, 2016.

[19] BV Offspark. Polarssl. https://polarssl. org/, last access, 2013.

[20] Leonardo B Oliveira, Diego F Aranha, Conrado PL Gouveˆa, Michael Scott, Danilo F Caˆmara, Julio Lo´pez, and Ricardo Dahab. Tinypbc: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. Computer Communications, 34(3):485–493, 2011.

[21] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level sensor network simulation with cooja. In Local Computer Networks, Proceedings 2006 31st IEEE Conference on, pages 641–648. IEEE, 2006.

[23] Piotr Szczechowiak, Leonardo B Oliveira, Michael Scott, Martin Collier, and Ricardo Dahab. Nanoecc: Testing the limits of elliptic curve cryptography in sensor networks. In Wireless sensor networks, pages 305–320. Springer, 2008.

[24] Tony Tam, Mohamed Alfasi, and Mohammad Mozumdar. Securing resource constraints embedded devices using elliptic curve cryptography. In SPIE Defense+ Security, pages 90850N–90850N. International Society for Optics and Photonics, 2014.

[25] Chiu Chiang Tan, Haodong Wang, Sheng Zhong, and Qun Li. Ibe-lite: a lightweight identity-based cryptography for body sensor networks. Information Technology in Biomedicine, IEEE Transactions on, 13(6):926–932, 2009.

[26] Crossbow Technology. http://www.xbow.com.

[27] Chu-Sing Yang Yueh-Min Huang Tien-Wen Sung, Ting-Ting Wi. Reliable data broadcast for zig- bee wireless sensor networks. International Journal On Smart Sensing and Intelligent Systems, 3(3):504, 2010.

[28] Xiaokang Xiong, Duncan S Wong, and Xiaotie Deng. Tinypairing: a fast and lightweight pairing- based cryptographic library for wireless sensor networks. In Wireless Communications and Net- working Conference (WCNC), pages 1–6. IEEE, 2010.

[29] Geng Yang, Jiang-Tao Wang, Hong-Bing Cheng, and Chun-Ming Rong. A key establish scheme for wsn based on ibe and diffie-hellman algorithms. Dianzi Xuebao(Acta Electronica Sinica), 35(1):180–184, 2007.

[30] Tobias Zillner. Zigbee exploited: The good, the bad and the ugly