A Secure Communication Protocol for Drones and Smart Objects

Jongho Won
Department of Computer Science
Purdue University, IN, USA
won12@purdue.edu

Seung-Hyun Seo
Department of Computer Science
Purdue University, IN, USA
seosh77@gmail.com

Elisa Bertino
Cyber Center
Purdue University, IN, USA
bertino@purdue.edu

ABSTRACT

In many envisioned drone-based applications, drones will communicate with many different smart objects, such as sensors and embedded devices. Securing such communications requires an effective and efficient encryption key establishment protocol. However, the design of such a protocol must take into account constrained resources of smart objects and the mobility of drones. In this paper, a secure communication protocol between drones and smart objects is presented. To support the required security functions, such as authenticated key agreement, non-repudiation, and user revocation, we propose an efficient Certificateless Signcryption Tag Key Encapsulation Mechanism (eCLSC-TKEM). eCLSC-TKEM reduces the time required to establish a shared key between a drone and a smart object by minimizing the computational overhead at the smart object. Also, our protocol improves drone's efficiency by utilizing dual channels which allows many smart objects to concurrently execute eCLSC-TKEM. We evaluate our protocol on commercially available devices, namely AR.Drone2.0 and TelosB, by using a parking management testbed. Our experimental results show that our protocol is much more efficient than other protocols.

Categories and Subject Descriptors

C.2.2 [**Compute-Communication Networks**]: Network Protocols; E.3 [**Data Encryption**]: Public key cryptosystems

General Terms

Security, Design

Keywords

Certificateless Signcryption; Drone Communications

1. INTRODUCTION

Over recent years, drones are increasingly being used not only for military tasks, but also for civilian tasks, such as environment and traffic monitoring, delivery services, and aerial surveys. An Australian textbook rental startup company, Zookal, has already started using drones to deliver books. Amazon will soon start a new delivery method, Amazon Prime Air, which would offer 30-minute deliveries using drone-like octocopters. Also, some research projects [22, 29] have adopted drones as mobile collectors for monitoring applications based on wireless sensor networks (WSNs). For example, on-ground sensors can be deployed in farms to monitor the conditions of soil and drones can periodically collect information from these sensors and perform in-network processing of this information. In such context, drones can also be used for communicating with the on-ground sensors in order to send the sensors re-configuration instructions, such as instructing the sensors to change the sampling rates.

In many current and foreseen applications involving drones, including the Internet of Things (IoT), security is an important requirement. Drones, as many computing devices, are vulnerable to malicious attacks such as impersonation, manipulation and interception. Moreover, since drones may move around in unattended hostile areas with collected sensor data, they are vulnerable to physical capture. It is thus critical to address security requirements, such as authentication, non-repudiation, confidentiality and integrity, and securing communications between drones and other devices (referred to as smart objects in what follows), such as on-ground sensors. An important security building block is represented by cryptography which in turn requires a key management scheme. However, implementing a key management scheme suitable for WSNs that involve both smart objects and drones is quite challenging because of (1) the mobility and limited flight time of drones and (2) the constrained resources of smart objects.

Most encryption key management schemes proposed for WSNs adopt a symmetric-key-based approach instead of an asymmetric-key-based approach in order to address the limited energy and processing capability of sensors [6, 8]. However, the symmetric-key-based approach suffers from high communication overhead and requires large amounts of memory space to store the shared pairwise keys. Also such approach is not scalable, not resilient against compromises, and unable to support adequate node mobility. Public key cryptography (PKC) is relatively more expensive than symmetric key encryption in terms of computational costs, but recent improvements in the implementation of elliptic curve cryptography (ECC) [13] have demonstrated the practical applicability of PKC to WSNs. In order to enhance

ASIA CCS'15, April 14–17, 2015, Singapore..
Copyright © 2015 ACM 978-1-4503-3245-3/15/04 ...$15.00.
http://dx.doi.org/10.1145/2714576.2714616.

scalability and flexibility, asymmetric key based approaches that use ECC and identity-based PKC have been proposed for WSNs [15, 27, 7]. However, ECC-based schemes with certificates [27] and pairing operation-based ID-PKC [15, 7] schemes, when directly applied to WSNs, suffer from certificate management overhead and computational overhead from pairing operations, respectively. Moreover, since drones, unlike sensors, are likely to record a wide range of information, they can become a target for physical capture. We thus need an approach to minimize information leakage in the event that a drone is captured by attackers.

To address security and efficiency requirements for communications between drones and smart objects, we present the efficient Certificate-less Signcryption Tag Key Encapsulation Mechanism (eCLSC-TKEM) which supports authenticated key agreement and non-repudiation. The key feature of eCLSC-TKEM is to combine one-way key agreement and digital signatures into one efficient algorithm. Since eCLSC-TKEM is based on certificateless PKC (CL-PKC), it inherits the advantages of CL-PKC such as the elimination of the overhead resulting from the certificate management and the key escrow problem. eCLSC-TKEM supports user revocation by adopting Boneh et al.'s revocation technique [5] which adds a valid time period to the partial private keys that are issued. After the time period expires, new private keys are generated. Therefore, if a drone is captured, information leakage is limited to the time period during which the private keys were valid. To improve efficiency, eCLSC-TKEM minimizes the computational overhead at the smart object. The entire system efficiency highly depends on the computation time required by the smart object rather than the drone since a smart object is equipped with a low speed processor, while the drone has a PC-like processor. The contributions of our paper are summarized as follows.

- An efficient Certificateless Signcryption Tag Key Encapsulation Mechanism (eCLSC-TKEM) is proposed. The formal security model and a security proof are provided. Based on eCLSC-TKEM, a secure communication protocol for drone applications is presented. Our protocol is the most energy-efficient protocol supporting sharing of symmetric encryption keys for secure communications and non-repudiation.
- The *dual channel strategy* is introduced to concurrently perform eCLSC-TKEM with many smart objects at the same time, and thus to save the drone's energy.
- A secure communication protocol for the real drone application, i.e., smart parking management, has been implemented. The performance of eCLSC-TKEM has been evaluated in a testbed consisting of the commercially available devices AR.Drone2.0 and TelosB.

The remainder of this paper is organized as follows: In Section 2, we briefly discuss related work and requirements for drone-related communication protocols. In Section 3, we introduce our eCLSC-TKEM and dual channel strategy. In Section 4, we describe the design of our protocol through an example application. Then, the performance of our protocol is evaluated in Section 5. In Section 6, the security model and proof of eCLSC-TKEM are presented. Then, we outline conclusions and future works in Section 7.

2. BACKGROUND

2.1 Mobile data collectors in WSN

Several approaches [22, 25, 4, 29, 23, 18] have been proposed for using mobile data collectors in WSNs because of their advantages over traditional static WSNs such as connectivity, cost, reliability and energy efficiency. However, the use of mobile collectors introduces new security risks because mobile collectors are privileged nodes storing collected data and are exposed to physical capture. Zhou et al. [29] analyzed the security impact of mobile collector compromises and proposed a key pre-distribution scheme for group-based sensor networks. Song et al. [23] proposed a privilege-dependent pairwise key establishment scheme. This scheme revokes the privileges of a compromised mobile collector immediately after its compromise is detected. Rasheed et al. [18] proposed a secure data collection mechanism based on hash chains under the assumption that a mobile collector moves along a predetermined path. In this protocol, once a mobile collector is authenticated, a cluster head transfers aggregated data to the mobile collector. Although these protocols deal with the compromise of mobile collectors, their scalability is limited since they are based on symmetric key pre-distribution. Our protocol addresses the scalability problem by taking advantage of an asymmetric-key approach, while minimizing the computational overhead at the sensors.

To save the energy of sensor nodes or to extend the contact time between sensor nodes and mobile collectors, previous approaches [19, 28] adopted multiple radios/channels. However, these approaches did not consider the impact of slow public key operations at resource-constraint sensor nodes on mobile collectors.

2.2 CLSC-TKEM and CL-AKA

The authenticated key agreement (AKA) scheme is one of the most fundamental cryptographic mechanisms. It supports user authentication and generates shared secret keys between two parties over an insecure network. Traditional PKC-based AKA has the overhead of certificate management, whereas ID-PKC based AKA has the key escrow problem. To solve those issues, Al-Riyami et al. introduced certificateless public key cryptography (CL-PKC) [3]. Then, several certificateless authenticated key agreement (CL-AKA) protocols were built based on bilinear pairings. However, since the computational costs required for pairing operations are much higher than those for standard operations, such as EC point multiplication, it is hard to implement pairing-based applications on resource-constrained devices. Several pairing-free CL-AKA protocols [10, 9, 26, 24] have thus been proposed. However, most of those protocols were proved to be insecure and only two of them still remain secure: Sun's CL-AKA [24] and Yang's CL-AKA [26]. Recently, Li et al. [12] proposed a certificateless signcryption tag KEM (CLSC-TKEM) protocol. CLSC-TKEM supports not only practical authenticated key agreement but also designated verifier signature. Later, Selvi et al. [20] showed a security weakness in Li et al.'s CLSC-TKEM and presented an improved CLSC-TKEM. Since both CLSC-TKEM protocols [12, 20] rely on bilinear pairing operations, they are not suitable for resource-constrained devices. Recently, Seo et al. [21] proposed a pairing-free CLSC-TKEM protocol that does not use bilinear pairing operations. However, none of the existing CL-AKA and CLSC-TKEM protocols addresses

Table 1: Comparison of protocols

Protocol	Computational overhead on a smart object (on-line)	Security functionality			
		Key agreement	User authentication	Non-repudiation	User revocation
Yang's CL-AKA [26]	9EM + 1V (8EM + 1V)	yes	yes	no	no
Sun's CL-AKA [24]	6EM (5EM)	yes	yes	no	no
CLSC-TKEM [21]	5EM (3EM)	yes	yes	yes	no
eCLSC-TKEM	4EM (2EM)	yes	yes	yes	yes

EM: EC point multiplication, V: signature verification. 'On-line' means the computational overhead except ephemeral public key generations such as U and V generation in our protocol. The 'On-line' overhead is more meaningful than the entire overhead since ephemeral public keys can be generated in advance before a key agreement protocol starts.

user revocation which means that if drones are captured, the attacker will have full access not only to the information already collected and recorded in the drone, but also to future information to be collected by the drone.

In order to minimize information leakage in case of physical capture of drones, eCLSC-KTEM utilizes Boneh et al.'s revocation technique [5]. The key generation center (KGC) in eCLSC-TKEM adds time constraint to its partial private keys. In other words, partial private keys issued in eCLSC-TKEM are only valid for specified time periods. After the time period ends, new private keys will be generated. By adding this time constraint, we limit information leakage. To revoke a compromised drone, the KGC stops issuing partial private/public keys, including a valid time component, to the drone. Our approach prevents unauthorized users from being able to generate full private/public keys for future time periods. Although eCLSC-TKEM does not completely eliminate the risk of information leakage in case of physical capture, it limits the amount of compromised information to the information acquired during the last time period right before the revocation took place. Table 1 summarizes the comparison between eCLSC-TKEM and existing pairing-free CL-AKA and CLSC-TKEM.

2.3 Requirements for secure drone communications

The requirements for a communication protocol between drones and smart objects are summarized in terms of security and efficiency as follows.

- **Security** First, authenticated key establishment should be supported for confidentiality and user authentication. Second, it should be possible to verify the integrity of data and support non-repudiation. Third, information leakage should be minimized in the event that a drone is captured and its secret key is exposed. Finally, different access rights to smart objects should be supported for drones. Only authenticated/authorized drones should have access to smart objects' internal data according to their rights.
- **Efficiency** Since drones and smart objects are battery-powered, energy efficiency as well as security is a critical issue. Therefore, protocol executions should be completed as soon as possible to save energy. A drone usually has PC-like processing speed while a smart object has sensor-like processing speed. It is thus critical that protocols impose minimal computational overhead on smart objects. In addition, a drone may communicate with many smart objects at the same time. Therefore, the protocol should support concurrent cryptographic operations.

3. BUILDING BLOCKS

In this section, eCLSC-TKEM and the dual channel strategy are presented as major building blocks for our secure drone communication protocol.

3.1 eCLSC-TKEM

Unlike existing protocols, eCLCS-TKEM satisfies all the security requirements such as authenticated key agreement, non-repudiation and user revocation with the minimum computational overhead at smart objects (see Table 1). Note that the CL-AKA protocols [26, 24] satisfy only the first requirement. To support non-repudiation, the CL-AKA protocols must be extended with a signature scheme. Although CLSC-TKEM satisfies the first and second requirements, it does not support user revocation or multiple access rights. eCLSC-TKEM is a 8-tuple: (SetUp, SetSecretValue, PartialPrivateKeyExtract, SetPrivateKey, SetPublicKey, SymmetricKeyGen, Encapsulation Decapsulation). The description of each probabilistic polynomial time algorithm is as follows.

1) SetUp: This algorithm is run by the KGC to generate the system parameters `params` and a master secret key `msk`, given a security parameter k as input. KGC takes a security parameter $k \in \mathbb{Z}^+$ as input, and returns two system parameters: Ω and the KGC's master private key msk. Given k, the KGC performs the following steps:

- Chooses a k-bit prime q and determine the tuple $\{F_q, E/F_q, G_q, P\}$, where the point P is the generator of G_q.
- Chooses the master key $x \in \mathbb{Z}_q^*$ uniformly at random and computes the system public key $P_{pub} = xP$.
- Chooses cryptographic hash functions $H_0 : \{0,1\}^* \times G_q^2 \times \{0,1\}^* \rightarrow \mathbb{Z}_q^**$, $H_1 : G_q^3 \times \{0,1\}^* \times G_q \rightarrow \{0,1\}^n$, $H_2 : G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \rightarrow \mathbb{Z}_q^*$, and $H_3 : G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \rightarrow \mathbb{Z}_q^*$. Here, n is the key length of a DEM.
- Publishes $\Omega = \{F_q, E/F_q, G_q, P, P_{pub}, H_0, H_1, H_2, H_3\}$ as the system's parameter and keeps the master key x secret.

2) SetSecretValue: This algorithm is run by each entity to generate a secret value and the corresponding public value for oneself. The entity A with an identity ID_A chooses $x_A \in \mathbb{Z}_q^*$ uniformly at random as its secret value and generates the corresponding public key as $P_A = x_A P$.

3) PartialPrivateKeyExtract: The KGC runs this algorithm to generate the partial private key of a user. It takes the KGC's master secret key, the id of the user ID_A, the permitted time period t_A and the system parameter as inputs. It returns the partial private key of the entity. In order to obtain the partial private key, the entity A sends (ID_A, P_A) to the KGC. The KGC then executes the following steps:

- Chooses $r_A \in \mathbb{Z}_q^*$ and computes $R_A = r_A P$.
- Computes $d_A = r_A + xH_0(ID_A, R_A, P_A, t_A) \bmod q$.

The partial private key of the entity A is d_A. The entity can validate its private key by checking whether $d_A P = R_A + H_0(ID_A, R_A, P_A, t_A)P_{pub}$ holds.

4) SetPrivateKey: This algorithm is run by each entity to generate the full private key. The entity A takes the pair $sk_A = (d_A, x_A)$ as its full private key.

5) SetPublicKey: This algorithm is run by each entity to generate the full public key. The entity A takes the pair $pk_A = (P_A, R_A)$ as its full public key.

6) SymmetricKeyGen: This algorithm is run by the sender A to obtain the symmetric key K and an internal state information ω, which is not known to the receiver B. Given the sender (entity A)'s identity ID_A, the full public key pk_A, the full private key sk_A, the receiver (entity B)'s identity ID_B, the time interval t_B and the full public key pk_B as inputs, the sender executes this symmetric key generation algorithm to obtain the symmetric key K as follows:

- Chooses $l_A, s_A \in \mathbb{Z}_q^*$ and computes $U = l_A P$, $V = s_A P$.
- Computes $Y = R_B + H_0(ID_B, R_B, P_B, t_B) \cdot P_{pub} + P_B$, $T = s_A \cdot Y (= s_A \cdot (H_0(ID_B, R_B, P_B, t_B) \cdot P_{pub} + R_B + P_B))$ and $K = H_1(Y, V, T, ID_B, P_B)$.
- Outputs K and the intermediate information $\omega = (l_A, s_A, U, V, T, ID_A, pk_A, sk_A, ID_B, pk_B, t_B)$.

7) Encapsulation: This algorithm is executed by the sender A to obtain the encapsulation φ. It takes ω corresponding to K and an arbitrary tag τ as inputs. Given a state information ω and an arbitrary tag τ, the sender A obtains the encapsulation φ by performing the following steps:

- Computes $H = H_2(U, \tau, T, ID_A, P_A, ID_B, P_B)$, $H' = H_3(U, \tau, T, ID_A, P_A, ID_B, P_B)$ and $W = d_A + l_A \cdot H + x_A \cdot H'$
- Outputs $\varphi = (U, V, W)$.

8) Decapsulation: This algorithm is executed by the receiver B to obtain the key K encapsulated in φ. Given the encapsulation φ, a tag τ, the sender's identity ID_A, full public key pk_A, the time interval t_A the receiver's identity ID_B, the full public key pk_B and the full private key sk_B, the key K is computed as follows:

- Computes $Y = (d_B + x_B) \cdot P \; (= (r_B + xH_0(ID_B, R_B, P_B, t_B) + x_B) \cdot P = R_B + H_0(ID_B, R_B, P_B, t_B) \cdot P_{pub} + P_B)$, $T = (d_B + x_B) \cdot V (= (d_B + x_B) \cdot s_A P = s_A \cdot Y)$.
- Computes $H = H_2(U, \tau, T, ID_A, P_A, ID_B, P_B)$ and $H' = H_3(U, \tau, T, ID_A, P_A, ID_B, P_B)$.
- If $W \cdot P = R_A + H_0(ID_A, R_A, P_A, t_A) \cdot P_{pub} + H \cdot U + H' \cdot P_A$, outputs $K = H_1(Y, V, T, ID_B, P_B)$.
  Otherwise, outputs an invalid encapsulation error. The correctness of the above equation is as follows:
  $$\begin{aligned} W \cdot P &= (d_A + l_A \cdot H + x_A \cdot H') \cdot P \\ &= d_A \cdot P + l_A \cdot P \cdot H + x_A \cdot P \cdot H' \\ &= (r_A + xH_0(ID_A, R_A, P_A, t_A)) \cdot P + U \cdot H + H' \cdot P_A \\ &= R_A + H_0(ID_A, R_A, P_A, t_A) \cdot P_{pub} + H \cdot U + H' \cdot P_A \end{aligned}$$

3.2 Dual Channel Strategy for Concurrency using LPL

Since smart objects and drones are battery-powered, their energy should be efficiently used. Our protocol utilizes dual channels and low power listening (LPL) [1]. The asynchronous duty cycling technique, known as LPL, is one of most promising power-saving techniques for WSNs and results in higher performance than synchronous duty cycling techniques in terms of energy and throughput [17].

Each smart object has one radio and changes its channel according to its situations, while drones are equipped with two radios, i.e., the wake-up radio and the data radio. As depicted in Fig. 1, a smart object periodically turns a radio transceiver on (wake-up) and off (sleep) on the wake-up channel to save its energy. Through the wake-up radio, a mobile drone continuously broadcasts wake-up signals including its ID and public keys. When a smart object wakes up, it quickly checks whether the wake-up channel is busy. If the wake-up channel is idle, the smart object can save energy by sleeping again until the next wake-up time. If a mobile drone broadcasting the wake-up signals approaches some smart objects, the wake-up channel becomes busy. If the smart objects listen the wake-up signals, they stay awake and receive a whole wake-up signal. Then, each smart object concurrently initiates SymmetricKeyGen and Encapsulation and switches the channel from the wake-up channel to the data channel. The Encapsulation output is transmitted to the drones using the data channel. These concurrent executions of eCLSC-TKEM in smart objects can save drone's energy. If a drone has only one radio, it might require precise time synchronization with smart objects to make schedules or perform eCLSC-TKEM with each smart object one by one, which results in a waste of its limited flight time.

4. SECURE COMMUNICATION PROTOCOL FOR DRONES

In this section, we illustrate the design of our protocols in the context of a smart parking management application.

4.1 Smart parking management

Today's parking management is a manual, labor-intensive process. Parking enforcement officers must periodically patrol on-street parking areas and check cars one by one to identify cars that are parked over time . By utilizing drones and sensors, parking management can be more efficient and cost effective. Cars might attach an independent device equipped with a GPS and a radio transceiver. For example, an university can issue those independent devices to registered car owners as parking permits for parking management on its campus. Then, a drone would patrol time limited parking areas and collect information from every parked car. Information includes the identity and location of the car, the type of parking permit and the current time. By collecting this information at certain time intervals, drones can detect if a car is parked at an invalid zone or has been parked at the same spot for longer than the time limit.

In this scenario, since privacy-sensitive information is involved, only authorized drones should be allowed to collect this information. More importantly, since the information is used to fine drivers who parked their cars illegally, a protocol must support non-repudiation and integrity of the collected information.

4.2 Registration

We assume that every car is equipped with a smart object which includes a low-speed CPU, a small memory, a GPS sensor and a radio transceiver. The smart object (A) generates its own secret value (x_A) and corresponding public key (P_A) by executing the SetSecretValue algorithm. For each smart object, the KGC generates a partial private/public key pair (d_A, R_A) by executing the PartialPrivateKeyExtract algorithm and transfers it to the smart object through a se-

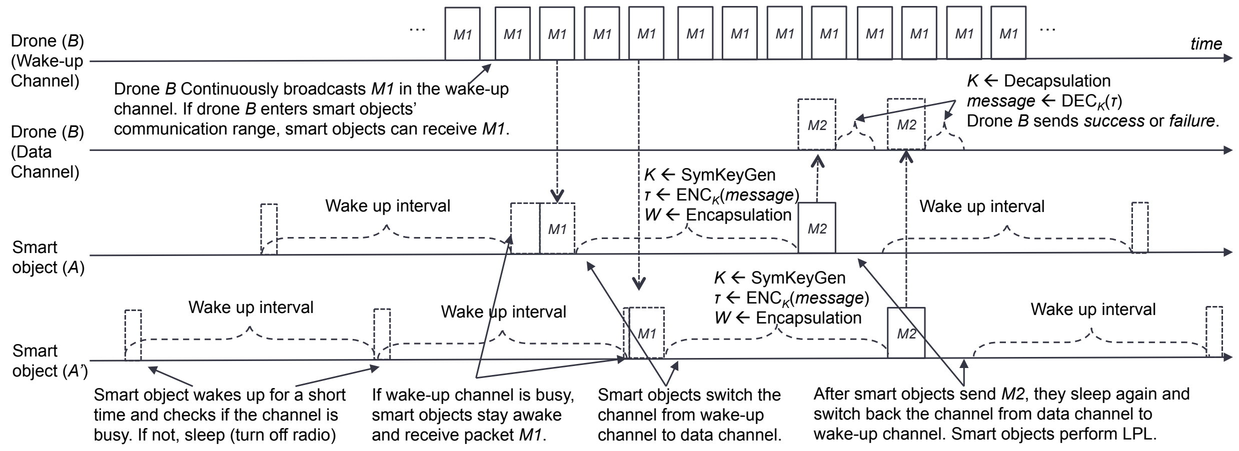
Drone (B) (Wake-up Channel)
Drone (B) (Data Channel)
Smart object (A)
Smart object (A')
time
M1
M2
Drone B Continuously broadcasts M1 in the wake-up channel. If drone B enters smart objects' communication range, smart objects can receive M1.
K ← Decapsulation
message ← DEC_K(τ)
Drone B sends success or failure.
K ← SymKeyGen
τ ← ENC_K(message)
W ← Encapsulation
Wake up interval
Smart object wakes up for a short time and checks if the channel is busy. If not, sleep (turn off radio)
If wake-up channel is busy, smart objects stay awake and receive packet M1.
Smart objects switch the channel from wake-up channel to data channel.
After smart objects send M2, they sleep again and switch back the channel from data channel to wake-up channel. Smart objects perform LPL.

Figure 1: Solid-line square: packet transmission, dash-line square: listen or packet reception. $M1 = \{ID_B, P_B, R_B, t_B\}$**,** $M2 = \{ID_A, P_A, R_A, t_A, U, V, W, \tau\}$**, Decapsulation result (success/failure) transmissions are omitted.**

cure channel. Note that the partial private key is only valid for a certain period of time, for example one year, since a permitted time period is included in t_A of the PartialPrivateKeyExtract algorithm. Thus, a car owner should periodically renew the partial private/public key. We assume that a drone stays in a secure place when it is off duty. The drone (B) generates its secret value (x_B) and corresponding public key (P_B) by executing the SetSecretValue algorithm. Before the drone goes out to patrol, it requests a partial private/public key (d_B, R_B) from the KGC. The partial private key is only valid for the maximum flight time, for example 30 minutes; this time period is inserted into t_B of PartialPrivateKeyExtract algorithm. Therefore, in the event that the drone is compromised, the information leakage is limited to this time period. Similarly, access rights granted to the drone are also inserted in t_B.

4.3 Key establishment using dual channels

Fig. 1 illustrates the flow of our protocol. A smart object (A) runs LPL in the wake-up channel and tries to detect wake-up signals from a drone. The drone (B) continuously broadcasts wake-up signals ($M1$) in the wake-up channel while moving, so that smart objects can detect $M1$ when they awake. $M1$ contains the drone's ID (ID_B), the public keys (P_B, R_B) and the permitted time period/access right (t_B). If a smart object wakes up and receives $M1$, it stops LPL and generates a symmetric key (K) by performing SymmetricKeyGen. Then, using K, the smart object generates a ciphertext (τ) by encrypting messages, such as permit type, location information (loc) and current time (ct). After the smart object generates W by executing the Encapsulation algorithm, it switches the channel from the wake-up channel to the data channel and sends $M2$ to B. Since τ is signed by A as part of the Encapsulation, A cannot repudiate τ afterward. $M2$ contains the ID of the smart object (ID_A), the public keys (P_A, R_A), the permitted time period (t_A), ephemeral public keys (U, V), the result value of Encapsulation (W), and the ciphertext (τ).

If the drone receives $M2$, it executes Decapsulation. If the validation process is successful, the drone (B) generates K and decrypts τ using K. Right after obtaining loc and ct from τ, B compares loc and ct with its own current location (loc') and current time (ct'), respectively. Then, based on the comparison outcome, it may perform additional actions. For example, if $|loc - loc'| > 10m$ or $ct - ct' > 5$ mins, B might take actions such as taking a picture of the car or sending a message to a human manager. Finally, B sends the decapsulation result to A. If the result is successful, both the drone (B) and the smart object (A) can use K to exchange encrypted messages.

5. EXPERIMENTS

To evaluate our protocol, we implemented not only our protocol, but also the CL-AKA [26, 24] and CLSC-TKEM [21] protocols on the commercially available devices AR.Drone2.0 [2] and TelosB sensors.

5.1 Experiment Setup

5.1.1 Drone

AR.Drone2.0 [2] is a quad-copter equipped with front and ground cameras and a Wi-Fi (2.4GHz). It has a 1GHz 32-bit ARM cortex A8 CPU and a 1Gbit DDR2 RAM. The main board of the AR.Drone2.0 runs the BusyBox based GNU/Linux distribution with 2.6.32 kernel. After the drone is booted, it works as a Wi-Fi access point. The drone can be controlled from smartphones or laptops by sending UDP control commands through the Wi-Fi. As ECC library, we adopted the micro-ecc [14] optimized for the ARM processor. To compile our code, we used a 32-bit Linux machine since our code had to be cross-compiled for the 32-bit ARM architecture. Then, we transferred the compiled code to AR.Drone2.0 using FTP and executed it by Telnet.

For the wake-up radio and the data radio, two TelosBs were connected to the drone using USB-to-serial interfaces (see Fig. 2). The data radio is connected to an external USB-to-serial interface which is located nearby the battery connector. The wake-up radio is connected to a pin connector on the main board. TelosB is equipped with a CC2420 IEEE 802.15.4 radio transceiver which works at the 2.4GHz ISM band. To avoid interferences between the Wi-Fi and the IEEE 802.15.4 radios, the channel 6 was selected as the Wi-Fi channel and the channel 11 and 26 were selected as the wake-up channel and the data channel, respectively.

5.1.2 Smart object

We used a TelosB equipped with an extremely low speed CPU (TIMSP430 F1161) as a smart object in order to show that our protocol works well even with low speed devices.

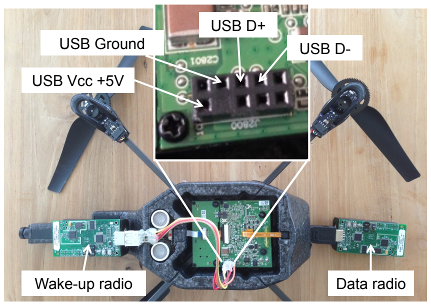

Figure 2: Dual radios attached on AR.Drone2.0

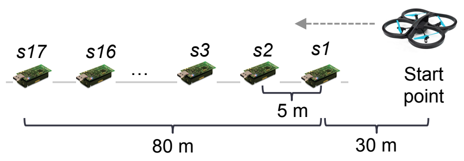

Figure 3: Network topology

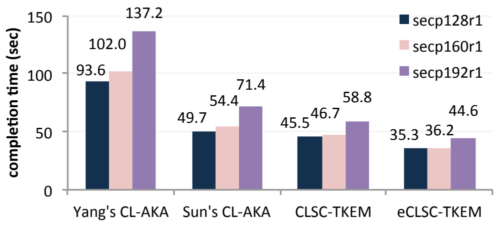

Figure 4: Impact of key bit size

Although the maximum communication range of CC2420 is approximately 100m when the RF power is set to 0dBm, we set the RF power to -7dBm to save power and its communication range became roughly 30m. TelosB is operated by TinyOS 2.0 which is an open-source operating system designed for low-power devices. We used the LPL functionalities in TinyOS 2.0. In addition, TinyECC [13] was used as the basic ECC operations of our protocol.

5.1.3 *Network topology*

Fig. 3 illustrates network topology which mimics the scenario of the smart parking management. 17 smart objects are deployed at 5m intervals in a line and a drone starts from the starting point which is 30m apart from $s1$ at an altitude of 10m. The mission of the drone is to collect messages from all smart objects. and the mission completion time of the drone was measured. The drone proceeds from the start point to $s17$. If the drone reaches a smart object (sx), but is unable to finish the data collection task with sx, the drone waits until the collection is completed.

5.2 Experimental results

5.2.1 *Impact of key bit size*

Fig. 4 shows the mission completion time of four protocols with different ECC key bit sizes when the system adopts 5sec wake-up interval and dual channels. When secp160r1 is used, the mission completion time of our protocol is 36.2sec which is 1.3, 1.5 and 2.8 times faster than Seo's CLSC-TKEM, Sun's CL-AKA and Yang's CL-AKA, respectively. The completion time of each protocol increases as the key bit size increases. However, the difference between a 128-bit key and a 160-bit key is much smaller than the difference between a 160-bit key and a 192-bit key, which implies

Table 2: Comparison of the on-line computation time of a smart object (unit: second)

Protocol	secp128r1	secp160r1	secp192r1
Yang's CL-AKA [26]	32.84	36.22	50.43
Sun's CL-AKA [24]	15.10	16.98	23.84
CLSC-TKEM [21]	13.37	13.87	18.77
eCLSC-TKEM	9.25	9.61	13.03

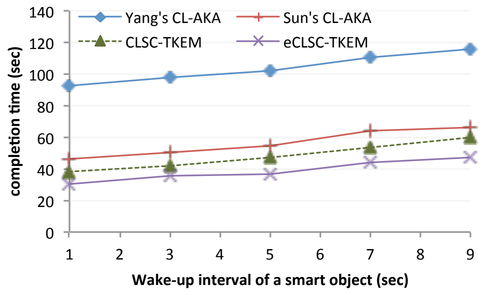

Figure 5: Impact of interval between wake-ups

that a 160-bit key may be a reasonable choice since it provides better security than a 128-bit key with a very small time increase. These results are confirmed by Table 2 which compares the computation time measurements of each protocol on a smart object with different ECC curves. The computation time at a smart object in our protocol is 1.4, 1.8 and 3.8 times faster than Seo's CLSC-TKEM, Sun's CL-AKA and Yang's CL-AKA, respectively, when secp160r1 is used. Note that in our protocol the smart object is required to compute only two EC point multiplications after it starts communicating with the drone (online), while the other protocols require more than two EC point multiplications (see Table 1). These results are similar to the completion time results since the overall performance highly depends on the computation time at the smart object.

5.2.2 *Impact of interval between wake-ups*

We analyzed the impact of the wake-up interval of smart objects when the system adopts secp160r1 and dual channels. As shown in Fig. 5, our protocol, compared to CLSC-TKEM and Sun's CL-AKA, has a similar completion time with longer intervals between wake-ups, thus saving energy in smart objects. Specifically, the mission completion time of our protocol is 46.8sec when the wake-up interval is 9sec. However, in CLSC-TKEM, the wake-up interval that achieves a similar mission completion time, i.e., 46.7sec, is 5sec. In Sun's protocol, the wake-up interval that achieves the similar mission completion time, i.e., 46.2sec, is 1sec. As a result, by using our protocol a smart object can save energy 1.8 and 9 times more than when using the CLSC-TKEM and Sun's CL-AKA, respectively, if the mission completion times of three protocols are equally set.

5.2.3 *Impact of drone's altitude*

Since our protocol is efficient, a drone using our protocol does not require a long contact time with smart objects and thus complete the mission faster than others even at a higher altitude. For example, based on experiments we have carried out, the time required by eCLSC-TKEM to complete a mission at an altitude of 20m was 47sec, whereas the time required by CLSC-TKEM at an altitude of 5m was 49sec. [1]

[1]secp160r1 and 5sec wake-up interval are used. These detailed results are not included in the paper for lack of space.

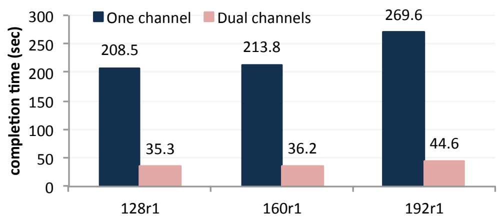

Figure 6: Impact of the dual channel strategy

5.2.4 *Impact of dual channel strategy*

Finally, Fig. 6 shows the impact of the dual channel strategy on the system when the wake-up interval is 5sec. For the key establishment scheme, eCLSC-TKEM is utilized. The drone completes the mission approximately 6 times faster than when only one channel is used. If the dual channel strategy is adopted, the drone can take advantage of independent dual channels. The drone can continuously broadcast the wake-up signals on the wake-up channel while, at the same time, it can exchange eCLSC-TKEM output with smart objects on the data channel. The experiment results shown in Fig. 6 refer to the case in which 3 or 4 smart objects are in the communication range of the drone (see also Fig.5). The results show that all the smart objects in the range are able to concurrently initiate the execution of the eCLSC-TKEM protocol once they receive the wakeup signal. However, if only one channel is used, the drone must perform eCLSC-TKEM with smart objects one by one. In the one-channel system, the drone broadcasts the wake-up signals. Once a smart object receives the wake-up signal, it sends an acknowledge to the drone. Then, the drone must stop broadcasting the wake-up signals to receive the eCLSC-TKEM output from the smart object. If the whole process of eCLSC-TKEM with the smart object is successfully finished, the drone starts broadcasting the wake-up signals again to wake up another smart object. Since each smart object takes much longer time for the execution of the eCLSC-TKEM protocol than the drone, the drone must wait wasting its limited flight time doing nothing. As a result, the dual channel strategy is essential for efficient and secure communication protocols that support many different security functions.

6. SECURITY ANALYSIS

6.1 Security Model of eCLSC-TKEM

An efficient certificateless signcryption tag KEM must consider three types of adversaries: $\mathcal{A}_{\mathcal{I}}$, $\mathcal{A}_{\mathcal{II}}$ and $\mathcal{A}_{\mathcal{III}}$. $\mathcal{A}_{\mathcal{I}}$ represents a dishonest user who can replace other user's public keys but has no knowledge about the master secret key of the KGC. $\mathcal{A}_{\mathcal{II}}$ represents a malicious KGC which has knowledge of the KGC's master secret key. However, $\mathcal{A}_{\mathcal{II}}$ is unable to replace the users' public keys. $\mathcal{A}_{\mathcal{III}}$ represents a previously functional user, whose partial private/public keys have been revoked by the KGC. $\mathcal{A}_{\mathcal{III}}$ cannot replace other users' public keys. Except for the consideration of $\mathcal{A}_{\mathcal{III}}$, the security model of eCLSC-TKEM is similar to that of CLSC-TKEM [20, 12]. eCLSC-TKEM must satisfy confidentiality, that is, indistinguishability against an adaptive chosen ciphertext and identity attacks (IND-CCA2), and unforgeability, that is, existential unforgeability against adaptive chosen messages and identity attacks (EUF-CMA). In order to describe the security model of eCLSC-TKEM, we consider the two formal games IND-eCLSC-TKEM-CCA2 game and EUF-eCLSC-TKEM-CMA game.

1) IND-eCLSC-TKEM-CCA2 Game: The adversary $\mathcal{A}$ can be either $\mathcal{A}_{\mathcal{I}}$, $\mathcal{A}_{\mathcal{II}}$ or $\mathcal{A}_{\mathcal{III}}$. The challenger $\mathcal{C}$ should keep a history of query-answers while interacting with adversaries. $\mathcal{C}$ runs the $\mathsf{SetUp}()$ algorithm to generate the public parameters $\mathtt{params}$ and the master private key $\mathtt{msk}$ respectively. If $\mathcal{A}$ is either $\mathcal{A}_{\mathcal{I}}$ or $\mathcal{A}_{\mathcal{III}}$, $\mathcal{C}$ gives $\mathtt{params}$ to $\mathcal{A}$ while keeping $\mathtt{msk}$ secret. If $\mathcal{A}$ is $\mathcal{A}_{\mathcal{II}}$, $\mathcal{C}$ gives both $\mathtt{params}$ and $\mathtt{msk}$ to $\mathcal{A}$.

Phase I: $\mathcal{A}$ may perform a polynomially bounded number of the following queries in an adaptive fashion.

- **Extract-Secret-Value queries**: $\mathcal{C}$ runs $\mathsf{SetSecretValue}$ to get x_U with identity ID_U, and then returns it to $\mathcal{A}_{\mathcal{I}}$. In the case of $\mathcal{A}_{\mathcal{III}}$, $\mathcal{C}$ runs $\mathsf{SetSecretValue}$ before the challenge time period and returns x_U to $\mathcal{A}_{\mathcal{III}}$. The adversary $\mathcal{A}_{\mathcal{I}}$ or $\mathcal{A}_{\mathcal{III}}$ cannot query any identity for which the corresponding public key has been replaced. $\mathcal{A}_{\mathcal{II}}$ is excluded in this query.
- **Extract-Partial-Private-Key queries**: In the case of $\mathcal{A} \in \{\mathcal{A}_{\mathcal{I}}, \mathcal{A}_{\mathcal{II}}\}$, these can be made for all identities except for the target identity. If $\mathcal{A}$ is $\mathcal{A}_{\mathcal{III}}$, these can be made for any identity before the challenge time period. $\mathcal{C}$ runs $\mathsf{PartialPrivateKeyExtract}$ to obtain the partial private key d_U and the permitted time period t_U. Then $\mathcal{C}$ sends d_U and t_U to $\mathcal{A}$.
- **Request-Public-Key queries**: In the case of $\mathcal{A} \in \{\mathcal{A}_{\mathcal{I}}, \mathcal{A}_{\mathcal{II}}\}$, $\mathcal{C}$ runs $\mathsf{SetPublicKey}$ to get the full public key pk_U and then returns it to $\mathcal{A}_{\mathcal{I}}$. If $\mathcal{A}$ is $\mathcal{A}_{\mathcal{III}}$, $\mathcal{C}$ runs $\mathsf{SetPublicKey}$ to get the full public key pk_U and returns it to $\mathcal{A}_{\mathcal{III}}$ before the challenge time period.
- **Public-Key-Replacement queries**: $\mathcal{A}_{\mathcal{I}}$ may replace the public key pk_U corresponding to the user identity ID_U with any value pk'_U of $\mathcal{A}_{\mathcal{I}}$'s choice. $\mathcal{A}_{\mathcal{II}}$ and $\mathcal{A}_{\mathcal{III}}$ are excluded in this query.
- **Symmetric Key Generation queries**: In the case of $\mathcal{A} \in \{\mathcal{A}_{\mathcal{I}}, \mathcal{A}_{\mathcal{II}}\}$, $\mathcal{A}$ chooses a sender's identity ID_A and a receiver's identity ID_B. $\mathcal{C}$ obtains the private key of the sender, sk_A and t_B from the corresponding "query-answer" list. Then, $\mathcal{C}$ runs $\mathsf{SymmetricKeyGen}$ to obtain the symmetric key K and an internal state information ω by using ID_A, ID_B, sk_A, pk_B and t_B. It stores ω while keeping the ω secret from the view of $\mathcal{A}$. Finally, $\mathcal{C}$ sends K to $\mathcal{A}$. $\mathcal{C}$ may not obtain the sender's secret value if the associated public value of the sender A is replaced. In this case, $\mathcal{A}$ is required to provide the secret value of A to $\mathcal{C}$. We do not allow queries where $ID_A = ID_B$. If $\mathcal{A}$ is $\mathcal{A}_{\mathcal{III}}$, $\mathcal{C}$ runs the above operations for any time instant before the challenge time period.
- **Key Encapsulation queries**: In the case of $\mathcal{A} \in \{\mathcal{A}_{\mathcal{I}}, \mathcal{A}_{\mathcal{II}}\}$, $\mathcal{A}$ produces an arbitrary tag τ for sender A. $\mathcal{C}$ checks whether there exists a corresponding ω value. If ω has been previously stored, then $\mathcal{C}$ computes $(\varphi) \leftarrow \mathsf{Encapsulation}(\omega, \tau)$, deletes ω and returns φ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ returns $\perp$ and terminates. In case that $\mathcal{A}$ is $\mathcal{A}_{\mathcal{III}}$, $\mathcal{C}$ runs the above operations for any time instant before the challenge time period.
- **Key Decapsulation queries**: In the case of $\mathcal{A} \in \{\mathcal{A}_{\mathcal{I}}, \mathcal{A}_{\mathcal{II}}\}$, $\mathcal{A}$ produces an encapsulation φ, a tag τ, the sender's identity ID_A, the public key pk_A, the receiver's identity ID_B and the public key pk_B. $\mathcal{C}$ obtains the receiver's private key sk_B and t_A from the corresponding "query-answer" list. $\mathcal{C}$ runs $\mathsf{Decapsulation}$ by using ID_A, ID_B, pk_A, sk_B, t_A, φ and τ. $\mathcal{C}$ may not be aware of the corresponding secret value if the associated public value of

ID_B is replaced. In this case $\mathcal{A}$ must provide the secret value of B to $\mathcal{C}$. We do not allow the queries where $ID_A = ID_B$. If $\mathcal{A}$ is $\mathcal{A}_{\mathcal{III}}$, $\mathcal{C}$ runs the above operations for any time instant before the challenge time period.

Challenge: At the end of Phase I decided by $\mathcal{A} \in \{\mathcal{A}_{\mathcal{I}}, \mathcal{A}_{\mathcal{II}}\}$, $\mathcal{A}$ generates a sender identity ID_{A^*} and a receiver identity ID_{B^*} on which $\mathcal{A}$ wishes to be challenged. Here, ID_{B^*} must not be queried to extract a sk_{B^*} in Phase I. Also, in case that $\mathcal{A}$ is $\mathcal{A}_{\mathcal{I}}$, ID_{B^*} may not be equal to an identity for which both the public key has been replaced and the partial private key has been extracted. At the end of Phase I which is decided by $\mathcal{A} \in \{\mathcal{A}_{\mathcal{III}}\}$, $\mathcal{A}_{\mathcal{III}}$ generates a sender identity ID_{A^*} and a receiver identity ID_{B^*} on which $\mathcal{A}_{\mathcal{III}}$ wishes to be challenged for some instant t'_{A^*} such that $t'_{A^*} > t_{A^*}$ (after he has been revoked). In the revoked period, $\mathcal{A}_{\mathcal{III}}$ has access to no new information. Now, $\mathcal{C}$ computes $(K_1, \omega^*) \leftarrow \mathsf{SymmetricKeyGen}(\mathtt{params}, ID_{A^*}, pk_{A^*}, sk_{A^*}, ID_{B^*}, pk_{B^*}, t_{B^*})$ and chooses $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the eCLSC-TKEM. The $\mathcal{C}$ chooses a bit $\delta \in_R \{0, 1\}$ and sends K_δ to $\mathcal{A}$. $\mathcal{A}$ generates an arbitrary tag τ^* and sends it to $\mathcal{C}$. $\mathcal{C}$ computes $(\varphi^*) \leftarrow \mathsf{Encapsulation}(\omega^*, \tau^*)$ and sends φ^* to $\mathcal{A}$ as a challenge encapsulation.

Phase II: In the case that $\mathcal{A}$ is $\mathcal{A}_{\mathcal{I}}$ or $\mathcal{A}_{\mathcal{II}}$, $\mathcal{A}$ can perform a polynomially bounded number of queries adaptively as in Phase I. However, when $\mathcal{A}$ is $\mathcal{A}_{\mathcal{III}}$, the notable difference is that $\mathcal{A}$ can perform a polynomially bounded number of queries adaptively, before the beginning of the challenge period as in Phase I. $\mathcal{A}$ may not make Extract-full-Private-Key queries on ID_{B^*}. In $\mathcal{A}_{\mathcal{I}}$, if the public key of ID_{B^*} has been replaced before the challenge phase, $\mathcal{A}_{\mathcal{I}}$ may not extract the partial private key for ID_{B^*}. Moreover, $\mathcal{A}$ may not make a key decapsulation query on (K_δ, φ^*) under ID_{A^*} and ID_{B^*}, unless the public key $pk_{ID_{A^*}}$ or $pk_{ID_{B^*}}$ has been replaced after the challenge phase.
Guess: $\mathcal{A}$ outputs a bit δ' and wins the game if $\delta' = \delta$.

The advantage of $\mathcal{A}$ is defined as $Adv^{IND-CCA2}(\mathcal{A}) = |2Pr[\delta' = \delta] - 1|$, where $Pr[\delta' = \delta]$ denotes the probability that $\delta' = \delta$. A eCLSC-TKEM is IND-CCA2 secure if there is no probabilistic polynomial-time adversary in the above games with non-negligible advantage in the security parameter k. The security of eCLSC-TKEM is based on the assumed intractability of the one-sided gap Diffie-Hellman problem (OGDH) [11].

2) EUF-eCLSC-TKEM-CMA Game: The Forger $\mathcal{F}$ can be either $\mathcal{F}_{\mathcal{I}}$, $\mathcal{F}_{\mathcal{II}}$ or $\mathcal{F}_{\mathcal{III}}$. The challenger $\mathcal{C}$ should keep a history of the query-answers while interacting with adversaries. $\mathcal{C}$ runs the $\mathsf{SetUp}()$ algorithm to generate the public parameters `params` and the master private key `msk` respectively. If $\mathcal{F}$ is either $\mathcal{F}_{\mathcal{I}}$ or $\mathcal{F}_{\mathcal{III}}$, $\mathcal{C}$ gives `params` to $\mathcal{F}$ while keeping `msk` secret. If $\mathcal{F}$ is $\mathcal{F}_{\mathcal{II}}$, $\mathcal{C}$ gives both `params` and `msk` to $\mathcal{F}$.

Training Phase: $\mathcal{F}$ may make a polynomially bounded number of queries to random oracles $H_i (0 \leq i \leq 3)$ at any time and $\mathcal{C}$ responds as follows:
All the oracles and queries needed in the training phase are identical to the queries allowed in Phase I of the IND-eCLSC-TKEM-CCA2 game.

Forgery: At the end of the Training Phase which is decided by $\mathcal{F} \in \{\mathcal{F}_{\mathcal{I}}, \mathcal{F}_{\mathcal{II}}\}$, $\mathcal{F}$ produces an encapsulation $\langle \tau^*, \varphi^*, ID_{A^*}, ID_{B^*} \rangle$ on a arbitrary tag τ^*, where ID_{A^*} is the sender identity and ID_{B^*} is the receiver identity. At the end of the Training Phase decided by $\mathcal{F} = \mathcal{F}_{\mathcal{III}}$, $\mathcal{F}_{\mathcal{III}}$ generates a sender identity ID_{A^*} and a receiver identity ID_{B^*} on which $\mathcal{F}_{\mathcal{III}}$ wishes to be challenged for some instant t'_{A^*} such that $t'_{A^*} > t_{A^*}$ (after $\mathcal{F}_{\mathcal{III}}$ has been revoked). Then, $\mathcal{F}$ sends $\langle \tau^*, \varphi^*, ID_{A^*}, ID_{B^*} \rangle$ to $\mathcal{C}$. If $\mathcal{F}$ is $\mathcal{F}_{\mathcal{I}}$, during the Training Phase, the partial private key for ID_{A^*} must not be queried and the public key for ID_{A^*} must not be replaced simultaneously. If $\mathcal{F}$ is $\mathcal{F}_{\mathcal{II}}$, the secret value x_{A^*} for ID_{A^*} must not be queried and the public key for ID_{A^*} must not be replaced, simultaneously. Moreover φ^* must not be returned by the key encapsulation oracle on the input $(\tau^*, \omega^*, ID_{A^*}, ID_{B^*})$ during the Training Phase. If the output of $\mathsf{Decapsulation}(\mathtt{params}, ID_{A^*}, pk_{A^*}, t_{A^*}, ID_{B^*}, pk_{B^*}, sk_{B^*}, \varphi^*, \tau^*)$ is valid, $\mathcal{F} \in \{\mathcal{F}_{\mathcal{I}}, \mathcal{F}_{\mathcal{II}}\}$ wins the game. If the output of $\mathsf{Decapsulation}(\mathtt{params}, ID_{A^*}, pk_{A^*}, t'_{A^*}, ID_{B^*}, pk_{B^*}, sk_{B^*}, \varphi^*, \tau^*)$ is valid, $\mathcal{F} = \mathcal{F}_{\mathcal{III}}$ wins the game.

The advantage of $\mathcal{F}$ is defined as the probability with which it wins the EUF-pCLSC-TKEM-CMA game. A eCLSC-TKEM satisfies existential unforgeability against an adaptively chosen message attack (EUF-eCLSC-TKEM-CMA), if no polynomially bounded forger $\mathcal{F}$ has non-negligible advantage in the above EUF-eCLSC-TKEM-CMA game between $\mathcal{C}$ and $\mathcal{F}$

6.2 Security Proof

In this section, we provide the formal security proof for the confidentiality and the existential unforgeability of our eCLSC-TKEM. The security of our eCLSC-TKEM relies on the hardness of the following problems.

Definition of OGDH For a group G_q with a generator P and a fixed point Q, the one-sided gap Diffie-Hellman problem (OGDH) [11] is defined as follows: for $x, y \in \mathsf{Z}_q^*$, given Q, R, compute xyP by accessing an one-sided decision Diffie-Hellman (ODDH) Oracle, where $Q = xP$ and $R = yP$.

Definition of ODDH For a group G_q with a generator P and a fixed point Q, the one-sided decision Diffie-Hellman oracle (ODDH) [11] is an oracle that for any $R', S' \in \mathsf{G}_q$ correctly answers the question: Is $z' \equiv xy' \pmod{p}$, where $x, y', z' \in \mathsf{Z}_q^*$ are integers such that $Q = xP, R' = y'P, S' = z'P$?

Definition of ECDLP The elliptic curve discrete log problem (ECDLP) is defined as follows: given a random instance P, Q, find a number $x \in \mathsf{Z}_q^*$ such that $Q = xP$.

Theorem 1. *In the random oracle model, the eCLSC-TKEM is IND-CCA2 secure under the assumption that the one-sided gap Diffie-Hellman (OGDH) problem is intractable.*
The Theorem 1 is proved based on Lemmas 1, 2 and 3. We adopt the security proof techniques from [21]. The proof of Lemma 3 is in the appendix, whereas the proofs of Lemmas 1 and 2 are omitted due to the space limitation.

Lemma 1. In the random oracle model, if there exists an adversary $\mathcal{A}_{\mathcal{I}}$ against the IND-eCLSC-TKEM-CCA2-I security of the eCLSC-TKEM with advantage a non-negligible δ, then an algorithm $\mathcal{C}$ exists that solves the OGDH problem with the following advantage ε

$$\varepsilon \geq \delta \cdot (1 - \frac{q_{ppri}}{q_C \cdot q_{H_0}}) \cdot (1 - \frac{q_{sv}}{q_C \cdot q_{H_0}}) \cdot (\frac{1}{q_C \cdot q_{H_0} - q_{ppri} - q_{sv}}) \cdot (\frac{1}{q_{H_1}})$$

Here, q_{H_0}, q_{H_1}, q_C, q_{ppri} and q_{sv} are the maximum number of queries that the PPT adversary may ask random oracles H_0 and H_1, create (ID_i), extract-partial-private-key queries and extract-secret-value queries.

Lemma 2. In the random oracle model, if there exists an adversary $\mathcal{A}_{\mathcal{II}}$ against the IND-eCLSC-TKEM-CCA2-II security of the eCLSC-TKEM with advantage a non-negligible δ, then there exist an algorithm $\mathcal{C}$ that solves the OGDH problem with the following advantage ε

$$\varepsilon \geq \delta \cdot (1 - \frac{q_{sv}}{q_C \cdot q_{H_0}}) \cdot (1 - \frac{q_{pkR}}{q_C \cdot q_{H_0}}) \cdot (\frac{1}{q_C \cdot q_{H_0} - q_{sv} - q_{pkR}}) \cdot (\frac{1}{q_{H_1}})$$

Here, q_{H_0}, q_{H_1}, q_C, q_{pkR} and q_{sv} are the maximum number of queries that the PPT adversary may ask random oracles H_0 and H_1, create (ID_i), public-key-replacement queries and extract-secret-value queries.

Lemma 3. In the random oracle model, if there exists an adversary $\mathcal{A}_{\mathcal{III}}$ against the IND-eCLSC-TKEM-CCA2-III security of the eCLSC-TKEM with advantage a non-negligible δ, then an algorithm $\mathcal{C}$ exists that solves the OGDH problem with the following advantage ε

$$\varepsilon \geq \delta \cdot (1 - \frac{t^\star}{t}) \cdot (1 - \frac{1}{q}) \cdot (\frac{1}{q_C \cdot q_{H_0} - q_{ppri} - q_{sv}}) \cdot (\frac{1}{q_{H_1}})$$

Here, q_{H_0}, q_{H_1}, q_C, q_{ppri} and q_{sv} are the maximum number of queries that the PPT adversary may ask random oracles H_0 and H_1, create (ID_i), extract-partial-private-key queries and extract-secret-value queries. t denotes the total possible time assuming that the time begins at 0. $t^\star$ is a valid time period of the target identity.

Theorem 2. *In the random oracle model, the eCLSC-TKEM is EUF-CMA secure under the assumption that the elliptic curve discrete logarithm problem (ECDLP) is intractable.* Theorem 2 is proved based on Lemmas 4, 5 and 6. We adopt the security proof techniques from [21]. The proof of Lemma 6 is in the appendix, whereas the proofs of Lemmas 4 and 5 are omitted due to the space limitation.

Lemma 4. In the random oracle model, if there exists a forger $\mathcal{F}_{\mathcal{I}}$ against the EUF-eCLSC-TKEM-CMA-I security of the eCLSC-TKEM with advantage a non-negligible δ, then there exists an algorithm $\mathcal{C}$ that solves the ECDLP with the following advantage ε

$$\varepsilon \geq \delta \cdot q_E \cdot (1 - \frac{q_{H_0} \cdot q_C}{q}) \cdot (1 - \frac{q_{H_2}^2}{q}) \cdot (1 - \frac{q_{H_3}^2}{q}) \cdot (1 + \frac{1}{q}) \cdot (\frac{1}{q_C}) \cdot (1 - \frac{q_{ppri}}{q_{H_0}}) \cdot (1 - \frac{q_{sv}}{q_{H_0}})$$

Here, q_C, q_E, q_{H_i}, q_{ppri} and q_{sv} are the maximum number of queries that the forger may make create (ID_i) queries, key encapsulation queries, random oracle queries to H_i $(0 \leq i \leq 3)$, extract-partial-private-key queries and extract-secret-value queries.

Lemma 5. In the random oracle model, if there exists a forger $\mathcal{F}_{\mathcal{II}}$ against the EUF-eCLSC-TKEM-CMA-II security of the eCLSC-TKEM with advantage a non-negligible δ, then there exists an algorithm $\mathcal{C}$ that solves the ECDLP with the following advantage ε

$$\varepsilon \geq \delta \cdot q_E \cdot (1 - \frac{q_{H_0} \cdot q_C}{q}) \cdot (1 - \frac{q_{H_2}^2}{q}) \cdot (1 - \frac{q_{H_3}^2}{q}) \cdot (1 + \frac{1}{q}) \cdot (\frac{1}{q_C}) \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot (1 - \frac{q_{pkR}}{q_{H_0}})$$

Here, q_C, q_E, q_{H_i}, q_{pkR} and q_{sv} are the maximum number of queries that the forger may make create (ID_i) queries, key encapsulation queries, random oracle queries to H_i $(0 \leq i \leq 3)$, public key replacement queries and extract-secret-value queries.

Lemma 6. In the random oracle model, if there exists a forger $\mathcal{F}_{\mathcal{III}}$ against the EUF-eCLSC-TKEM-CMA-III security of the eCLSC-TKEM with advantage a non-negligible δ, then there exists an algorithm $\mathcal{C}$ that solves the ECDLP with the following advantage ε

$$\varepsilon \geq \delta \cdot q_E \cdot (1 - \frac{q_{H_0} \cdot q_C}{q}) \cdot (1 - \frac{q_{H_2}^2}{q}) \cdot (1 - \frac{q_{H_3}^2}{q}) \cdot (1 + \frac{1}{q}) \cdot (\frac{1}{q_C}) \cdot (1 - \frac{t^\star}{t})$$

Here, q_C, q_E and q_{H_i} are the maximum number of queries that the forger may make create (ID_i) queries, key encapsulation queries, random oracle queries to H_i $(0 \leq i \leq 3)$. t denotes the total possible time and assuming that the time begins at 0. $t^\star$ is a valid time period of target identity

7. CONCLUSIONS AND FUTURE WORKS

In this paper, a secure communication protocol between drones and smart objects is presented. To satisfy security and efficiency requirements, we propose the eCLSC-TKEM with the dual channel strategy. Our protocol efficiently supports four security functions: key agreement, user authentication, non-repudiation, and user revocation. Our experimental analysis carried in a smart parking management testbed shows that our protocol is 1.3, 1.5 and 2.8 times faster than other protocols, i.e., Seo's CLSC-TKEM [21], Sun's CL-AKA [24] and Yang's CL-AKA [26], respectively.

As future work, we plan to investigate how to mitigate the $O(N)$ key update overhead at the KGC (N is the number of users). The overhead at the KGC can be dispersed by introducing intermediate entities between the KGC and drones. The KGC issues a valid partial private key not to directly drones but to the entity who is responsible for drones' operations. The entity can generate proxy partial private keys for drones using its valid partial private key. The key idea of the proxy partial private key might be similar to the proxy certificate or the proxy signature.

Acknowledgments

The work reported in this paper has been partially supported by the Purdue Cyber Center and by the National Science Foundation under grant CNS-1111512.

8. REFERENCES

[1] Low power listening, http://tinyos.stanford.edu/tinyos-wiki/index.php/writing_low-power_applications.

[2] Parrot, http://ardrone2.parrot.com.

[3] S. Al-Riyami and K. Paterson. Certificateless public key cryptography. In C.-S. Laih, editor, *ASIACRYPT*, volume 2894 of *LNCS*, pages 452–473. Springer, 2003.

[4] G. Anastasi, M. Conti, E. Monaldi, and A. Passarella. An adaptive data-transfer protocol for sensor networks with data mules. In *WoWMoM*, pages 1–8, 2007.

[5] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of CRYPTO '01*, pages 213–229. Springer, 2001.

[6] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Security and Privacy*, 2003.

[7] K. Chatterjee, A. De, and D. Gupta. An improved id-based key management scheme in wireless sensor network. In *Advances in Swarm Intelligence*, volume 7332 of *LNCS*, pages 351–359. Springer, 2012.

[8] W. Du, J. Deng, Y. S. Han, and P. K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *CCS '03. Proceedings*, 2003.

[9] M. Geng and F. Zhang. Provably secure certificateless two-party authenticated key agreement protocol without pairing. In *CIS '09*, pages 208–212, 2009.

[10] D. He, J. Chen, and J. Hu. A pairing-free certificateless authenticated key agreement protocol. *Int. Journal of Comm. Sys.*, pages 221–230, 2012.

[11] N. Koblitz and A. Menezes. Intractable problems in cryptography. In *Proc. 9th International Conf. Finite Fields and Their Applications*, 2010.

[12] F. Li, M. Shirase, and T. Takagi. Certificateless hybrid signcryption. In *Information Security Practice and Experience*, volume 5451 of *LNCS*. Springer, 2009.

[13] A. Liu and P. Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *IPSN '08*, pages 245–256, April 2008.

[14] K. MacKay. https://github.com/kmackay/micro-ecc.

[15] S. M. Mizanur Rahman and K. El-Khatib. Private key agreement and secure communication for heterogeneous sensor networks. *J. Parallel Distrib. Comput.*, 70(8):858–870, Aug. 2010.

[16] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *JOURNAL OF CRYPTOLOGY*, 13:361–396, 2000.

[17] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the ACM SenSys '04*, 2004.

[18] A. Rasheed and R. Mahapatra. Secure data collection scheme in wireless sensor network with mobile sink. In *IEEE NCA '08*, 2008.

[19] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava. Optimizing sensor networks in the energy-latency-density design space. *Mobile Computing, IEEE Trans. on*, 1(1):70–80, Jan 2002.

[20] S. Selvi, S. Vivek, and C. Rangan. Certificateless kem and hybrid signcryption schemes revisited. In *Information Security, Practice and Experience*, volume 6047 of *LNCS*, pages 294–307. Springer, 2010.

[21] S. Seo and E. Bertino. Elliptic curve cryptography based certificateless hybrid signcryption scheme without pairing, http://www.cerias.purdue.edu/apps /reports_and_papers/view/4698. *CERIAS report*, '13.

[22] R. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: modeling a three-tier architecture for sparse sensor networks. In *SNPA '03, Proceedings*, 2003.

[23] H. Song, S. Zhu, W. Zhang, and G. Cao. Least privilege and privilege deprivation: Toward tolerating mobile sink compromises in wireless sensor networks. *ACM Trans. Sen. Netw.*, 4(4):23:1–23:34, Sept. 2008.

[24] H. Sun, Q. Wen, H. Zhang, and Z. Jin. A novel pairing-free certificateless authenticated key agreement protocol with provable security. *Frontiers of Computer Science*, 7(4):544–557, 2013.

[25] Y. Tirta, Z. Li, Y.-H. Lu, and S. Bagchi. Efficient collection of sensor data in remote fields using mobile collectors. In *ICCCN '04. Proceedings*, 2004.

[26] G. Yang and C.-H. Tan. Strongly secure certificateless key exchange without pairing. In *ASIACCS*, 2011.

[27] X. Zhang, J. He, and Q. Wei. Eddk: Energy-efficient distributed deterministic key management for wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2011.

[28] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *ACM MobiHoc*, pages 187–198, 2004.

[29] L. Zhou, J. Ni, and C. Ravishankar. Supporting secure communication and data collection in mobile sensor networks. In *INFOCOM '06. Proceedings*, 2006.

APPENDIX

A. PROOF OF LEMMA 3

Suppose that there exists a Type III adversary $\mathcal{A}_{\mathcal{III}}$ who can break the IND-eCLSC-TKEM-CCA2-III security of the eCLSC-TKEM with a non-negligible probability in polynomial time. A challenger $\mathcal{C}$ is challenged with an instance of the OGDH (One-sided Gap Diffie-Hellman) problem. $\mathcal{C}$ can utilize $\mathcal{A}_{\mathcal{III}}$ to compute the solution of the OGDH instance by accessing a ODDH (One-sided Decision Diffie-Hellman)oracle. $\mathcal{C}$ sets the master private/public key pair as $(x, P_{pub} = xP)$, where P is the generator of the group G_q and the hash functions $H_i(0 \leq i \leq 3)$ are treated as random oracles. $\mathcal{C}$ sends the system parameters $\Omega = \{F_q, E/F_q, G_q, P, P_{pub}, H_0, H_1, H_2, H_3\}$ to $\mathcal{A}_{\mathcal{III}}$. To maintain the consistency, $\mathcal{C}$ maintains lists $L_i(0 \leq i \leq 3)$). It also maintains a list of issued private keys and public keys including valid time period in L_k. $\mathcal{C}$ can simulate the challenger's execution of each phase of the formal Game. Let $\mathcal{C}$ select a random index j, where $1 \leq j \leq q_C$ and fix ID_j as the target identity for the challenge phase. Let's that $\mathcal{A}_{\mathcal{III}}$ was revoked at the time interval beginning at $t^\star$.

Phase 1: $\mathcal{A}_{\mathcal{III}}$ may make use of all random oracles $H_i(0 \leq i \leq 3)$ at any time and $\mathcal{C}$ responds as follows:

Create(ID_i): When $\mathcal{A}_{\mathcal{III}}$ submits a Create(ID_i) query to $\mathcal{C}$, $\mathcal{C}$ responds as follows: (1) If case 1 ($ID_i \neq ID_j$) or case 2 ($ID_i = ID_j$ and $t_j < t^\star$), $\mathcal{C}$ picks $e_i, b_i, x_i \in_R Z_q^*$, then sets $H_0(ID_i, R_i, P_i, t_i) = -e_i$, $R_i = e_i P_{pub} + b_i P$ and computes the public key as $P_i = x_i P$. $d_i = b_i$ and it satisfies the equation $d_i P = R_i + H_0(ID_i, R_i, P_i, t_i) P_{pub}$. $\mathcal{C}$ inserts $\langle ID_i, R_i, P_i, t_i, -e_i \rangle$ into the list L_0 and $\langle ID_i, d_i, x_i, R_i, P_i, t_i \rangle$ into the list L_k. (2) If $ID_i = ID_j$ and $t_j > t^\star$, $\mathcal{C}$ chooses $e_j, x_j \in_R Z_q^*$ and sets $H_0(ID_j, R_j, P_j, t_j) = -e_j$, $P_j = x_j P$, and $R_j = e_j P_{pub} - P_j + aP$. Here, $\mathcal{C}$ does not know a. $\mathcal{C}$ uses the aP given in the instance of the OGDH problem. $\mathcal{C}$ inserts $\langle ID_j, R_j, P_j, t_j, -e_j \rangle$ into the list L_0 and

$\langle ID_j, \perp, x_j, R_j, P_j, t_j\rangle$ into the list L_k.

H_0 **queries**: When $\mathcal{A}_{\mathcal{III}}$ submits a H_0 query with ID_i, $\mathcal{C}$ searches the list L_0. If there is a tuple $\langle ID_i, R_i, P_i, t_i, -e_i\rangle$, $\mathcal{C}$ responds with the previous value $-e_i$. Otherwise, $\mathcal{C}$ chooses $e_i \in_R Z_q^*$ and returns $-e_i$ as the answer. Then, $\mathcal{C}$ inserts $\langle ID_i, R_i, P_i, t_i, -e_i\rangle$ into the list L_0.

H_1 **queries**: When $\mathcal{A}_{\mathcal{III}}$ submits a H_1 query with $(Y_i, V_i, T_i, ID_i, P_i)$, $\mathcal{C}$ checks whether the ODDH oracle returns 1 when queried with (aP, V_i, T_i). If the ODDH oracle returns 1, $\mathcal{C}$ outputs T_i and stop. Then $\mathcal{C}$ goes through the L_1 with entries $\langle Y_i, V_i, *, ID_i, P_i, l_i\rangle$, for different values of l_i, such that the ODDH oracle returns 1 when queried on the tuple (aP, V_i, T_i). Note that in this case $ID_i = ID_j$ and $t_j > t^\star$. If such a tuple exists, it returns l_i and replaces the symbol $*$ with T_i. Otherwise, $\mathcal{C}$ chooses $l \in_R \{0,1\}^n$ and updates the L_1, which is initially empty, with a tuple containing the input and return values. $\mathcal{C}$ then returns l to $\mathcal{A}_{\mathcal{III}}$.

H_2 **queries**: $\mathcal{C}$ checks whether $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i\rangle$ exists in the L_2. If it exists, $\mathcal{C}$ returns $H = h_i$ to $\mathcal{A}_{\mathcal{III}}$. Otherwise, $\mathcal{C}$ chooses $h_i \in_R Z_q^*$, adds $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i\rangle$ to the L_2 and returns $H = h_i$ to $\mathcal{A}_{\mathcal{III}}$.

H_3 **queries**: $\mathcal{C}$ checks whether $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i'\rangle$ exists in the L_3. If it exists, $\mathcal{C}$ returns $H' = h_i'$ to $\mathcal{A}_{\mathcal{III}}$. Otherwise, $\mathcal{C}$ chooses $h_i' \in_R Z_q^*$, adds $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i'\rangle$ to the L_3 and returns $H' = h_i'$ to $\mathcal{A}_{\mathcal{III}}$.

Extract-Partial-Private-Key queries: In order to respond to the query for the partial private key of a user with ID_i, $\mathcal{C}$ performs the following steps: (1) If $ID_i = ID_j$ and $t_j > t^\star$, $\mathcal{C}$ aborts the execution. (2) If case 1 ($ID_i = ID_j$) or case 2 ($ID_i = ID_j$ and $t_j < t^\star$), $\mathcal{C}$ retrieves $\langle ID_i, d_i, x_i, R_i, P_i, t_i\rangle$ from L_k, returns (d_i, R_i) which satisfies the equation $d_iP = R_i + H_0(ID_i, R_i, P_i, t_i)P_{pub}$.

Extract-Secret-Value queries: $\mathcal{A}_{\mathcal{III}}$ produces ID_i to $\mathcal{C}$ and requests a secret value of ID_i. If case 1 (the public key of ID_i has not been replaced and $ID_i = ID_j$) or case 2 (the public key of ID_i has not been replaced, $ID_i = ID_j$ and $t_j < t^\star$), then $\mathcal{C}$ responds with x_i by retrieving from L_k. If $\mathcal{A}_{\mathcal{III}}$ has already replaced the public key of ID_i, $\mathcal{C}$ does not provide the corresponding secret value to $\mathcal{A}_{\mathcal{III}}$. If $ID_i = ID_j$ and $t_j > t^\star$, $\mathcal{C}$ aborts.

Request-Public-Key queries: $\mathcal{A}_{\mathcal{III}}$ produces ID_i to $\mathcal{C}$ and requests a public key of ID_i. $\mathcal{C}$ checks in the L_k for $\langle ID_i, d_i, x_i, R_i, P_i, t_i\rangle$. If it exists, $\mathcal{C}$ returns the corresponding public key (R_i, P_i, t_i). Otherwise, $\mathcal{C}$ recalls Create(ID_i) query to obtain (R_i, P_i, t_i) and returns (R_i, P_i, t_i).

Public-Key-Replacement queries: $\mathcal{A}_{\mathcal{III}}$ chooses values (R_i', P_i', t_i') to replace the public key (R_i, P_i, t_i) of ID_i. $\mathcal{C}$ updates the corresponding tuple in the L_k as $\langle ID_i, -, -, R_i', P_i', t_i'\rangle$. The current value of the user's public key is used by $\mathcal{C}$ for responses to any queries made by $\mathcal{A}_{\mathcal{III}}$.

Symmetric Key Generation queries: $\mathcal{A}_{\mathcal{III}}$ produces a sender's ID_A, public key (R_A, P_A, t_A), the receiver's ID_B and public key (R_B, P_B, t_B) to $\mathcal{C}$. For each query (ID_A, ID_B), $\mathcal{C}$ proceeds as follows: (1) If case 1 ($ID_A = ID_j$) or case 2 ($ID_A = ID_j$ and $t_j < t^\star$), $\mathcal{C}$ computes sk_A corresponding to ID_A by executing the Extract-Partial-Private-Key and Extract-Secret-Value algorithm. Then, $\mathcal{C}$ gets K and ω by running the actual SymmetricKeyGen algorithm. $\mathcal{C}$ stores ω and overwrite any previous value. $\mathcal{C}$ sends K to $\mathcal{A}_{\mathcal{III}}$. (2) If $ID_A = ID_j$ and $t_j > t^\star$, $\mathcal{C}$ chooses $r_1, r_2, h_t, h_t' \in_R Z_q^*$ and computes $U = r_1P - {h_t}^{-1} \cdot aP + {h_t}^{-1} \cdot P_t$, $V = r_2P$, $Y = R_B + H_0(ID_B, R_B, P_B, t_B) \cdot P_{pub} + P_B$, $T = r_2 \cdot Y \bmod q$=$r_2 \cdot (H_0(ID_B, R_B, P_B, t_B)P_{pub} + R_B + P_B) \bmod q$ and $K = H_1(Y, V, T, ID_B, P_B)$, where R_B and P_B are obtained by calling the Request-Public-Key query oracle on ID_B. Note that ω is $\omega = (r_1, r_2, h_t, h_t', U, V, T, ID_A, pk_A, ID_B, pk_B)$. (3) $\mathcal{C}$ goes through the L_1 looking for an entry (Y, V, T, ID_B, P_B, k) for some k such that ODDH(P_B, V, Y)=1. If such an entry exists, it computes $K \leftarrow l$. Otherwise it uses a random l and updates the L_1 with $(Y, V, *, ID_B, P_B, l)$. $\mathcal{C}$ stores ω and sends K to $\mathcal{A}_{\mathcal{III}}$.

Key Encapsulation queries: $\mathcal{A}_{\mathcal{III}}$ produces an arbitrary tag τ, the sender's ID_A, public key (R_A, P_A, t_A), the receiver's ID_B and public key (R_B, P_B, t_B) and sends them to $\mathcal{C}$. The full private key of the sender $sk_A = (d_A, x_A)$ is obtained from the L_k. $\mathcal{C}$ checks whether a corresponding ω value has been stored previously. (1) If ω does not exist, $\mathcal{C}$ returns an invalid reply. (2) If case 1 (a corresponding ω exists and $ID_A = ID_j$) or case 2 (a corresponding ω exists, $ID_A = ID_j$ and $t_j < t^\star$), then $\mathcal{C}$ computes φ with ω and τ by using the actual Encapsulation algorithm, and deletes ω. (3) If a corresponding ω exists, $ID_A = ID_j$ and $t_j > t^\star$, then $\mathcal{C}$ computes φ by performing the following steps. Note that ω is $(r_1, r_2, h_j, h_j', U, V, T, ID_A, pk_A, ID_B, pk_B)$ and $\mathcal{C}$ does not know the private key corresponding to ID_t. So $\mathcal{C}$ should perform the encapsulation in a different way:

- $H = h_j$ and add $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_j\rangle$ to L_2.
- $H' = h_j'$ and add $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_j'\rangle$ to L_3.
- Compute $W = h_j \cdot r_1 + h_j' \cdot x_A$.
- Output $\varphi = (U, V, W)$ as the encapsulation.

We show that $\mathcal{A}_{\mathcal{III}}$ can pass the verification of $\varphi = (U, V, W)$ to validate the encapsulation, because the equality $W \cdot P = R_A + H_0(ID_A, R_A, P_A, t_A) \cdot P_{pub} + H \cdot U + H' \cdot P_A$ holds as follows: $R_A + H_0(ID_A, R_A, P_A, t_A) \cdot P_{pub} + H \cdot U + H' \cdot P_A = aP + e_jP_{pub} - P_j + (-e_j) \cdot P_{pub} + H \cdot (r_1P - {h_j}^{-1} \cdot aP + {h_j}^{-1} \cdot P_j) + H' \cdot P_A = h_j \cdot r_1P + h_j' \cdot P_A$$= W \cdot P$

Key Decapsulation queries: $\mathcal{A}_{\mathcal{III}}$ produces an encapsulation $\varphi = (U, V, W)$, a tag τ, the sender's ID_A, the public key (R_A, P_A, t_A), the receiver's ID_B and the public key (R_B, P_B, t_B) to $\mathcal{C}$. The full private key of the receiver $sk_B = (d_B, x_B)$ is obtained from the list L_k. (1) If case 1 ($ID_B = ID_j$) or case 2 ($ID_B = ID_j$ and $t_j < t^\star$), then $\mathcal{C}$ computes the decapsulation of φ by using the actual Decapsulation algorithm. (2) If $ID_B = ID_t$ and $t_j > t^\star$, then the point cannot be computed. In order to return a consistent answer, $\mathcal{C}$ computes K from φ as follows:

- Searches in the L_2 and L_3 for entries $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_j\rangle$ and $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_j'\rangle$, respectively.
- If $H = h_j$ and $H' = h_j'$ exist then $\mathcal{C}$ checks whether the equality $W \cdot P = R_A + H_0(ID_A, R_A, P_A, t_A) \cdot P_{pub} + H \cdot U + H' \cdot P_A$ holds.
- If the above equality holds, the T is retrieved from the L_2 and L_3. Both the T values should be equal.
- $\mathcal{C}$ goes through L_1 and looks for $\langle Y, V, T, ID_B, P_B, l\rangle$ such that the ODDH oracle returns 1 when queried on the (aP, V, T). If such entry exists, the corresponding $K \leftarrow l$ value is returned as the decapsulation of φ.
- If $\mathcal{C}$ reaches this point of execution, it puts the entry $\langle Y, V, *, ID_B, P_B, l\rangle$ for a random l on the L_1 and returns $K \leftarrow l$. The $*$ denotes an unknown value. The identity component with $*$ is a receiver ID_B.

Challenge: At the end of Phase I, $\mathcal{A}_{\mathcal{III}}$ sends a sender's ID_{A*} and a receiver's ID_{B*} to $\mathcal{C}$. Here, the partial private

key of the revoked receiver was not queried in Phase I. Let the time ${t'^*}_j \in ({t^*}_j, {t^*}_j + \alpha)$. $\mathcal{C}$ aborts the game if case 1 ($ID_{B*} = ID_j$) or case 2 ($ID_{B*} = ID_j$ and ${t'^*}_j < t^\star$). Otherwise, $\mathcal{C}$ performs the following steps to compute the challenge encapsulation φ^*: (1) Choose $r \in_R Z_q^*$ and compute $U^* = rP$. (2) Set $V^* = bP$ and choose $T^* \in_R G_q$. Here, $\mathcal{C}$ does not know b. $\mathcal{C}$ uses the bP given in the instance of the OGDH problem. (3) Choose $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the eCLSC-TKEM. (4) Choose a random hash value l^* and set $K_1 = l^*$. (5) $\mathcal{C}$ chooses a bit $\delta \in_R \{0,1\}$ and sends K_δ to $\mathcal{A_{III}}$. (6) $\mathcal{A_{III}}$ generates τ^* and sends it to $\mathcal{C}$. (7) Choose $h_i, h'_i \in_R \mathbb{Z}_q^*$, store $\langle U^*, \tau^*, T^*, ID_{A*}, P_{A*}, ID_{B*}, P_{B*}, h_i\rangle$ to the L_2 and $\langle U^*, \tau^*, T^*, ID_{A*}, P_{A*}, ID_{B*}, P_{B*}, h'_i\rangle$ to the L_3. (8) Since $\mathcal{C}$ knows the sender's private key, $\mathcal{C}$ computes $W^* = d_{A*} + r \cdot h_i + x_{A*} \cdot h'_i$. (9) $\mathcal{C}$ returns $\varphi^* = \langle U^*, V^*, W^*\rangle$.
Phase II: $\mathcal{A_{III}}$ adaptively queries the oracles as in Phase I. Besides it cannot query decapsulation on φ^*.
Guess: Since $\mathcal{A_{III}}$ can break the IND-eCLSC-TKEM-CCA2-III security (which is assumed at the beginning of the proof), $\mathcal{A_{III}}$ should have asked a H_1 query with $(Y^*, V^*, T^*, ID_{B^*}, P_{B^*})$ as inputs. It is to be noted that $T^* = b \cdot Y^* = b \cdot (-e_j \cdot P_{pub} + e_j \cdot P_{pub} - P_j + aP + P_{B^*}) = ab \cdot P$, where $P_j = P_{B*}$ because of $ID_j = ID_{B*}$. Therefore, if the L_1 has q_{H_1} queries corresponding to the sender ID_{A^*} and receiver ID_{B^*}, one of the T^*'s among q_{H_1} values stored in the list L_1 is the solution for the OGDH problem instance. $\mathcal{C}$ chooses one T value uniformly at random from the q_{H_1} values from the L_1 and outputs it as the solution for the OGDH instance.
Analysis: $\mathcal{C}$ lets E_1, E_2 and E_3 be the events in which $\mathcal{C}$ aborts the IND-eCLSC-TKEM-CCA2-III game.
(1) E_1: The $\mathcal{A_{III}}$ returns decapsulation for ${t'^*}_j < t^\star$). The probability is $Pr[E_1] = \frac{t^\star}{t}$. t denotes the total possible time and assuming that the time begins at 0.
(2) E_2: An invalid public key replacement by $\mathcal{A_{III}}$ was not detected. The probability is $Pr[E_2] = \frac{1}{q}$.
(3) E_3: $\mathcal{A_{III}}$ does not choose the target identity ID_j during the challenge. The probability is $Pr[E_3] = 1 - \frac{1}{q_C \cdot q_{H_0} - q_{ppri} - q_{sv}}$.
Thus, the probability that $\mathcal{C}$ does not abort the IND-eCLSC-TKEM-CCA2-III game is

$$Pr[\neg E_1 \wedge \neg E_2 \wedge \neg E_3] = (1 - \frac{t^\star}{t}) \cdot (1 - \frac{1}{q}) \cdot (\frac{1}{q_C \cdot q_{H_0} - q_{ppri} - q_{sv}})$$

The probability that $\mathcal{C}$ randomly chooses the T from L_1 and T is the solution of OGDH problem is $\frac{1}{q_{H_1}}$. So, the probability that $\mathcal{C}$ finds the OGDH instance is as follows:

$$Pr[\mathcal{C}] = \delta \cdot (1 - \frac{t^\star}{t}) \cdot (1 - \frac{1}{q}) \cdot (\frac{1}{q_C \cdot q_{H_0} - q_{ppri} - q_{sv}}) \cdot (\frac{1}{q_{H_1}})$$

Therefore, the $Pr[\mathcal{C}]$ is non-negligible, because δ is non-negligible. This contradicts the OGDH assumption.

B. PROOF OF LEMMA 6

A challenger $\mathcal{C}$ is challenged with an instance of the ECDLP. To solve the ECDLP, given $\langle P, bP\rangle \in G_q$, $\mathcal{C}$ must find b. Let $\mathcal{F_{III}}$ be a forger who is able to break the EUF-eCLSC-TKEM-CMA-III security of the eCLSC-TKEM. $\mathcal{C}$ can utilize $\mathcal{F_{III}}$ to compute the solution b of the ECDLP instance by playing the following interactive game with $\mathcal{F_{III}}$. To solve the ECDLP, $\mathcal{C}$ sets the master private/public key pair as $(x, P_{pub} = xP)$, where P is the generator of the group G_q and the hash functions $H_i (0 \le i \le 3)$ are treated as random oracles. The $\mathcal{C}$ sends the system parameter $\Omega = \{F_q, E/F_q, G_q, P, P_{pub}, H_0, H_1, H_2, H_3\}$ to $\mathcal{F_{III}}$. In order to avoid the inconsistency between the responses to the hash queries, $\mathcal{C}$ maintains lists $L_i (0 \le i \le 3)$). It also maintains a list L_k to maintain the list of issued private keys and public keys including the valid time period. $\mathcal{C}$ can simulate the Challenger's execution of each phase of the formal game.
Training Phase: $\mathcal{F_{III}}$ may make a series of polynomially bounded number of queries to random oracles $H_i (0 \le i \le 3)$ at any time and $\mathcal{C}$ responds as follows: All the oracles and queries needed in the training phase are identical to those of the Create(ID_i) queries, H_0 queries, H_1 queries, H_2 queries, H_3 queries, Extract-Partial-Private-Key queries, Extract-Secret-Value queries, Public-Key-Replacement queries, Symmetric Key Generation queries, Key Encapsulation queries and Key Decapsulation queries in IND-pCLSC-TKEM-CCA2-III game.
Forgery: Eventually, $\mathcal{F_{III}}$ returns a valid encapsulation $\langle \tau, \varphi = (U, V, W), ID_A, ID_B\rangle$ on a arbitrary tag τ, where ID_A is the sender identity and ID_B is the receiver identity, to $\mathcal{C}$. If $ID_A = ID_j$ and ${t'^*}_j > t^\star$, $\mathcal{C}$ aborts the execution of this game. Otherwise, $\mathcal{C}$ searches the list L_2 and outputs another valid encapsulation $\langle \tau, \varphi^* = (U, V, W^*), ID_A, ID_B\rangle$ with different h_i^* such that $h_i^* = h_i$ on the same τ as done in forking lemma [16]. Thus, we can get $W \cdot P = R_A - e_j \cdot P_{pub} + h_i \cdot U + h'_i \cdot P_A$ and $W^* \cdot P = R_A - e_j \cdot P_{pub} + h_i^* \cdot U + h'_i \cdot P_A$. Let $U = bP$. Then if we subtract these two equations, we get following value.
$W^* \cdot P - W \cdot P = h_i^* \cdot U - h_i \cdot U$
$\Rightarrow (W^* - W)P = (h_i^* - h_i) \cdot U$
$\Rightarrow (W^* - W)P = (h_i^* - h_i) \cdot bP$
$\Rightarrow (W^* - W) \cdot (h_i^* - h_i)^{-1} = b$
Therefore, $\mathcal{F_{III}}$ solves the ECDLP as $b = \frac{W^* - W}{h_i^* - h_i}$ using the algorithm $\mathcal{C}$ for given a random instance $\langle P, bP\rangle \in G_q$.
Analysis: In order to assess the probability of success of the challenger $\mathcal{C}$. We assume that $\mathcal{F_{III}}$ can ask q_C create (ID_i) queries, q_E key-encapsulation queries and q_{H_i} random oracle queries to H_i ($0 \le i \le 3$). We also assume that $\mathcal{F_{III}}$/ never repeats H_i ($0 \le i \le 3$) a query with the same input.
(1) The success probability of the Create(ID_i) query execution is $(1 - \frac{q_{H_0}}{q})^{q_C} \ge 1 - \frac{q_{H_0} \cdot q_C}{q}$.
(2) The success probability of the H_2 query execution is $(1 - \frac{q_{H_2}}{q})^{q_{H_2}} \ge 1 - \frac{q_{H_2}^2}{q}$.
(3) The success probability of the H_3 query execution/ is $(1 - \frac{q_{H_3}}{q})^{q_{H_3}} \ge 1 - \frac{q_{H_3}^2}{q}$.
(4) The success probability of the key encapsulation query execution is $\frac{q_E}{(1 - \frac{1}{q})} \ge q_E \cdot (1 + \frac{1}{q})$.
(5) The probability of both $ID_i = ID_j$ and ${t'^*}_j > t^\star$ is $\frac{1}{q_C} \cdot (1 - \frac{t^\star}{t})$. t denotes the total possible time and assuming that the time begins at 0.
Thus, the success probability that $\mathcal{C}$ can win the EUF-eCLSC-TKEM-CMA-III game is

$$\varepsilon \ge \delta \cdot q_E \cdot (1 - \frac{q_{H_0} \cdot q_C}{q}) \cdot (1 - \frac{q_{H_2}^2}{q}) \cdot (1 - \frac{q_{H_3}^2}{q}) \cdot (1 + \frac{1}{q}) \cdot (\frac{1}{q_C}) \cdot (1 - \frac{t^\star}{t})$$

Therefore, the probability that $\mathcal{C}$ computes the solution of ECDLP is non-negligible, because δ is non-negligible.