Received January 31, 2017, accepted March 13, 2017, date of publication March 20, 2017, date of current version April 24, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2684128

Certificateless Cryptographic Protocols for Efficient Drone-Based Smart City Applications

JONGHO WON[1], (Student Member, IEEE), SEUNG-HYUN SEO[2], (Member, IEEE), AND ELISA BERTINO[1], (Fellow, IEEE)

[1]Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA
[2]Division of Electrical Engineering, Hanyang University, Ansan 15588, South Korea

Corresponding authors: J. Won (won12@purdue.edu) and S.-H. Seo (seosh77@hanyang.ac.kr)

This work was supported in part by the Purdue Cyber Center, in part by the National Science Foundation under Grant ACI-1547358, and in part by the National Research Foundation of Korea grant funded by the Korean government under Grant 2015R1C1A1A01052491. This paper was presented at the ACM Asia Conference on Computer and Communications Security 2015 [1].

ABSTRACT Smart cities aim to improve the quality of urban services and their energy efficiency by utilizing information and communication technologies. In such context, drones can be utilized to support various services, such as traffic monitoring, search/rescue, and surveillance, by communicating with many different smart objects like sensors. Securing such communications is crucial to making correct decisions and requires efficient cryptographic protocols. However, the design of such protocols must consider: 1) the mobility and the limited battery of drones and 2) the constrained resources of smart objects. In this paper, a suite of cryptographic protocols is presented to deal with three different communication scenarios: one-to-one, one-to-many, and many-to-one. For one-to-one, we propose an efficient Certificateless Signcryption Tag Key Encapsulation Mechanism (eCLSC-TKEM) that supports authenticated key agreement, non-repudiation, and user revocation. 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. For one-to-many, we propose a Certificateless Multi-Recipient Encryption Scheme (CL-MRES) by which a drone can efficiently send privacy-sensitive data to multiple smart objects. For many-to-one, we present a Certificateless Data Aggregation (CLDA) protocol, which allows drones to efficiently collect data from hundreds of smart objects. Also, for efficiency, we propose a dual channel strategy that allows many smart objects to concurrently execute our protocols. We evaluate eCLSC-TKEM via a smart parking management test-bed. Also, we have implemented CL-MRES and CLDA on a board with a graphics processing unit (GPU) and show their GPU-accelerated performance.

INDEX TERMS Certificateless cryptography, certificateless signcryption, data collection, drone, smart city.

I. INTRODUCTION

With the advent of low-cost general-purpose computers and the availability of inexpensive sensors, actuators and wireless transceivers, the interconnected physical objects, called Internet-of-Things (IoT), is being promoted. IoT applications that leverage cloud computing and analytics for big data are enabling smart city initiatives all over the world. The main goal of a smart city is to enhance the quality of urban life and to provide a sustainable environment by monitoring and controlling the city's public infrastructure and services. In such a context, drones represent a key technology for deploying novel monitoring applications. For example, the Wisconsin state police recently reported that drones would help with search and rescue. The Ministry of Environment and Water in Dubai has started using drones for monitoring the work of crushers and quarries. Nokia [2] plans to use flying drones as a public safety system for cities. PrecisionHawk [3] has been offering remote sensing and data processing services using drones for various applications such as infrastructure monitoring and search/rescue.

Recent advances in sensor and embedded device technologies are also pushing the pervasive data acquisition and processing capabilities in different city infrastructures such as roads, traffic signals, sidewalks and bridges to monitor

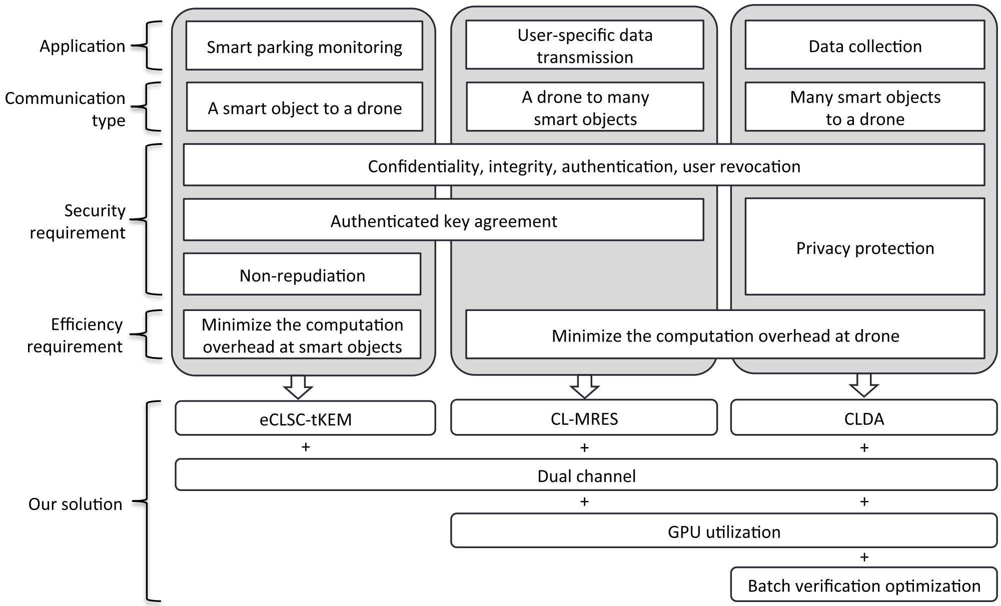

Application
Communication type
Security requirement
Efficiency requirement
Our solution
Smart parking monitoring
User-specific data transmission
Data collection
A smart object to a drone
A drone to many smart objects
Many smart objects to a drone
Confidentiality, integrity, authentication, user revocation
Authenticated key agreement
Privacy protection
Non-repudiation
Minimize the computation overhead at smart objects
Minimize the computation overhead at drone
eCLSC-tKEM
CL-MRES
CLDA
+
Dual channel
GPU utilization
Batch verification optimization

FIGURE 1. Requirements and our solutions.

various city-related information, such as traffic conditions, air quality and structural health. For example, in structural health monitoring [4], sensors can be deployed on structural critical points, such as boundaries or joints of a bridge. After a critical event, such as an earthquake, a drone can fly over sensors to collect data about structural conditions from the sensors. Also, drones can be utilized to update sensor software or to change sensor configuration settings such as sample rates. Even cars may play a role as sensors in smart cities. Drones can collect traffic information from cars to enhance travel efficiency and physical safety. In such context, drones can be used to periodically collect information from these sensors and perform in-network processing of this information.

In the smart city applications based on drones, security is an important requirement. Drones, like many network-enabled mobile devices, are vulnerable to cyber/physical attacks, such as eavesdropping, manipulation, impersonation and physical capture. Furthermore, since drones carrying valuable data might fly over hostile urban areas, they might become the targets of attacks. Therefore, it is critical to address security requirements, such as confidentiality, integrity, authentication, revocation, authenticated key agreement, non-repudiation and privacy protection. However, supporting all the security requirements in one protocol is not desirable since each security functionality requires additional computational costs. Thus, it is crucial to define essential security requirements according to specific categories of applications. In addition, efficiency in applications involving both drones and sensors (referred to as smart objects in what follows) is critical because of (1) the mobility and limited battery life of drones and (2) the constrained resources of smart objects. In particular, it is critical that security protocols take into account the asymmetry in computational power of the devices involved in the applications (e.g. smart objects and drones). In this paper, we address all those requirements by designing, implementing, and testing a suite of efficient cryptographic protocols.

A. CONTRIBUTIONS AND PROTOCOL OVERVIEW

The contributions of this paper are three-fold: 1) a suite of cryptographic protocols, 2) efficiency enhancement techniques to these protocols, and 3) a test-bed implementation of these protocols in different settings.

1) A SUITE OF CRYPTOGRAPHIC PROTOCOLS

As shown in Fig. 1, we consider three different communication types between a drone and smart objects, and their corresponding applications: 1) a smart object → a drone (secure monitoring), 2) a drone → many smart objects (user-specific data transmission), and 3) many smart objects → a drone (data collection). Fig. 1 also shows different security/efficiency requirements for each application. To deal with such requirements, we introduce three cryptographic protocols: 1) an efficient Certificateless SignCryption Tag Key Encapsulation Mechanism (eCLSC-TKEM), 2) a Certificateless Multi-Recipient Encryption Scheme (CL-MRES), and 3) a Certificateless Data Aggregation (CLDA).

1) *eCLSC-TKEM*: eCLSC-TKEM is best-suited when a smart object sends privacy-sensitive messages to a drone and the messages must not be repudiated. The smart parking management presented in Sec. V-B is an example application of eCLSC-TKEM. The main feature of eCLSC-TKEM is to integrate one-way key agreement with digital signature to create one efficient

algorithm which can be used to support authenticated key agreement and non-repudiation. Another advantage of eCLSC-TKEM is that it is based on certificateless public key cryptography (CL-PKC). This means that eCLSC-TKEM does not have the key escrow problem that affects identity-based public key cryptography (ID-PKC) [5], nor does it have the certificate management overhead which exists in the certificate-based public key cryptography.

eCLSC-TKEM adopts Boneh et al.'s revocation scheme [5] to revoke users. That is, when a partial private key is generated, the validity period of the key is specified. After the period expires, the partial private key is automatically revoked and a new partial private key must be generated. Therefore, even if the partial private key of a drone is stolen by an attacker, the malicious use of the key is limited to the period.

Another design goal of eCLSC-TKEM is to increase efficiency by minimizing the computational cost at the smart object. In heterogeneous systems, devices have different computing capabilities and thus the overall execution time of cryptographic operations is dominated by the execution time of low-end devices. eCLSC-TKEM is best-suited to heterogeneous systems, like drone-based smart city applications, since drones are usually equipped with high-end mobile processors, while smart objects have low-speed processors.

2) *CL-MRES*: CL-MRES is a hybrid encryption for multiple recipients and is designed for a drone to efficiently and securely transmit user-specific data to a large number of smart objects. To build CL-MRES, we utilize a random re-use technique and our eCLSC-TKEM excluding the digital signature functionality. Since the drone must deal with a large number of smart objects, the computation overhead at the drone should be minimized. Although CL-MRES does not support non-repudiation, it significantly reduces computational and communication overhead on the drone compared to when the drone uses eCLSC-TKEM for each smart object.
3) *CLDA*: Based on the security of eCLSC-TKEM, we also propose a Certificateless Data Aggregation (CLDA) protocol. For smart city monitoring services, sensors can be embedded in city infrastructure or even cars and smart phones may play the role of sensors. A drone can be used to collect data from hundreds of such sensors. Every collected value must be authenticated to prevent data pollution attacks and encrypted to assure data confidentiality and privacy. CLDA allows drones to efficiently collect data from hundreds of smart objects by utilizing the EC-ElGamal homomorphic encryption and an optimized batch verification technique.

2) EFFICIENCY ENHANCEMENT TECHNIQUES

Along with these three cryptographic protocols, we introduce three additional techniques to enhance the performance of our protocols.

1) *Dual channel strategy*: A drone has a limited flight time. The dual channel strategy helps drones conserve their battery life by allowing them to concurrently execute the time-consuming crypto-algorithms.
2) *GPU utilization*: When a drone must deal with a large number of smart objects in a short time period, it is critical to minimize the execution time of crypto-algorithms at the drone so that the drone saves its flight time. If the drone is equipped with a GPU, the execution time can be significantly reduced.
3) *Batch verification optimization*: When a drone collects data and signatures from a large number of smart objects, the overall performance of CLDA relies on the efficiency of signature verification at the drone. We introduce a batch verification optimization technique to boost the speed of the verification procedure.

3) TEST-BED IMPLEMENTATION

We have implemented our secure communication protocols for real drone applications, i.e., smart parking management and traffic monitoring. For the implementations, we consider two kinds of drones: a *medium-capacity* drone and a *high-capacity* drone. A medium-capacity drone has a moderate-speed CPU and is used as a patrol drone for smart parking management. A high-capacity drone has a GPU as well as a CPU and is used as a large-scale data collector. The performance of eCLSC-TKEM has been evaluated in a smart parking management test-bed consisting of a medium-capacity drone, i.e., AR.Drone2.0 and several sensors, i.e., TelosBs.

To show the performance of CL-MRES and CLDA, we have implemented them on Nvidia Tegra K1, which is a GPU-enabled SoC used in many modern vehicles, such as Audi and Tesla. GPUs, together with cameras, are essential parts for high-capacity drones for image processing, e.g., for obstacle recognition and collision avoidance. We show that the performance of CL-MRES and CLDA can be significantly boosted by a GPU and the batch verification optimization technique.

B. ORGANIZATION OF THE PAPER

The remainder of this paper is organized as follows: In Section II, we present related work. In Section III, we provide relevant background. In Section IV, we introduce our eCLSC-TKEM, CL-MRES, CLDA, and the dual channel strategy. In Section V, we describe the design of our protocols through example applications. Then, the performance of our protocols is evaluated in Section VI. In Section VII, we outline conclusions.

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 [12]	9EM + 1V (8EM + 1V)	yes	yes	no	no
Sun's CL-AKA [13]	6EM (5EM)	yes	yes	no	no
Selvi's CLSC-TKEM [14]	4EM+1P+1EX (3EM+1P+1EX)	yes	yes	yes	no
Seo's CLSC-TKEM [15]	5EM (3EM)	yes	yes	yes	no
eCLSC-TKEM [1]	4EM (2EM)	yes	yes	yes	yes

EM: EC point multiplication, V: signature verification, P: pairing, EX: modular exponentiation. '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.

II. RELATED WORK

A. MOBILE DATA COLLECTORS IN WSN

Several studies [6]–[9] have shown that mobile agents that collect data from static sensors can improve energy efficiency, reliability, connectivity and cost. However, the use of mobile collectors presents new security challenges. Once a mobile collector has collected data and becomes a privileged node, it may be subject to loss or capture, which would allow the data to be viewed by unintended parties. Zhou *et al.* [6] analyzed the impact of compromised mobile collectors on reliability and introduced a key pre-distribution scheme that is resilient against node capture attacks. Song *et al.* [7] introduced a privilege-based pairwise key establishment protocol. In this protocol, when a compromise of a mobile collector is detected, the privileges of the mobile collector are immediately revoked. Rasheed *et al.* [8] proposed a data collection scheme which uses hash chains that allow sensors to authenticate the mobile data collector. This scheme works only when the mobile collector traverses a deterministic path. Rasheed *et al.* [9] proposed a three-tier security scheme for authentication and pairwise key establishment. This scheme requires two separate key pools, one for pairwise key establishment between sensors, and one for a mobile collector to access the network. The two separate key pools enhance network resistance to mobile collector replication attacks. Although these schemes improve security against mobile collector compromises, they are not scalable because they are based on symmetric key pre-distribution. In this paper, we address the scalability problem by designing our protocols based on asymmetric key cryptography and minimizing the computational overhead at low-end devices like sensors.

Previous schemes [10], [11] have made use of multiple radios in order to reduce the sensor energy consumption or to increase the contact time between a sensor and a mobile collector. However, those schemes did not address the problem of system performance degradation caused by slow asymmetric cryptography executions at low-end sensors.

B. CLSC-TKEM AND CL-AKA

Authenticated Key Agreement (AKA) is a protocol that allows users to share a secret key over an insecure network only when they are authenticated. However, AKA based on traditional certificates inherits the certificate management overhead, whereas AKA based on ID-PKC has the key escrow problem.

To address those issues, Al-Riyami et al. proposed certificateless public key cryptography (CL-PKC) [16]. Thereafter, several AKA schemes based on CL-PKC were introduced. These schemes were designed based on pairing-based cryptography (PBC). However, since the time required to compute a pairing operation is much greater than the time required to compute other standard operations, e.g., EC point multiplication, these PBC-based protocols are not suitable for systems with low-end devices like sensors. Despite the recent advances in implementation techniques, one pairing computation is 2 times to 7 times slower than one EC point multiplication depending on the parameters and hardware [17]. Several pairing-free CL-AKA protocols [12], [13], [18], [19] 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 [13] and Yang's CL-AKA [12]. Recently, Li *et al.* [20] 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.* [14] showed a security weakness in Li et al.'s CLSC-TKEM and presented an improved CLSC-TKEM. Since both CLSC-TKEM protocols [14], [20] rely on bilinear pairing operations, they are not suitable for resource-constrained devices.

Seo *et al.* [15] first 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 address 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 prevent permanent exploitation of a compromised private key, eCLSC-KTEM adopts Boneh et al.'s revocation scheme [5]. In eCLSC-TKEM, the key generation center (KGC) inserts a time period as an input when it generates a partial private key for a user. As a result, the partial private key is only valid for the time period. If the time period expires, a new private key must be generated. By inserting this time period, we limit the malicious use of the key even if it is leaked. To revoke a compromised drone, the KGC stops generating a partial private key for 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 CL-AKA and CLSC-TKEM.

C. RANDOM RE-USE AND MULTI-RECIPIENT MULTI-MESSAGE PUBLIC KEY ENCRYPTION

A multi-recipient multi-message public key encryption (MR-MM-PKE) scheme enables a sender to simultaneously encrypt multiple messages for multiple receivers in a single operation. Kurosawa [21] first presented the security model for an MR-MM-PKE scheme and proposed random re-use constructions based on ElGamal and Cramer-Shoup encryption. The random re-use MR-MM-PKE constructions use an ordinary encryption scheme to encrypt messages by using the same random for their respective receivers. Depending on the structure of the encryption scheme, the random re-use technique can significantly reduce the computational and communication overhead while the used encryption scheme remains secure under random re-use. Kurosawa claimed that both ElGamal and Cramer-Shoup encryptions are secure in this setting, while reducing the cost of computation by almost 50%, compared to encrypting messages individually. However, the MR-MM-PKE security model by Kurosawa does not consider inside attackers such as malicious receivers. Bellare *et al.* [22] addressed the weaknesses of Kurosawa's security model and introduced a strengthened security model for the MR-MM-PKE scheme which considers insider attackers. Bellare *et al.* also introduced the concept of reproducibility for an encryption scheme and proved that all the schemes with reproducibility are amenable to a generic conversion to an MR-MM-PKE by employing random re-use. Smart [23] introduced the concept of multi-recipient key encapsulation (MR-KEM) and Barbosa et al. [24] introduced MR-KEM in the identity-based public key cryptography setting. MR-KEM can be constructed as an MR-PKE scheme by adding data encapsulation mechanism (DEM); however, MR-KEM [23] supports only a single-key MR-KEM that generates the same session key for all the recipients. It is limited to the applications where the same message is encrypted for all the receivers. Recently, Pinto *et al.* [25] have revisited the security model of the MR-MM-PKE scheme and presented the notion of a multi-recipient multi-key key encapsulation mechanism (MR-MK-KEM). They proposed the MR-MM-PKE scheme by combining this KEM with an appropriate data encapsulation mechanism (DEM). In this paper, we propose the CL-MRES (Certificateless Multi-Recipient Encyption Scheme) as a hybrid encryption for multiple recipients. To build CL-MRES, we utilize a random re-use technique and our eCLSC-TKEM excluding the digital signature functionality. Our CL-MRES efficiently supports multi-message encryption for multiple recipients as a certificateless hybrid approach.

D. HOMOMORPHIC ENCRYPTION IN WSN

In WSNs, the sensed data might be stored in the network and processed in intermediate nodes to reduce communication overhead and the required amount of storage. To minimize information leakage when a sensor node is compromised, in-network data aggregation schemes based on homomorphic encryption have been proposed [26], [27]. They mainly focus on the optimized implementations of the Elliptic Curve-based ElGamal (EC-ElGamal) homomorphic encryption on resource-constrained devices. In this paper, we show how to merge the EC-ElGamal homomorphic encryption with our certificateless approach. Only authenticated smart objects can send valid sensed values and only an authenticated collector can obtain the aggregate sum of these values. The encrypted sensed values from smart objects are homomorphically aggregated in a drone to save the drone's storage, computational overhead and communication overhead, and to preserve the privacy of the smart objects.

E. COMPARISON WITH OUR PREVIOUS WORK

In our previous work [1], we introduced eCLSC-TKEM and the dual channel strategy. Although these schemes can handle security and efficiency issues in one-to-one communication scenarios, they are not suitable other communication scenarios commonly used in smart city applications, such as one-to-many and many-to-one. In the current paper, we address security and efficiency issues in one(many)-to-many(one) communication scenarios by introducing CL-MRES and CLDA as cryptography protocols. Also, we introduce a GPU utilization technique and a batch verification optimization technique to enhance their performance.

III. BACKGROUND

A. GPU-UTILIZATION FOR ELLIPTIC CURVE CRYPTOGRAPHY

Recent work has shown that elliptic curve cryptography (ECC) can be accelerated by a GPU. There are two approaches for the use of a GPU: *multi-threads for one EC point multiplication* [28], [29] and *single-thread for one EC point multiplication* [30]. The former focuses on improving the computation time of one EC point multiplication. It divides one EC point multiplication procedure into independent subtasks that can be computed by several threads in parallel. This approach aims at keeping all threads busy so that no GPU resources are wasted. However, evenly dividing an EC point multiplication algorithm is difficult due to the sequential nature of the EC point multiplication algorithm. On the other hand, the latter aims at high throughput, i.e., increasing the number of EC point multiplications per second. This approach can achieve high GPU-utilization since one thread computes one EC point multiplication. However, it suffers from higher latency when the GPU must deal with a few EC point multiplications. We adopted the latter approach since the GPU is utilized in our protocol when a large number of EC point multiplications need to be computed.

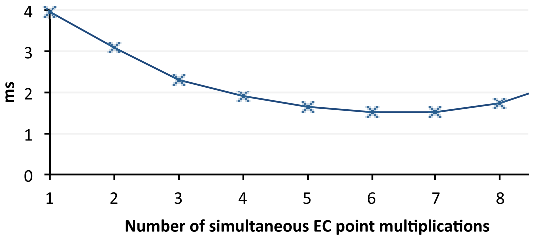


FIGURE 2. Computation time per EC point multiplication on CPU.

B. SIMULTANEOUS MULTIPLE EC POINT MULTIPLICATIONS

An optimization for simultaneous multiple EC point multiplications [31] was developed to speed up digital signature verification. If the optimization is utilized, the sum of more than two EC point multiplications, i.e., $\sum_{i=1}^{n} k_i \cdot P_i$, ($n \geq 2$, k_i: a scalar and P_i: an EC point) is calculated more quickly than when n EC point multiplications are independently calculated and added. For example, to compute $\sum_{i=1}^{3} k_i \cdot P_i$, the algorithm pre-computes all possible additions of points, i.e., $(P_1 + P_2)$, $(P_1 + P_3)$, $(P_2 + P_3)$ and $(P_1 + P_2 + P_3)$. Then, the algorithm sets the result point R to infinity $\mathcal{O}$. Finally, the bits of k_1, k_2 and k_3 are scanned from the most significant bit to the least significant bit. For each bit, R is doubled and the pre-computed points are added according to the bit value of k_i (e.g. if the bit of k_1 and the bit of k_3 are 1, then $(P_1 + P_3)$ is added to R). To measure the performance of this optimization technique, we utilized the MIRACL [32] ECC library and tested the technique on the CPU of the Nvidia Jetson TK1 developer kit [33]. Fig. 2 shows the computation time per EC point multiplication for calculating $\sum_{i=1}^{n} k_i \cdot P_i$, $(n = 1, 2, \ldots, 9)$ when secp160r1 is utilized for EC curve parameters. As shown in Fig. 2, when six EC points are simultaneously multiplied and added, the computation time per EC point multiplication is minimized. However, if more than six points are computed, the computation time begins to increase since the pre-computation overhead for all possible additions of points increases exponentially. $(2^n - 1 - n)$ pre-computations are required for $\sum_{i=1}^{n} k_i \cdot P_i$.

For this experiment, given a total number of EC point multiplications, we found the optimal combination of the numbers of simultaneous EC point multiplications. For instance, assume that a drone is required to compute $S = \sum_{i=1}^{9} k_i \cdot P_i$. If the drone computes $k_i \cdot P_i$ individually and adds them, it takes 35.7ms. If the drone runs the simultaneous multiple EC point multiplications on S, it takes 20.3ms. However, the time can be further reduced by properly dividing the number of simultaneous EC multiplications by dividing S into $S_1 = \sum_{i=1}^{4} k_i \cdot P_i$ and $S_2 = \sum_{i=5}^{10} k_i \cdot P_i$. Then, simultaneous multiple EC point multiplications are run on S_1 and S_2 separately, and then $S_1 + S_2$ is computed. The total computation time of such optimization technique is only 15.7ms. We utilized this optimization technique for our batch verification procedure.

IV. BUILDING BLOCKS

In this section, eCLSC-TKEM, CL-MRES, CLDA and the dual channel strategy are presented as major building blocks for our secure drone communication protocols. The formal security model and the security proofs of eCLSC-TKEM, CL-MRES and CLDA are provided in Appendices.

A. eCLSC-TKEM

eCLCS-TKEM meets all the security requirements, i.e., authenticated key agreement (AKA), non-repudiation and user revocation (see Table 1), while it minimizes the computational overhead at smart objects. Note that the CL-AKA protocols [12], [13] support only AKA. For non-repudiation, they must be extended with a digital signature scheme. Although the CLSC-TKEM protocols [14], [15] support AKA and non-repudiation, they do not support user revocation.

eCLSC-TKEM consists of 8 algorithms: (SetUp, SetSecretValue, PartialPrivateKeyExtract, SetPrivateKey, SetPublicKey, SymmetricKeyGen, Encapsulation Decapsulation). Each probabilistic polynomial time algorithm is as follows.

1) SETUP

The KGC generates the system parameters `params` Ω and a master private key `msk`, given a security parameter $k \in \mathbb{Z}^+$ as input. Given k, the KGC executes the following operations:

- Determines a k-bit prime q and the tuple $\{F_q, E/F_q, G_q, P\}$, where P is the generator of G_q.
- Chooses the master private key $x \in \mathbb{Z}_q^*$ uniformly at random and computes the system public key $P_{pub} = x \cdot P$.
- Chooses cryptographic hash functions $H_0 : \{0, 1\}^* \times G_q^2 \times \{0, 1\}^* \to \mathbb{Z}_q^**$, $H_1 : G_q^3 \times \{0, 1\}^* \times G_q \to \{0, 1\}^n$, $H_2 : G_q \times \{0, 1\}^* \times G_q \times \{0, 1\}^* \times G_q \times \{0, 1\}^* \times G_q \to \mathbb{Z}_q^*$, and $H_3 : G_q \times \{0, 1\}^* \times G_q \times \{0, 1\}^* \times G_q \times \{0, 1\}^* \times G_q \to \mathbb{Z}_q^*$. Here, n is the key length of a symmetric key encryption algorithm.
- 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 executed by each user. A user generates a secret value and the corresponding public value for oneself. The user A with its identity ID_A randomly chooses $x_A \in \mathbb{Z}_q^*$ as its secret value and computes the corresponding public key as $P_A = x_A \cdot P$.

3) PARTIALPRIVATEKEYEXTRACT

The KGC generates a partial private key for a user. This algorithm takes the KGC's master secret key, the id of the user ID_A, the public key of the user P_A and a permitted time period t_A as inputs. The user A sends (ID_A, P_A) to the KGC. In turn, the KGC generates and returns the partial private key of A as follows:

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

The partial private key of A is represented as d_A. The user A can validate d_A by determining if $d_A \cdot P = R_A + H_0(ID_A, R_A, P_A, t_A) \cdot P_{pub}$ holds.

4) SETPRIVATEKEY

Each user generates a full private key. The user A takes the pair (d_A, x_A) as its full private key sk_A.

5) SETPUBLICKEY

Each user generates a full public key. The user A takes the pair (P_A, R_A) as its full public key pk_A.

6) SYMMETRICKEYGEN

The sender A generates the symmetric key K and an internal state information Ω, which is not known to the receiver B. Given the sender (user A)'s identity ID_A, the full public key pk_A, the full private key sk_A, the receiver (user B)'s identity ID_B, the permitted time period t_B and the full public key pk_B as inputs, A performs the following steps to get the symmetric key K:

- Choose $s_A \in \mathbb{Z}_q^*$ and compute $V = s_A \cdot P$.
- Compute $Y = R_B + H_0(ID_B, R_B, P_B, t_B) \cdot P_{pub} + P_B$, $T = s_A \cdot Y\big(= s_A \cdot (H_0(ID_B, R_B, P_B, t_B) \cdot P_{pub} + R_B + P_B)\big)$ and $K = H_1(Y, V, T, ID_A, P_A, ID_B, P_B)$.
- Output K and the internal state information $\Omega = (s_A, V, T, ID_A, pk_A, sk_A, ID_B, pk_B, t_B)$.

7) ENCAPSULATION

The sender A obtains the encapsulation φ by taking Ω corresponding to K and a message M as inputs. Given Ω, K and M, the sender A executes the following two steps to get φ:

- **[Encryption step]** Compute $\tau = ENC_K(M)$.
- **[Sign step]** Choose $l_A \in \mathbb{Z}_q^*$ and compute $U = l_A \cdot P$,
  $H = H_2(U, \tau, V, ID_A, P_A, ID_B, P_B)$,
  $H' = H_3(U, \tau, V, ID_A, P_A, ID_B, P_B)$ and
  $W = d_A + l_A H + x_A H'$.

Output τ and $\varphi = (U, V, W)$.

8) DECAPSULATION

The receiver B decrypts τ using the key K encapsulated in φ. Given φ, τ, the sender's identity ID_A, full public key pk_A, the permitted time period t_A, the receiver's identity ID_B, the full public key pk_B and the full private key sk_B, B executes the following two steps to get K:

- **[Verification step]** Compute
  $H = H_2(U, \tau, V, ID_A, P_A, ID_B, P_B)$ and
  $H' = H_3(U, \tau, V, ID_A, P_A, ID_B, P_B)$.
  If $W \cdot P = R_A + H_0(ID_A, R_A, P_A, t_A) \cdot P_{pub} + H \cdot U + H' \cdot P_A$, perform the Decryption step. Otherwise, outputs an invalid encapsulation error. The correctness of the above equation is as follows:

$$\begin{aligned} &W \cdot P \\ &= (d_A + l_A \cdot H + x_A \cdot H') \cdot P \\ &= d_A \cdot P + l_A \cdot P \cdot H + x_A \cdot P \cdot H' \\ &= (r_A + xH_0(ID_A, R_A, P_A, t_A)) \cdot P + U \cdot H + H' \cdot P_A \\ &= R_A + H_0(ID_A, R_A, P_A, t_A) \cdot P_{pub} + H \cdot U + H' \cdot P_A \end{aligned}$$

- **[Decryption step]** Compute

$$\begin{aligned} T &= (d_B + x_B) \cdot V\big(= (d_B + x_B)s_A \cdot P = s_A \cdot Y\big), \\ Y &= (d_B + x_B) \cdot P\big(= (r_B + xH_0(ID_B, R_B, P_B, t_B) + x_B) \\ &\quad \cdot P = R_B + H_0(ID_B, R_B, P_B, t_B) \cdot P_{pub} + P_B\big), \\ K &= H_1(Y, V, T, ID_A, P_A, ID_B, P_B), \end{aligned}$$

  and $DEC_K(\tau)$ to obtain M.

B. CERTIFICATELESS HYBRID ENCRYPTION SCHEME (CLHES) FOR MULTI-RECEIVERS

If we remove the Sign operation from the Encapsulation phase and the Verification operation from the Decapsulation phase in the eCLSC-TKEM scheme, we can construct a certificateless hybrid encryption scheme (CLHES). Such CLHES consists of the following algorithms: (SetUp, KeyGen, HybridEncryption, HybridDecryption). As KeyGen algorithm generates a pair of a certificateless full public key and a full private key, it consists of the following algorithms: SetSecretValue, PartialPrivateKeyExtract, SetPrivateKey and SetPublicKey. Except for the HybridEncryption and HybridDecryption algorithms, all the algorithms are the same as the algorithms of eCLSC-TKEM. The HybridEncryption algorithm consists of the SymmetricKeyGen algorithm and the Encryption operation of Encapsulation algorithm of eCLSC-TKEM. The HybridDecryption algorithm consists of the Decryption operation of Decapsulation algorithm of eCLSC-TKEM. Moreover, CLHES can be extended into a certificateless multi-recipient encryption scheme (CL-MRES) by applying the random re-use (RR) technique, because CLHES is reproducible (see Appendix B). This CL-MRES is more effective than a naive method that individually encrypts messages using CLHES for one-to-many applications for several reasons. First, it results in bandwidth reduction, since the transmission of ciphertexts only requires half of the normal bits computed by the naive method, when ciphertexts are being broadcast or multi-cast by a sender. Second, the suggested scheme reduces about 50% of the the number of EC point multiplications for HybridEncryption as compared to the naive method. In CL-MRES, the hybrid decryption algorithm is identical to ordinary CLHES. The only difference between CLHES and CL-MRES is that the sender's random number s_A gets re-used to generate each recipient's symmetric key K_i $(1 \leq i \leq n)$. Thus, in this section we will describe only the HybridEncryption and HybridDecryption algorithms for multi-receivers.

1) HYBRIDENCRYPTION

Given public parameters Ω, a list $L = \{ID_{B_1}, \ldots, ID_{B_n}\}$ of the receiver identities, the receivers' time intervals t_{B_i} and full public keys pk_{B_i} $(1 \leq i \leq n)$ as inputs, the sender A executes the following steps to obtain the symmetric keys $K_i (1 \leq i \leq n)$ and encrypt the messages $M_i (1 \leq i \leq n)$ as follows:

- Choose $s_A \in \mathbb{Z}_q^*$ uniformly at random and compute $V = s_A \cdot P$.
- Repeat the following steps for all $ID_{B_i} \in L$, $i = 1, 2, \ldots, n$.
    1) Parse pk_{B_i} as (R_{B_i}, P_{B_i}) and t_{B_i}.
    2) Compute $Y_i = R_{B_i} + H_0(ID_{B_i}, R_{B_i}, P_{B_i}, t_{B_i}) \cdot P_{pub} + P_{B_i}$, $T_i = s_A \cdot Y_i$ and $K_i = H_1(Y_i, V, T_i, ID_A, P_A, ID_{B_i}, P_{B_i})$.
    3) Perform the symmetric encryption scheme to encrypt each message M_i for each receiver B_i. That is $\tau_i = ENC_{K_i}(M_i)$.
- Output $(V, \tau_1, \tau_2, \ldots, \tau_n)$.

2) HYBRIDDECRYPTION

Given ciphertexts $(V, \tau_1, \tau_2, \ldots, \tau_n)$, a list $L = \{ID_{B_1}, \ldots, ID_{B_n}\}$ of the receiver identities, the receivers' time intervals t_{B_i}, the full public keys pk_{B_i} and the full private keys sk_{B_i} $(1 \leq i \leq n)$ as inputs, each receiver B_i computes K_i and decrypts τ_i as follows:

- Compute $T_i = (d_{B_i} + x_{B_i}) \cdot V$ $\big(= (d_{B_i} + x_{B_i}) s_A \cdot P = s_A \cdot Y_i \big)$.
- Compute $Y_i = (d_{B_i} + x_{B_i}) \cdot P$ $\big(= (r_{B_i} + xH_0(ID_{B_i}, R_{B_i}, P_{B_i}, t_{B_i}) + x_{B_i}) \cdot P = R_{B_i} + H_0(ID_{B_i}, R_{B_i}, P_{B_i}, t_{B_i}) \cdot P_{pub} + P_{B_i} \big)$.
- Compute $K_i = H_1(Y_i, V, T_i, ID_A, P_A, ID_{B_i}, P_{B_i})$ and $DEC_{K_i}(\tau_i)$ to obtain M_i.

C. CERTIFICATELESS DATA AGGREGATION (CLDA)

In this section, we show an efficient aggregation protocol with which a drone A collects sensor values from authenticated smart objects and transfers their aggregate sum to an authenticated base station B in an efficient way. This is accomplished by combining EC-Elgamal additive homomorphic encryption scheme [26] with our certificateless approach. Let the full public and private key of B be (P_B, R_B) and (d_B, x_B), respectively. The full public and private of each smart object i are (P_i, R_i) and (d_i, x_i), respectively, where $1 \leq i \leq n$. Let O_i denote the data of i where $O_i \in G_q$. We assume that mapping actual sensor values into elliptic curve points O_i and vice-versa is easy since the range of the sensed data values is limited.

1) SENSOR DATA ENCRYPTION

This algorithm is executed by each smart object i. Given the base station's identity ID_B, the full public key pk_B and the time interval t_B as inputs, each smart object executes the following steps:

- Chooses $l_i, s_i \in \mathbb{Z}_q^*$ and computes $U_i = l_i \cdot P$, $V_i = s_i \cdot P$.
- Computes $T_i = s_i(R_B + H_0(ID_B, R_B, P_B, t_B) \cdot P_{pub} + P_B)$ and $C_i = T_i + O_i$.
- Computes $H_i = H_4(U_i, C_i, V_i, ID_B, P_B, ID_i, P_i)$, $H_i' = H_5(U_i, C_i, V_i, ID_B, P_B, ID_i, P_i)$ and $\sigma_i = d_i + l_i H_i + x_i H_i'$.
- Sends $\psi_i = (U_i, V_i, C_i, \sigma_i, ID_i, P_i, R_i, t_i)$ to A.

2) BATCH VERIFIATION

This algorithm is executed by the drone A. Given the base station's identity ID_B, the full public key pk_B, the time interval t_B, and ψ_i as inputs, A executes the following steps:

- Computes $H_i = H_4(C_i, U_i, ID_B, P_B, ID_i, P_i)$ and $H_i' = H_5(C_i, U_i, ID_B, P_B, ID_i, P_i)$.
- If $(\sum_{i=1}^{n} \sigma_i) \cdot P = \sum_{i=1}^{n} \big(R_i + H_0(ID_i, R_i, P_i, t_i) P_{pub}\big) + \sum_{i=1}^{n} H_i \cdot U_i + \sum_{i=1}^{n} H_i' \cdot P_i$, goes to the next step. Otherwise, outputs a verification failure error and verifies them individually. The correctness of the above equation is as follows:

$$
\begin{aligned}
(\sum_{i=1}^{n} \sigma_i) \cdot P &= \Big(\sum_{i=1}^{n} (d_i + l_i H_i + x_i H_i') \Big) \cdot P \\
&= \sum_{i=1}^{n} d_i \cdot P + \sum_{i=1}^{n} l_i H_i \cdot P + \sum_{i=1}^{n} x_i H_i' \cdot P \\
&= \sum_{i=1}^{n} \big(R_i + H_0(ID_i, R_i, P_i, t_i) \cdot P_{pub} \big) \\
&\quad + \sum_{i=1}^{n} H_i \cdot U_i + \sum_{i=1}^{n} H_i' \cdot P_i
\end{aligned}
$$

- After the verification, the drone A sends a *success* or *failure* message to C_i.

Note that the privacy of each smart object is preserved since A cannot decrypt C_i. However, A can confirm that C_i is sent by an authenticated smart object i. The batch verification reduces the number of time-consuming EC point multiplications from $4n$ to $3n + 1$.

3) DATA AGGREGATION

This algorithm is executed by A. A computes $C = \sum_{i=1}^{n} C_i$ and $V = \sum_{i=1}^{n} V_i$ and deletes C_i and V_i $(1 \leq i \leq n)$ Then, A sends (C, V) to the base station B.

4) AGGREGATE SUM DECRYPTION

This algorithm is executed by B. Given (C, V) and the B's full private key sk_B, B can obtain the aggregate sum O by computing $O = C - (d_B + x_B) \cdot V$.

- The correctness of the equation is as follows:

$$
\begin{aligned}
O &= \sum_{i=1}^{n} C_i - (d_B + x_B) \sum_{i=1}^{n} V_i \\
&= \sum_{i=1}^{n} (T_i + O_i)
\end{aligned}
$$

$$
\begin{aligned}
& s_i \sum_{i=1}^{n} (R_B + H_0(ID_B, R_B, P_B, t_B) \cdot P_{pub} + P_B) \\
&= \sum_{i=1}^{n} (T_i + O_i) - \sum_{i=1}^{n} T_i \\
&= \sum_{i=1}^{n} O_i
\end{aligned}
$$

Since B only obtains the aggregate sum, the privacy of each smart object is preserved.

D. DUAL CHANNEL STRATEGY FOR CONCURRENCY USING LPL

Smart objects and drones must be operated in energy-efficient ways because they are usually battery-powered. To save their energy, we adopt low power listening (LPL) for smart objects and dual channels for drones. LPL [34] is an asynchronous duty cycling technique commonly used in WSNs and can significantly save sensor energy by reducing idle listening time.

A drone has two radios operated in different channels, i.e., the wake-up channel and the data channel. Each smart object has only one radio and switches between the two channels according to the need. As shown in Fig. 3, a smart object runs LPL, i.e., periodically turns its radio on (wake-up) and off (sleep) in the wake-up channel. When a smart object wakes up, it quickly checks the wake-up channel to see if it is busy. If it is not, the smart object sleeps again until the next wake-up time to save energy. A mobile drone continuously broadcasts wake-up signals using the radio in the wake-up channel. If the drone approaches the smart object, the wake-up channel around the smart object becomes busy due to wake-up signals broadcast by the drone. If the smart object listens a portion of a wake-up signal, it stays awake to receive a whole wake-up signal. For the drone to efficiently run eCLSC-TKEM with a set of smart objects, each smart object concurrently executes SymmetricKeyGen and Encapsulation after receiving the wake-up signal. Then, the smart object switches its radio channel from the wake-up channel to the data channel. Each smart object sends the Encapsulation output to the drone through the data channel. These concurrent executions of eCLSC-TKEM using the dual channels can conserve the drone's energy. If the drone had only one radio, it would either have to make precise schedules with the smart objects using a time synchronization procedure, or it would have to perform all of the eCLSC-TKEM steps with each smart object at a time. This would be a waste of the drone's flight time.

Obviously, operating two radio transceivers requires more energy than operating one radio transceiver. However, the energy consumed by a radio transceiver is negligible considering that the power to let a drone fly is five orders of magnitude greater than the power to operate a radio transceiver.[1] Therefore, the energy saved by running the dual channel strategy using the two radios overwhelms the energy increased by operating one more radio.

V. SMART TRAFFIC AND PARKING MANAGEMENT PROTOCOL FOR SMART CITY

In this section, we present how our protocols are used for a smart traffic and parking management application.

A. CAR REGISTRATION

We assume that a government or an institute provides each car owner with a smart object that is a low-end embedded device with a radio transceiver and a GPS. The smart object (A) executes the SetSecretValue algorithm to generate its own secret value (x_A) and the public key (P_A). The KGC runs the PartialPrivateKeyExtract algorithm to generate a partial private/public key pair (d_A, R_A) for A and transfers the pair to A through a secure channel. Notice that the partial private key expires after a permitted time period t_A, e.g., one year. Hence, a car owner must obtain a new partial private/public pair before it expires. The smart object is attached to the car.

We assume that a drone stays in a secure place when it is off duty. The drone (B) runs the SetSecretValue algorithm to generate its secret value (x_B) and the public key (P_B). Before the drone is dispatched for a mission, it obtains a partial private/public key (d_B, R_B) from the KGC. The permitted time period t_B should be set to as short as possible, e.g., the drone's maximum flight time, so that even if the drone is compromised, the malicious use of the compromised partial private key is limited to this time period. The KGC can give appropriate access rights to the drone as a part of ID_B. For instance, ID_B can be $\{id_B||read||write||permitted\_zones\}$ so that the drone can read data from smart objects and reconfigure (write) the settings of smart objects which are located within the permitted zones.

B. PARKING MANAGEMENT

Today's parking management is labor-intensive and inefficient. Parking enforcement officers patrol on-street parking zones by periods and check each car to see if it has violated the parking time limit. This process can be made more efficient by automating it with the use of drones and smart objects. For example, a university may provide each registered car owner with a smart object which include a radio transceiver and a GPS, and a function as a parking permit for campus parking management. In this case, a drone would patrol the campus and collect data from every parked car. The data would include the identity of a car, the parking permit type, the current time and location. By gathering these data at regular intervals, the drone would be able to determine if cars are illegally parked. E.g., the drone could see if a car has been parked at an on-street parking area for longer than the time permitted.

In this scenario, since all the data collected by the drone are privacy-sensitive, they must be encrypted and collected by only authorized drones. More to the point, the data sent by

[1] The power consumption of DJI S1000 (drone) during flight is from 1,500W to 4,000W, while the TX power of CC2420 is 52mW.

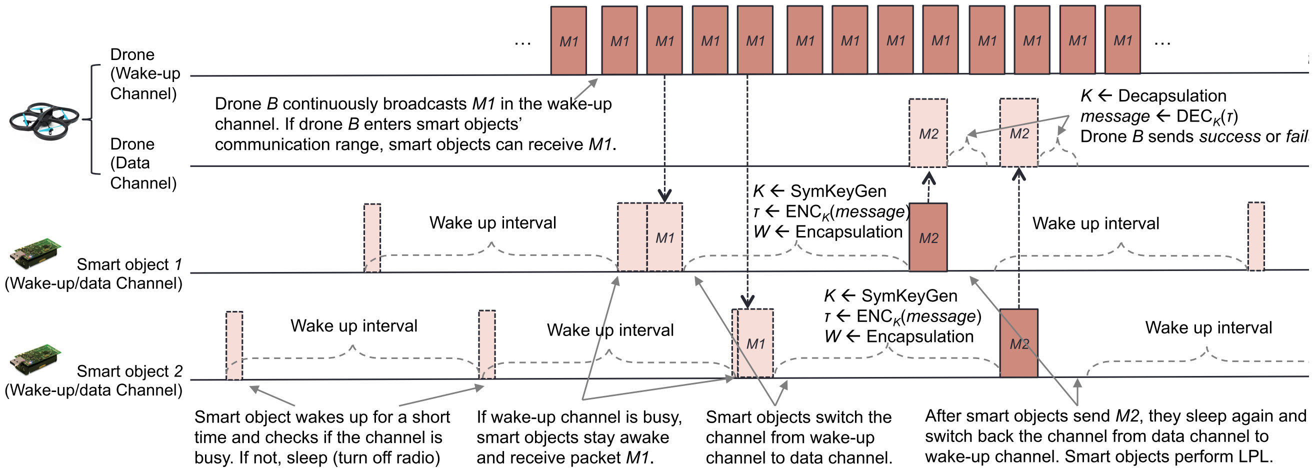

M1
Drone (Wake-up Channel)
Drone (Data Channel)
Drone B continuously broadcasts M1 in the wake-up channel. If drone B enters smart objects' communication range, smart objects can receive M1.
K ← Decapsulation
message ← DEC_K(τ)
Drone B sends success or fail
M2
Smart object 1 (Wake-up/data Channel)
Smart object 2 (Wake-up/data Channel)
Wake up interval
K ← SymKeyGen
τ ← ENC_K(message)
W ← Encapsulation
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 3. Smart parking management. Solid-line rectangle: transmitted message, dash-line rectangle: received message. $M1 = \{ID_B, P_B, R_B, t_B\}$, $M2 = \{ID_A, P_A, R_A, t_A, U, V, W, \tau\}$, Decapsulation result transmissions are omitted.

the cars must not be modified and repudiated afterwards since the data are used to fine the car owners who have illegally parked their cars.

1) PROTOCOL DESCRIPTION

Fig. 3 shows how eCLSC-TKEM and the dual channel strategy work for our smart parking management. Each smart object has one radio transceiver, while a drone (B) has two radio transceivers working in different channels, i.e., the wake-up channel and the data channel. A smart object (A) executes LPL in the wake-up channel. The drone's radio operated in the wake-up channel continuously broadcasts wake-up signals ($M1$) so that awake smart objects can detect $M1$ as the drone approaches. $M1$ consists of the drone's ID (ID_B), its public keys (P_B, R_B) and its permitted time period (t_B).

After receiving $M1$, a smart object suspends LPL and executes SymmetricKeyGen to generate a symmetric key K. The smart object creates a message $\mathcal{M}$ containing its permit type, its current location (loc) and its current time (ct), and obtains its ciphertext τ ($= ENC_K(\mathcal{M})$).

Then, the smart object (A) executes Encapsulation to generate W. A changes its radio channel from the wake-up channel to the data channel and sends $M2$ to B. $M2$ consists of the smart object's ID (ID_A), the public keys (P_A, R_A), the permitted time period (t_A), the ephemeral public keys (U, V), the Encapsulation output (W), and τ. Since A digitally signs τ in the Encapsulation algorithm, A cannot deny having sent τ.

After receiving $M2$ using the radio operated in the data channel, the drone B runs Decapsulation. If the validation check is passed, B decrypts τ after generating K. Then, B compares loc and ct with its own current location loc' and current time ct', respectively. If the validation check fails or the comparison outcome is abnormal, B takes additional actions. For instance, if $|loc' - loc| > 10m$ or $|ct' - ct| > 1$ min, B can take a photo of the car or send a message to a human manager. Finally, B sends an acknowledgement stating the decapsulation result (success or failure) to A. If all the decapsulation steps are successfully completed, K can be used to encrypt more messages exchanged between A and B .

2) SECURITY ANALYSIS

The parking management based on eCLSC-TKEM meets all the security requirements described in Fig. 1 as follows:

- *Confidentiality and integrity*: eCLSC-TKEM ensures the confidentiality of messages, i.e., indistinguishability against an adaptive chosen ciphertext and identity attacks (IND-CCA2) based on Theorem 1 in Appendix A. Theorem 2 in Appendix A supports that eCLSC-TKEM guarantees the integrity of the messages, i.e., existential unforgeability against adaptive chosen messages and identity attacks (EUF-CMA).
- *Authenticated key agreement*: The drone and the smart object can be authenticated by each other. Only when they have the valid full private/public keys, they can correctly generate a shared symmetric key K, and thus they can mutually be authenticated.
- *User revocation*: The KGC inserts a permitted time period in t_i when it generates the partial private key d_i for each entity i. Therefore, after the time period, d_i is automatically revoked. Each entity is responsible to periodically renew its d_i and R_i to correctly run the protocols. This property is applied to our other protocols too, i.e., CL-MRES and CLDA.
- *Non-repudiation*: The smart object (A) cannot repudiate a message τ since τ is digitally signed using A's full private key in the Encapsulation step.

C. TRAFFIC MONITORING AND MANAGEMENT

Modern city traffic monitoring systems utilize fixed sensors such as cameras or inductive loops which are installed on roads at regular intervals or at important locations such as intersections or interchanges. Due to the high installation cost, they can observe traffic only at selected areas. Since the locations of such sensors are fixed, the system cannot respond to exceptional events such as holiday traffic or car accidents that happen in areas where the sensors are not installed.

However, if every car has a sensor with a network interface, we can make a system by which drones can collect traffic information from cars. Such a system can provide more flexible, accurate and find-grained traffic information than traditional traffic monitoring systems. Imagine drone operating companies that collect data using drones and provision city agencies with such data. In this scenario, since privacy-sensitive data such as speed, acceleration and the number of passengers can be collected, the data must be encrypted and only an authorized base station is allowed to read the data. Moreover, drones must collect the data from only authenticated cars to prevent statistics from being tampered by malicious parties. To assure the privacy of each car, the base station is allowed to get only the aggregate sum of the values. Since a drone very often collects data from large numbers of cars, the collection procedure must be efficient in terms of storage, communication and computation. To satisfy such requirements in many-to-one communication scenarios, we utilize CLDA.

In addition, a drone may need to send private messages to hundreds of cars in a short time period. For example, the drone may send the information about the traffic at each car's destination or provide subscription-based information service for each car. To efficiently encrypt such messages and sign them in one-to-many communication scenarios, we utilize CL-MRES.

1) PROTOCOL DESCRIPTION

Fig. 4 shows the flow of the data collection procedure using CLDA and the dual channel strategy in our traffic monitoring and management. A drone (A) continuously broadcasts wake-up signals ($M1 = \{ID_B, P_B, R_B, t_B\}$, i.e., the public information of the base station B) in the wake-up channel while it moves. Cars (smart objects) run LPL in the wake-up channel and try to detect wake-up signals from a drone. Once a car detects a wake-up signal, it executes the Sensor data encryption protocol and sends $M2 = \{U_i, V_i, C_i, \sigma_i\}$ to the drone through the data channel. If the number of $M2$ messages received from cars becomes larger than a certain threshold, the drone runs the Batch verification protocol to check the authenticity and integrity of the data. If the verification procedure is passed, the drone sends *success* messages to cars. If not, the drone verifies each $M2$ message individually. Then, it runs the Data aggregation on the collected data in order to reduce the required storage space and the communication overhead for sending the collected data to the base station. Also, the drone can save its computation resources since it does not need to decrypt the data. Only computationally cheap EC point additions are required by the Data aggregation protocol. The drone deletes all $M2$ messages after the completion of the aggregation procedure.

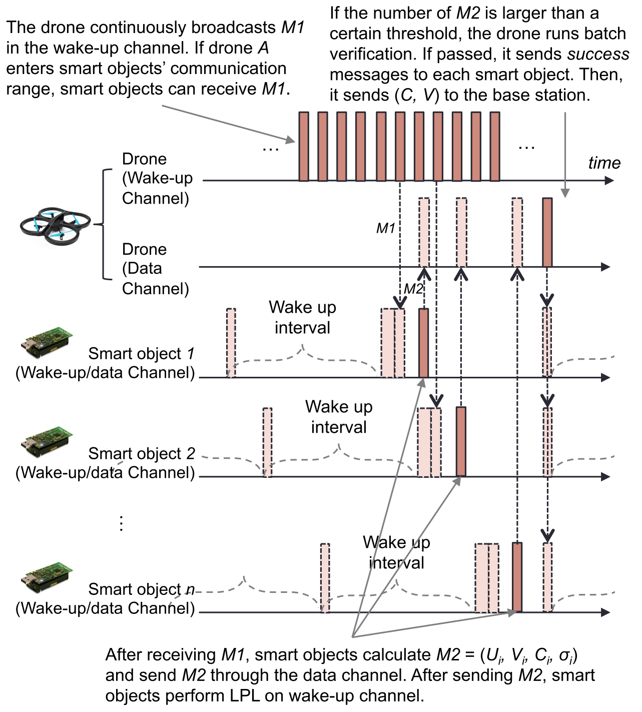

The drone continuously broadcasts *M1* in the wake-up channel. If drone *A* enters smart objects' communication range, smart objects can receive *M1*.
If the number of *M2* is larger than a certain threshold, the drone runs batch verification. If passed, it sends *success* messages to each smart object. Then, it sends (*C*, *V*) to the base station.
Drone (Wake-up Channel)
Drone (Data Channel)
time
M1
M2
Wake up interval
Smart object 1 (Wake-up/data Channel)
Wake up interval
Smart object 2 (Wake-up/data Channel)
Wake up interval
Smart object n (Wake-up/data Channel)
After receiving *M1*, smart objects calculate *M2* = (*U_i*, *V_i*, *C_i*, *σ_i*) and send *M2* through the data channel. After sending *M2*, smart objects perform LPL on wake-up channel.

FIGURE 4. Traffic monitoring (CLDA). Solid-line rectangle: transmitted message, dash-line rectangle: received message. $M1 = \{ID_B, P_B, R_B, t_B\}$. $M2 = \{U_i, V_i, C_i, \sigma_i\}$.

After the drone finishes collecting data, it transfers the C and V to the base station B runs the Aggregate sum decryption algorithm to obtain the aggregate sum.

In a real application, it is crucial to keep the time for the batch verification short since the verification time can be a bottleneck of this protocol. The drone as a mobile data collector might have to collect data from hundreds of smart objects in a very short time while it flies. If the arrival rate of the $M2$ messages is higher than the verification speed of the drone, the storage of the drone might be flooded and new arriving $M2$ messages might be dropped. If $M2$ messages begin to be dropped, the drone should stop and collect again the lost $M2$ messages, which consumes the drone's battery. Therefore, the number (θ) of $M2$ messages that are verified together must be large since the Batch verification algorithm reduces the number of EC point multiplications to be computed, and thus speeds up the verification procedure.

However, when $M2$ messages sparsely arrive, if θ is set to too large, the drone cannot execute the Batch verification algorithm until the number of $M2$ messages becomes θ, which delays the verification procedure. In addition, smart objects might consume their energy since they cannot sleep until they receive the result of the Batch verification algorithm. In this case, it would be better to verify each $M2$ message individually rather than to verify the messages in batches. Therefore, θ must be set adaptively according to the arrival rate of the $M2$ messages. When the drone needs to verify $M2$ messages individually, two strategies are possible: 1) verifying $M2$ messages one-by-one, and 2) launching new threads whenever $M2$ messages are received for each verification. We only consider the first strategy since the second strategy increases the average response time for the smart objects compared to the first strategy.

To send privacy-sensitive messages to hundreds of cars, a drone must encrypt the messages with individual keys in an efficient way. CL-MRES reduces the computation time on the drone the random re-use (RR) technique. Thus, we utilize CL-MRES to efficiently encrypt messages in the traffic monitoring and management. The drone encrypts each message using the HybridEncryption algorithm in CL-MRES. and sends $(V, \tau_1, \tau_2, \ldots, \tau_i, \ldots, \tau_n)$. Each car (i) decrypts the encrypted message (τ_i) using the HybridDecryption algorithm.

2) SECURITY ANALYSIS

The traffic monitoring and management based CLDA and CL-MRES meet all the security requirements described in Fig. 1.

- *Confidentiality and integrity*: CLDA is a variant signed ElGamal encryption [35] combining EC-ElGamal encryption with the signing function of our eCLSC-TKEM. The output message of the Sensor data encryption step in CLDA is an EC-ElGamal ciphertext together with the eCLSC-TKEM-based signature of that ciphertext. So, the security of CLDA is based on the unforgeability of eCLSC-TKEM and the confidentiality of the EC-ElGamal encryption. Here, the confidentiality is defined as indistinguishability under chosen plaintext attacks (IND-CPA) while unforgeability is defined as existential unforgeability against adaptive chosen messages and identity attacks (EUF- CMA). The confidentiality and integrity of CL-MRES are supported by Theorem 3 in Appendix B.
- *Authentication*: In CLDA, the drone can explicitly authenticate cars by verifying the signatures in the Batch verification step. The cars can implicitly authenticate the base station by using the full public key of the base station in the Sensor data encryption step. Only when the base station has the valid full private key, it can decrypt the encrypted data. In CL-MRES, only when the drone and a car i have the valid full private/public keys, they can correctly generate a shared symmetric key K_i and thus, they can mutually be authenticated. That is, CL-MRES supports authenticated key agreement.
- *Privacy protection*: In the CLDA protocol, each car encrypts sensor values using the full public key of the base station B and the drone does not carry the full private key of B. Therefore, the collected values are secure even if the drone is captured and the content of its internal memory is analyzed by an attacker. Since the drone homomorphically aggregates the encrypted data and deletes all $M2$ messages right after the completion of the aggregation procedure, the base station can get only the aggregate sum of the values. Therefore, the privacy of each smart object is assured under the assumption that the drone and the base station do not collude.

VI. EXPERIMENTS

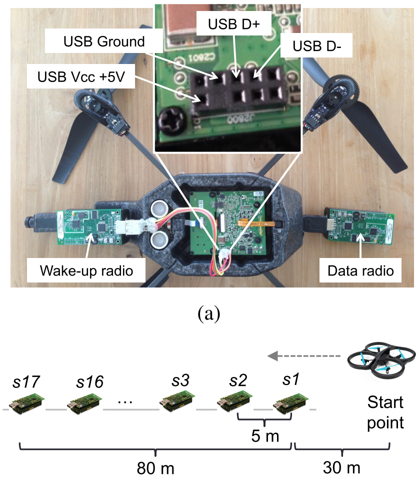
In this section, we present the performance of the eCLSM-TKEM, CL-MRES and CLDA protocols and how our efficiency enhancement techniques improve the performance.

A. EXPERIMENT SETUP OF THE PARKING MANAGEMENT

To evaluate the performance of eCLSC-TKEM, we implemented our protocols on commercially available devices: AR.Drone2.0 [36] (as a medium-capacity drone) and TelosBs (as smart objects). To compare eCLSC-TKEM with other certificateless-based schemes, we also implemented CL-AKA [12], [13] and CLSC-TKEM [15].

1) DRONE

AR.Drone2.0 [36] is a quad-copter equipped with a Wi-Fi radio, two (front/ground) cameras, an ARM cortex A8 processor (1GHz/32-bit) and an 1Gbit RAM. The operating system of AR.Drone2.0 is the BusyBox-based Linux (ver. 2.6.32). After booting up, the drone acts as a Wi-Fi access point and can be controlled by a remote Wi-Fi client, such as a laptop or a smartphone. We utilized the MIRACL library [32] as a crypto-library.



(a)

(b)

FIGURE 5. Experiment setup. (a) Dual radios attached on the drone. (b) Test-bed setup.

We utilized two TelosBs as the drone's radio transceivers as shown in Fig. 5(a). One radio working in the data channel was plugged into the USB port next to the battery. The other radio working in the wake-up channel was hooked up to a pin connector on the main board. The radio transceiver of TelosB works at the 2.4GHz public band, which is the same band at which the Wi-Fi works. To avoid interference between them,

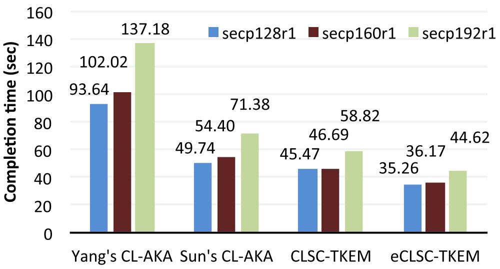


FIGURE 6. Impact of key bit size.

we chose the channel 6 as the Wi-Fi control channel, and the channel 11 and 26 as the wake-up channel and the data channel, respectively.

2) SMART OBJECT

For smart objects, we utilized 17 TelosBs. TelosB is a sensor platform equipped with an IEEE 802.15.4 radio transceiver, a low-end micro-controller (8MHz MSP430) with a 10KB RAM and a USB interface. We chose TelosBs as the smart objects in order to show that even such low-end platforms run our protocols well. The signal power of the smart objects was set to -7dBm and the communication range was approximately 30m. We installed TinyOS 2.0 for the operating system and utilized its LPL functionality. We also used TinyECC [37] as an elliptic curve cryptography library.

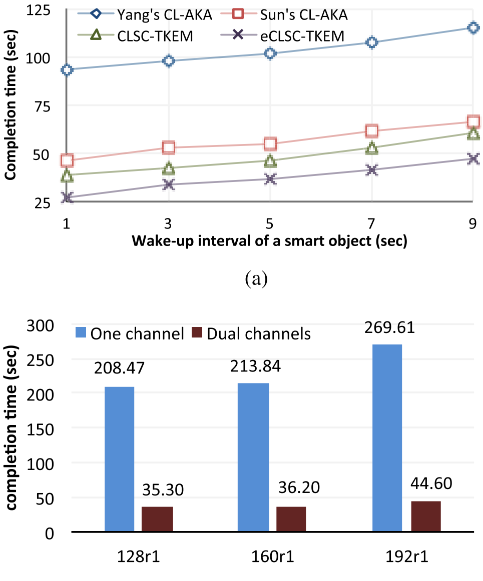
B. EXPERIMENTAL RESULTS OF THE PARKING MANAGEMENT

1) NETWORK TOPOLOGY

Fig. 5(b) shows the test-bed setup of our parking management system. We deployed the 17 smart objects in a line spacing them 5m apart and the drone started from the start point that was 30m apart from $s1$. The drone's altitude was set to 10m. In our test-bed, the drone's mission is to collect data from all the smart objects. The drone flies from the start point to the last smart object $s17$. When the drone arrives at a smart object sx, if the drone cannot complete the data collection task with sx, the drone maintains its present position until the task is completed. We measured the time to complete the mission.

2) IMPACT OF KEY BIT SIZE

Fig. 6 shows the time required to complete the mission when the system used three different key bit sizes. We activated LPL for the smart objects with the wake-up interval of 5 seconds. When secp160r1 was used, the drone with our protocol took 36.2 seconds to complete the mission and completed the mission 1.3, 1.5 and 2.8 times faster than Seo's CLSC-TKEM, Sun's CL-AKA and Yang's CL-AKA, respectively. All the protocols require more time to complete the mission if the key bit size becomes larger. However, the time 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 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. Table 2 shows the computation time required by a smart object when the four protocols with the three different elliptic curves are used. When secp160r1 is used, the smart object with our protocol can complete its task 1.4, 1.8 and 3.8 times faster than the smart object with Seo's CLSC-TKEM, Sun's CL-AKA and Yang's CL-AKA, respectively. Considering that overall system performance highly depends on the performance of low speed devices in a heterogenous system, the results in Table 2 explain why our protocol outperformed the others. Note that a smart object using our protocol needs to compute only two EC point multiplications after it receives a wake-up signal from a drone, while a smart object using the other protocols has to compute more than two EC point multiplications.



(a)

(b)

FIGURE 7. Experimental results. (a) Impact of interval between wake-ups. (b) Impact of the dual channel strategy.

TABLE 2. Comparison of protocols (unit: second).

Protocol	secp128r1	secp160r1	secp192r1
Yang's CL-AKA [12]	32.84	36.22	50.43
Sun's CL-AKA [13]	15.10	16.98	23.84
Seo's CLSC-TKEM [15]	13.37	13.87	18.77
eCLSC-TKEM [1]	9.25	9.61	13.03

3) IMPACT OF INTERVAL BETWEEN WAKE-UPS

Fig. 7(a) shows the time required to complete the mission when the smart objects adopt five different LPL wake-up intervals. We used secp160r1. The wider the interval between wake-ups is, the more energy the smart objects can save. Seo's CLSC-TKEM and Sun's CL-AKA require a narrow interval to achieve a mission completion time close to the completion time achieved by our protocol. For example, Sun's CL-AKA achieved a mission completion time of 46.2 seconds when the wake-up interval was set to 1 second, while our protocol achieved a close mission completion time (46.8 seconds) when the wake-up interval was set to 9 seconds. Seo's CLSC-TKEM achieved a close mission completion time (46.7 seconds) when the wake-up interval was set to 5 second. In other words, when our protocol was used, the smart objects consumed 1.8 and 9 times less energy than when Seo's CLSC-TKEM or Sun's CL-AKA was used, respectively.

4) IMPACT OF DUAL CHANNEL STRATEGY

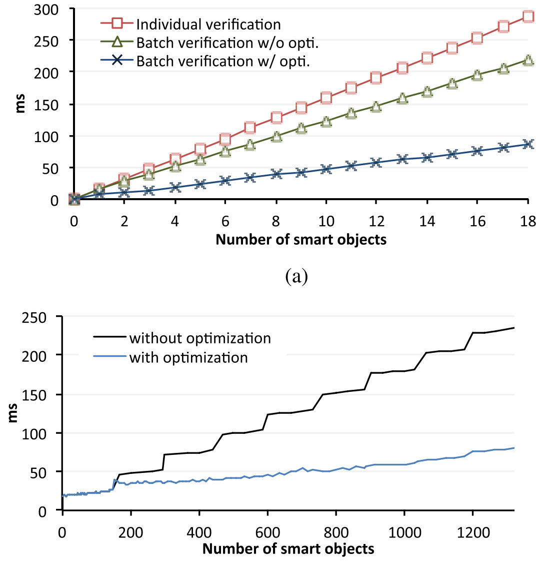
Finally, Fig. 7(b) shows the mission completion time when the dual channel strategy is used and when it is not used.. We utilized eCLSC-TKEM and set the wake-up interval to 5 seconds. When the system utilized the dual channel strategy, the mission was completed approximately 6 times faster than when only one channel was used. The dual channel strategy allows smart objects to concurrently execute eCLSC-TKEM with a drone, and thus the drone's flight time can be significantly saved. When the dual channel strategy was used, 3 or 4 smart objects within the communication range of the drone were able to concurrently start executing the eCLSC-TKEM protocol. However, if only one channel is used, the drone must execute eCLSC-TKEM with smart objects one by one. In the one-channel system, the drone broadcasts wake-up signals while moving. Once a smart object receives a wake-up signal, it sends an acknowledgement to the drone. Then, the drone must stop broadcasting wake-up signals in order to listen and receive the eCLSC-TKEM output (i.e., an encrypted message with its signature) from the smart object. However, since the smart object generates the eCLSC-TKEM output very slowly, the drone must wait, which wastes its limited flight time. After all the procedures of eCLSC-TKEM are successfully completed with the smart object, the drone can start broadcasting wake-up signals again in order to wake up another smart object. To sum up, the dual channel strategy is essential in order to save the energy of a mobile drone when the drone runs a cryptographic protocol with multiple low-end devices.

C. EXPERIMENT SETUP OF THE TRAFFIC MONITORING AND MANAGEMENT

To evaluate the performance of CLDA and CL-MRES, we implemented these schemes on the Nvidia Jetson TK1 developer kit [33] as a high-capacity drone. The kit is operated by Ubuntu Linux and is equipped with the Tegra K1 SoC which consists of a 2.3 GHz ARM Cortex-A15 CPU and 0.85 GHz NVIDIA Kepler GPU with 192 CUDA Cores. We chose this kit because the GPU in the Tegra K1 SoC is the only mobile GPU to support NVIDIA CUDA. We ported the functions in the MIRACL library [32] into CUDA-C functions in order to run them on the GPU. We utilized secp160r1 as an ECC parameters.

1) EXPERIMENTAL RESULTS OF CLDA

The overall performance of the CLDA protocol is dominated by the performance of the Batch verification algorithm. Therefore, we measured the execution time of the Batch verification algorithm on the CPU and the GPU. The drone collects a random value from virtual cars which run on a PC. They execute the Sensor data encryption algorithm and send $\psi_i = (U_i, V_i, C_i, \sigma_i, ID_i, P_i, R_i, t_i)$ to the drone. We assume all cars send valid signatures.



(a)

(b)

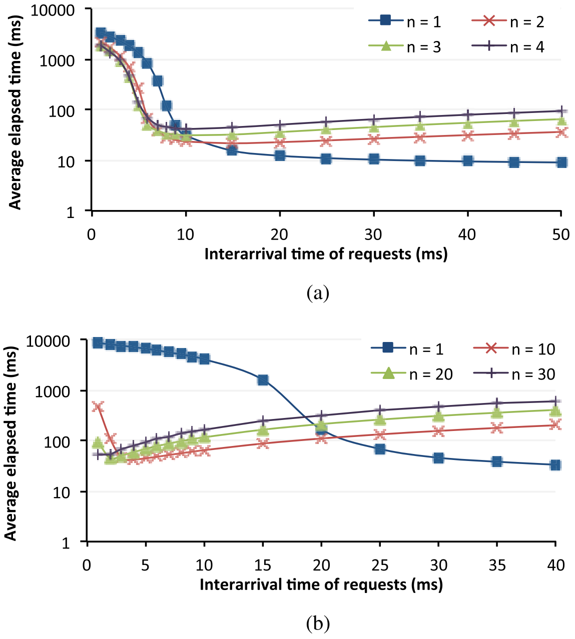
FIGURE 8. The computation time for the signature verification on the CPU or GPU. (a) CPU. (b) GPU.

In the first experiment, the drone executes the Batch verification algorithm on the CPU after all data are collected from n $(1 \leq n \leq 18)$ cars. We implemented three versions of the verification algorithm: 1) individual verification, 2) batch verification without optimization, and 3) batch verification with the optimization technique as described in Sec. III-B. Fig. 8(a) shows the computation time of the three versions of the protocol on the drone. When the number of cars is 18, the drone running the optimized batch verification requires only 85.5ms, while the drone running the individual verification and the drone running the batch verification without optimization require 285.8ms and 218.4ms, respectively. This result confirms that the batch verification with the optimization technique significantly reduces the computation time for the signature verifications. If the arrival rate of ψ_is is very high, the CPU is not appropriate to handle the ψ_is. In the second experiment, the Batch verification algorithm is run on the GPU after all data are collected from n $(1 \leq n \leq 1,320)$ cars. Two versions of the verification algorithm were implemented: 1) batch verification without optimization: each GPU thread computes one EC point multiplication, and 2) batch verification with the optimization technique: each GPU thread computes multiple EC point multiplications. We limit the maximum number of GPU threads that can be launched in parallel to 441 due to the limited GPU memory space. Therefore, in the first version, 147(=441/3) signatures are simultaneously verified using

441 threads in each cycle. However, in the second version, the number of EC point multiplications that are executed by each thread is selected according to the total number of EC point multiplications. For instance, if the total number of EC point multiplications is 441, each thread computes one EC point multiplication. However, if the total number of EC point multiplications is 882, each thread computes two EC point multiplications. As shown in Fig. 8(b), when the number of cars is 1,320, the optimized batch verification takes only 81.2ms, while the batch verification without optimization takes 233.9ms. This result shows that the time required by the batch signature verification is significantly reduced due to the optimization technique.

AVERAGE ELAPSED TIME OBSERVED BY CARS

As discussed in Sec. V-C, the drone has to adaptively set θ, i.e., the number of signatures that are verified together, according to the arrival rate of the signature verification requests. To measure the average elapsed time observed by cars, we developed a discrete-event simulator specialized for our protocol. Parameters for the simulations, such as the communication delay and the batch verification times on the CPU and the GPU, are based on the real measurements.



(a)

(b)

FIGURE 9. The average elapsed time observed by cars when the drone uses the CPU or GPU. (a) CPU. (b) GPU.

Fig. 9(a) shows the average elapsed time observed by cars when the drone uses the CPU. When θ is 1, the drone verifies signatures separately and cannot take advantage of the batch verification. Thus, if the inter-arrival time is small, the drone's CPU cannot handle signatures in a short time. However, as the mean inter-arrival time ($\mathcal{T}$) becomes large, the average elapsed time decreases since the time required for a single verification is smaller than $\mathcal{T}$. Thus, signatures are verified right after they arrive. When θ is n (≥ 2), the drone executes the batch verification once the drone receives n signature verification requests. Therefore, the drone can take advantage of the batch verification when $\mathcal{T}$ is small. However, as $\mathcal{T}$ becomes large, the average elapsed time increases since the drone cannot execute the **Batch verification** algorithm until n signature verification requests are collected.

Fig. 9(b) shows the average elapsed time when the GPU is utilized. Since the clock speed of the GPU is slower than the clock speed of the CPU, when $\mathcal{T}$ is large, verifying signatures using the CPU is faster than verifying signatures using the GPU. For example, when $\mathcal{T}$ is 50ms, the CPU can verify signatures in 9.1ms on average, while the GPU verifies them in 28.8ms. However, when $\mathcal{T}$ is small, the drone can utilize the parallel processing of the GPU. For instance, when $\mathcal{T}$ is 1ms, the CPU verifies signatures in 1,882ms, while the GPU can verify them in 51.5ms by setting θ to 30. Although we did not present all the results with different values of θs due to the page limit, if the drone can change θ in the optimal way, the average elapsed time is always kept lower than 47ms.

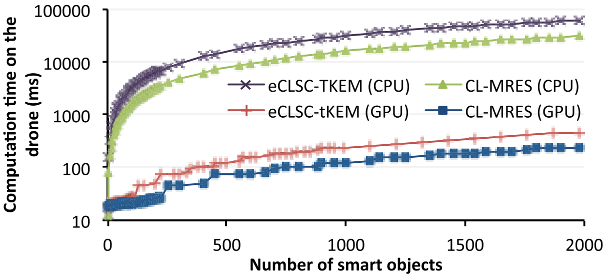


FIGURE 10. The performance comparison between eCLSC-TKEM and CL-MRES on the CPU and GPU.

2) EXPERIMENTAL RESULTS FOR CL-MRES

Since the overall performance of the CL-MRES protocol is dominated by the performance of the **HybridEncryption** algorithm, we measured the execution time of the **HybridEncryption** algorithm at the drone when it utilizes the CPU or the GPU. We also implemented eCLSC-TKEM and measured the execution time of the **SymmetricKeyGen** and **Encapsulation** algorithms without the signature generation step for fair comparisons. Fig. 10 shows the execution times of CL-MRES and eCLSC-TKEM at the drone when the number of cars (n) ranged from 1 to 2,000. When n is 1, the performance of CL-MRES is equal to the performance of eCLSC-TKEM. However, as n increases, CL-MRES is approximately 1.5 times faster than eCLSC-TKEM since CL-MRES re-uses randomness.

When n is 1, the CPU executes the CL-MRES protocol more quickly than the GPU since the clock speed of the CPU is higher than the clock speed of the GPU. However, as n increases, the GPU can execute the CL-MRES protocol much faster than the GPU since the GPU can compute EC point multiplications in parallel. For example, when n is 2,000, the CPU takes 15.87 seconds, while the GPU takes only 125 ms. These results confirm that the GPU utilization for CL-MRES is imperative when the drone has to communicate with a large number of cars.

VII. CONCLUSIONS

In this paper, a suite of secure communication protocols for smart city monitoring applications is presented. As building blocks, we propose eCLSC-TKEM, CL-MRES, CLDA and a dual communication channel strategy. eCLSC-TKEM efficiently supports four security functions: key agreement, user authentication, non-repudiation, and user revocation. CL-MRES is a hybrid encryption for multiple recipients and is designed for a drone to transmit user-specific data to a large number of smart objects. CLDA allows the data collection party, such a drone, to collect privacy-sensitive data from smart objects in an efficient and secure way by combining the optimized batch verification scheme and the ElGamal homomorphic encryption scheme with our certificateless approach. The dual channel strategy helps drones and cars save their battery life by allowing them to concurrently execute the time-consuming crypto-algorithms. Our protocols are applicable to data applications, other than smart cities, that involve different types of fixed and mobile devices with different capacities.

APPENDIX A
SECURITY PROOF OF eCLSC-TKEM

A. 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 [14], [20]. 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 SetUp() algorithm to generate the public parameters `params` and the master private key `msk` respectively. If $\mathcal{A}$ is either $\mathcal{A}_{\mathcal{I}}$ or $\mathcal{A}_{\mathcal{III}}$, $\mathcal{C}$ gives `params` to $\mathcal{A}$ while keeping `msk` secret. If $\mathcal{A}$ is $\mathcal{A}_{\mathcal{II}}$, $\mathcal{C}$ gives both `params` and `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 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 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 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 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 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 SymmetricKeyGen to obtain the ric 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 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 $(K_1, \omega^*) \leftarrow$ $\mathsf{SymmetricKeyGen}(\texttt{params}, ID_{A^*}, pk_{A^*}, sk_{A^*}, ID_{B^*}, pk_{B^*}$ $, t_{B^*})$ 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) [38].

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 $\texttt{params}$ and the master private key $\texttt{msk}$ respectively. If $\mathcal{F}$ is either $\mathcal{F}_{\mathcal{I}}$ or $\mathcal{F}_{\mathcal{III}}$, $\mathcal{C}$ gives $\texttt{params}$ to $\mathcal{F}$ while keeping $\texttt{msk}$ secret. If $\mathcal{F}$ is $\mathcal{F}_{\mathcal{II}}$, $\mathcal{C}$ gives both $\texttt{params}$ and $\texttt{msk}$ 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 $\mathsf{Decapsulation}(\texttt{params}, ID_{A^*}, pk_{A^*}, t_{A^*}, ID_{B^*},$ $pk_{B^*}, sk_{B^*}, \varphi^*, \tau^*)$ is valid, $\mathcal{F} \in \{\mathcal{F}_{\mathcal{I}}, \mathcal{F}_{\mathcal{II}}\}$ wins the game. If the output of $\mathsf{Decapsulation}(\texttt{params}, ID_{A^*}, pk_{A^*},$ $t'_{A^*}, ID_{B^*}, pk_{B^*}, sk_{B^*}, \varphi^*, \tau^*)$ 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 $\mathcal{F}$

B. FORMAL SECURITY PROOF OF eCLSC-TKEM

The security of our eCLSC-TKEM relies on the hardness of the following problems.

1) 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) [38] 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 $R = yP$.

2) 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) [38] 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 $Q = xP, R' = y'P, S' = z'P$?

3) 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 [15].

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.

Proof: Suppose that there exists a Type I adversary $\mathcal{A}_{\mathcal{I}}$ who can break the IND-eCLSC-TKEM-CCA2-I 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.

A challenger $\mathcal{C}$ is challenged with an instance of the OGDH (One-sided Gap Diffie-Hellman) problem. The OGDH for a group G_q with a generator P and a fixed second point $Q(= aP)$ has as input $R(= bP) \in G_q$ and computes for the point $S(= cP) \in G$ such that $c = ab \pmod q$, by accessing a ODDH (One-sided Decision Diffie-Hellman) oracle. Here, the ODDH oracle solves the OGDH problem for a group G_q with a generator P and a fixed second point Q as input $R', S' \in G_q$ and decides whether $c' = ab' \bmod q$, where $a, b', c' \in Z_q^*$ such that $Q = aP, R' = b'P, S' = c'P$. Let $\mathcal{A}_{\mathcal{I}}$ be an adversary who is able to break the IND-eCLSC-TKEM-CCA2-I security of the eCLSC-TKEM. $\mathcal{C}$ can utilize $\mathcal{A}_{\mathcal{I}}$ to compute the solution abP of the OGDH instance by playing the following interactive game with $\mathcal{A}_{\mathcal{I}}$. To solve the OGDH problem, $\mathcal{C}$ 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 $\mathcal{A}_{\mathcal{I}}$. In order to avoid the inconsistency between the responses to the hash queries, $\mathcal{C}$ maintains lists $L_i (0 \leq i \leq 3)$. It also maintains a list of issued private keys and public keys 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 t, where $1 \leq t \leq q_{H_0}$ and fix ID_t as the target identity for the challenge phase.

Phase I: $\mathcal{A}_{\mathcal{I}}$ may make a series of polynomially bounded numbers of queries to random oracles $H_i (0 \leq i \leq 3)$ at any time and $\mathcal{C}$ responds as follows:

Create(ID_i): When $\mathcal{A}_{\mathcal{I}}$ submits a Create(ID_i) query to $\mathcal{C}$, $\mathcal{C}$ responds as follows:

- If $ID_i = ID_t$, $\mathcal{C}$ chooses $e_t, x_t \in_R Z_q^*$ and sets $H_0(ID_t, R_t, P_t, t_t) = -e_t$, $R_t = e_t P_{pub} - P_t + aP$ and $P_t = x_t P$. Here, $\mathcal{C}$ does not know a. $\mathcal{C}$ uses the aP given in the instance of the OGDH problem. $\mathcal{C}$ inserts $\langle ID_t, R_t, P_t, t_t, -e_t \rangle$ into the list L_0 and $\langle ID_t, \perp, x_t, R_t, P_t, t_t \rangle$ into the list L_k.
- If $ID_i \neq ID_t$, $\mathcal{C}$ 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.

H_0 queries: When $\mathcal{A}_{\mathcal{I}}$ 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 $\mathcal{A}_{\mathcal{I}}$ submits a H_1 query with $(Y_i, V_i, T_i, ID_j, P_j, ID_i, P_i)$, where $i \neq j$, $\mathcal{C}$ checks whether the ODDH oracle returns 1 when queried with the tuple (aP, V_i, T_i). If the ODDH oracle returns 1, $\mathcal{C}$ outputs T_i and stop. Then $\mathcal{C}$ goes through the list L_1 with entries $\langle Y_i, V_i, *, ID_j, P_j, ID_i, P_i, l_i \rangle$, 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 $ID_i = ID_t$. 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 list L_1, which is initially empty, with a tuple containing the input and return values. $\mathcal{C}$ then returns l to $\mathcal{A}_{\mathcal{I}}$.

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

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

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:

- If $ID_i = ID_t$, $\mathcal{C}$ aborts the execution.
- If $ID_i \neq ID_t$, $\mathcal{C}$ retrieves the tuple $\langle ID_i, d_i, x_i, R_i, P_i, t_i \rangle$ from L_k, returns (d_i, R_i) which satisfies the equation $d_i P = R_i + H_0(ID_i, R_i, P_i, t_i) P_{pub}$.

Extract-Secret-Value queries: $\mathcal{A}_{\mathcal{I}}$ produces ID_i to $\mathcal{C}$ and requests a secret value of the user with ID_i. If the public key of ID_i has not been replaced and $ID_i \neq ID_t$, then $\mathcal{C}$ responds with x_i by retrieving from the list L_k. If $\mathcal{A}_{\mathcal{I}}$ has already replaced the public key, $\mathcal{C}$ does not provide the corresponding secret value to $\mathcal{A}_{\mathcal{I}}$. If $ID_i = ID_t$, $\mathcal{C}$ aborts.

Request-Public-Key queries: $\mathcal{A}_{\mathcal{I}}$ produces ID_i to $\mathcal{C}$ and requests a public key of the user with ID_i. $\mathcal{C}$ checks in the list L_k for a tuple of the form $\langle ID_i, d_i, x_i, R_i, P_i, t_i \rangle$. 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) as the answer.

Public-Key-Replacement queries: $\mathcal{A}_{\mathcal{I}}$ chooses values (R_i', P_i', t_i') to replace the public key (R_i, P_i, t_i) of a user ID_i. $\mathcal{C}$ updates the corresponding tuple in the list 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 computations or responses to any queries made by $\mathcal{A}_{\mathcal{I}}$.

Symmetric Key Generation queries: $\mathcal{A}_{\mathcal{I}}$ produces a sender's identity ID_A, public key (R_A, P_A, t_A), the receiver's identity 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:

- If $ID_A \neq ID_t$, $\mathcal{C}$ computes the full private key sk_A corresponding to ID_A by executing the Extract-Partial-Private-Key query and Extract-Secret-Value query algorithm. Then, $\mathcal{C}$ gets the symmetric key K and an internal state information ω by running the actual SymmetricKeyGen algorithm. $\mathcal{C}$ stores ω and overwrite any previous value. $\mathcal{C}$ sends the symmetric key K to $\mathcal{A}_{\mathcal{I}}$.
- If $ID_A = ID_t$ (and hence $ID_B \neq ID_t$), $\mathcal{C}$ chooses $r_1, r_2, h_t, h_t' \in_R Z_q^*$ and computes $U = r_1P - {h_t}^{-1}\cdot(aP + P_t)$, $V = r_2P$, $Y = R_B + H_0(ID_B, R_B, P_B, t_B)\cdot P_{pub} + P_B$, $T = r_2 \cdot Y \bmod q = r_2\cdot(H_0(ID_B, R_B, P_B, t_B)P_{pub} + R_B + P_B) \bmod q$ and $K = H_1(Y, V, T, ID_A, P_A, ID_B, P_B)$, where R_B and P_B are obtained by calling the Request-Public-Key query oracle on ID_B. Note that $\omega = (r_1, r_2, h_t, h_t', U, V, T, ID_A, pk_A, ID_B, pk_B)$.
- $\mathcal{C}$ goes through the list L_1 looking for an entry $(Y, V, T, ID_A, P_A, ID_B, P_B, k)$ for some k such that ODDH(Y, V, T)=1. If such an entry exists, it computes $K \leftarrow l$. Otherwise it uses a random l and updates the list L_1 with $(Y, V, *, ID_A, P_A, ID_B, P_B, l)$. $\mathcal{C}$ stores ω and sends the symmetric key K to $\mathcal{A}_{\mathcal{I}}$.

Key Encapsulation queries: $\mathcal{A}_{\mathcal{I}}$ produces an arbitrary tag τ, the sender's identity ID_A, public key (R_A, P_A, t_A), the receiver's identity 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 list L_k. $\mathcal{C}$ checks whether a corresponding ω value has been stored previously.

- If ω does not exist, $\mathcal{C}$ returns an invalid reply.
- If a corresponding ω exists and $ID_A \neq ID_t$, then $\mathcal{C}$ computes φ with ω and τ by using the actual Encapsulation algorithm, and deletes ω.
- If a corresponding ω exists and $ID_A = ID_t$, then $\mathcal{C}$ computes φ by performing the following steps. Note that ω is $(r_1, r_2, h_t, h_t', U, V, T, ID_A, pk_A, ID_B, pk_B)$ and $\mathcal{C}$ does not know the private key corresponding to ID_t. So $\mathcal{C}$ should perform the encapsulation in a different way:
  1) Set $H = h_t$ and add the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_t\rangle$ to the list L_2.
  2) Set $H' = h_t'$ and add the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_t'\rangle$ to the list L_3.
  3) Compute $W = h_t \cdot r_1 + h_t' \cdot x_A$.
  4) Output $\varphi = (U, V, W)$ as the encapsulation.

  We show that $\mathcal{A}_{\mathcal{I}}$ 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:

$$\begin{aligned} W \cdot P &= R_A + H_0(ID_A, R_A, P_A, t_A)\cdot P_{pub} \\ &\quad + H \cdot U + H' \cdot P_A \\ &= aP + e_tP_{pub} - P_t + (-e_t)\cdot P_{pub} \\ &\quad + h_t \cdot (r_1P - {h_t}^{-1}\cdot(aP + P_t)) + h_t' \cdot P_A \\ &= h_t \cdot r_1P + h_t' \cdot P_A \\ &= W \cdot P \end{aligned}$$

Key Decapsulation queries: $\mathcal{A}_{\mathcal{I}}$ produces an encapsulation $\varphi = (U, V, W)$, a tag τ, the sender's identity ID_A, the public key (R_A, P_A, t_A), the receiver's identity 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.

- If $ID_B \neq ID_t$, then $\mathcal{C}$ computes the decapsulation of φ by using the actual Decapsulation algorithm.
- If $ID_B = ID_t$, then $\mathcal{C}$ computes K from φ as follows:
  1) Searches in the list L_2 and L_3 for entries of the type $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_t\rangle$ and $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_t'\rangle$ respectively.
  2) If entries $H = h_t$ and $H' = h_t'$ 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.
  3) If the above equality holds, the corresponding value of T is retrieved from the lists L_2 and L_3. Both the T values should be equal.
  4) $\mathcal{C}$ goes through L_1 and looks for a tuple of the form $\langle Y, V, T, ID_A, P_A, ID_B, P_B, l\rangle$ 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.
  5) If $\mathcal{C}$ reaches this point of execution, it put the entry $\langle Y, V, *, ID_A, P_A, ID_B, P_B, l\rangle$ for a random l on the list L_1 and returns $K \leftarrow l$. The symbol $*$ denotes an unknown value. Note that the identity component of all entries with a $*$ is a receiver identity ID_B.

Challenge: At the end of Phase I, $\mathcal{A}_{\mathcal{I}}$ sends a sender's identity ID_{A*} and a receiver's identity ID_{B*} on which $\mathcal{A}_{\mathcal{I}}$ wishes to be challenged to $\mathcal{C}$. Here, the partial private key of the receiver ID_{B*} was not queried in Phase I. $\mathcal{C}$ aborts the game if $ID_{B*} \neq ID_t$. 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 $\mathcal{A}_{\mathcal{I}}$.
6) $\mathcal{A}_{\mathcal{I}}$ 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 $\varphi^* = \langle U^*, V^*, W^* \rangle$ to $\mathcal{A_I}$.

Phase II: $\mathcal{A_I}$ adaptively queries the oracles as in Phase I. Besides it cannot query decapsulation on φ^*.

Guess: Since $\mathcal{A_I}$ can break the IND-eCLSC-TKEM-CCA2-I security (which is assumed at the beginning of the proof), $\mathcal{A_I}$ should have asked a H_1 query with $(Y^*, V^*, T^*, ID_{A*}, P_{A*}, ID_{B^*}, P_{B^*})$ as inputs. It is to be noted that $T^* = b \cdot Y^* = b \cdot (-e_t \cdot P_{pub} + e_t \cdot P_{pub} - P_t + aP + P_{B^*}) = ab \cdot P$, where $P_t = P_{B*}$ because of $ID_t = ID_{B*}$. 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-I game.

- E_1 is an event in which $\mathcal{A_I}$ queries the partial private key of the target identity ID_t. The probability of E_1 is $Pr[E_1] = \frac{q_{ppri}}{q_C \cdot q_{H_0}}$.
- E_2 is an event in which $\mathcal{A_I}$ asks to query the set secret value of the target identity ID_t. The probability of E_2 is $Pr[E_2] = \frac{q_{sv}}{q_C \cdot q_{H_0}}$.
- E_3 is an event in which $\mathcal{A_I}$ does not choose the target identity ID_t as the receiver during the challenge. The probability of E_3 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-I game is

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

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:

$$Pr[\mathcal{C}(P, aP, bP) = abP] = \delta \cdot (1 - \frac{q_{ppri}}{q_C \cdot q_{H_0}}) \cdot (1 - \frac{q_{sv}}{q_C \cdot q_{H_0}}) \cdot (\frac{1}{q_C \cdot q_{H_0} - q_{ppri} - q_{sv}}) \cdot (\frac{1}{q_{H_1}})$$

Therefore, the $Pr[\mathcal{C}(P, aP, bP) = abP]$ is non-negligible, because δ is non-negligible. This contradicts the OGDH assumption.

Lemma 2: In the random oracle model, if there exists an adversary $\mathcal{A_{II}}$ 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.

Proof: A challenger $\mathcal{C}$ is challenged with an instance of the OGDH problem. Let $\mathcal{A_{II}}$ be an adversary who is able to break the IND-eCLSC-TKEM-CCA2-II security of the eCLSC-TKEM. $\mathcal{C}$ can utilize $\mathcal{A_{II}}$ to compute the solution abP of the OGDH instance by playing the following interactive game with $\mathcal{A_{II}}$. To solve the OGDH, $\mathcal{C}$ chooses $s \in_R \mathbb{Z}_q^*$, sets the master public key $P_{pub} = sP$, 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 parameter $\Omega = \{F_q, E/F_q, G_q, P, P_{pub} = sP, H_0, H_1, H_2, H_3\}$ and the master private key s to $\mathcal{A_{II}}$. In order to avoid the inconsistency between the responses to the hash queries, $\mathcal{C}$ maintains lists $L_i (0 \leq i \leq 3)$. It also maintains a list L_k of the issued private keys and public keys. $\mathcal{C}$ can simulate the challenger's execution of each phase of the formal Game. Let $\mathcal{C}$ select a random index t, where $1 \leq t \leq q_{H_0}$ and fixes ID_t as the target identity for the challenge phase.

Phase I: $\mathcal{A_{II}}$ may make a series of polynomially bounded number of queries to random oracles $H_i (0 \leq i \leq 3)$ at any time and $\mathcal{C}$ responds as follows:

Create(ID_i) queries: When $\mathcal{A_{II}}$ submits a Create(ID_i) query to $\mathcal{C}$, $\mathcal{C}$ responds as follows:

- If $ID_i = ID_t$, $\mathcal{C}$ chooses $l_t \in_R Z_q^*$ and sets $H_0(ID_t, R_t, P_t, t_t) = l_t$, computes $R_t = -l_t P_{pub}$, $d_t = -l_t \cdot s$ and the public key as $P_t = aP$. Here, $\mathcal{C}$ does not know a. $\mathcal{C}$ uses the aP given in the instance of the OGDH problem. $\mathcal{C}$ inserts $\langle ID_t, R_t, P_t, t_t, l_t \rangle$ into the list L_0 and $\langle ID_t, d_t, \perp, R_t, P_t, t_t \rangle$ into the list L_k.
- If $ID_i \neq ID_t$, $\mathcal{C}$ picks $a_i, x_i, l_i \in_R Z_q^*$, then sets $H_0(ID_i, R_i, P_i, t_i) = l_i$, computes $R_i = a_i P$, $d_i = a_i + l_i \cdot s$ and the public key as $P_i = x_i P$. $\mathcal{C}$ inserts $\langle ID_i, R_i, P_i, t_i, l_i \rangle$ 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.

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

H_1 queries: When $\mathcal{A_{II}}$ submits a H_1 query with $(Y_i, V_i, T_i, ID_j, P_j, ID_i, P_i)$, where $i \neq j$, $\mathcal{C}$ checks whether the ODDH (One-sided Decision Diffie-Hellman) oracle returns 1 when queried with the tuple (aP, V_i, T_i). If the ODDH oracle returns 1, $\mathcal{C}$ outputs Y_i and stop. Then $\mathcal{C}$ goes through the list L_1 with entries $\langle Y_i, V_i, *, ID_j, P_j, ID_i, P_i, l_i \rangle$, 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 $ID_i = ID_t$. 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 list L_1, which is initially empty, with a tuple containing the input and return values. $\mathcal{C}$ then returns l to $\mathcal{A_{II}}$.

H_2 queries: When $\mathcal{A}_{\mathcal{II}}$ submits a H_2 query with ID_i, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i\rangle$ exists in the list L_2. If it exists, $\mathcal{C}$ returns $H = h_i$ to $\mathcal{A}_{\mathcal{II}}$. Otherwise, $\mathcal{C}$ chooses $h_i \in_R Z_q^*$, adds the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i\rangle$ to L_2 and returns $H = h_i$ to $\mathcal{A}_{\mathcal{II}}$.

H_3 queries: When $\mathcal{A}_{\mathcal{II}}$ submits a H_3 query with ID_i, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h'_i\rangle$ exists in the list L_3. If it exists, $\mathcal{C}$ returns $H' = h'_i$ to $\mathcal{A}_{\mathcal{II}}$. Otherwise, $\mathcal{C}$ chooses $h'_i \in_R Z_q^*$, adds the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h'_i\rangle$ to L_3 and returns $H' = h'_i$ to $\mathcal{A}_{\mathcal{II}}$.

Extract-Partial-Private-Key queries: When $\mathcal{A}_{\mathcal{II}}$ asks a Extract-Partial-Private-Key query for ID_i, $\mathcal{C}$ checks whether the corresponding partial private key for ID_i, d_i exists in the list L_k. If it exists, $\mathcal{C}$ returns d_i to $\mathcal{A}_{\mathcal{II}}$. Otherwise, $\mathcal{C}$ recalls Create(ID_i) query to obtain d_i and returns d_i as the answer.

Extract-Secret-Value queries: If $\mathcal{A}_{\mathcal{II}}$ asks a Extract-Secret-Value query for ID_i, $\mathcal{C}$ answers as follows:

- If $ID_i = ID_t$, $\mathcal{C}$ aborts.
- If $ID_i \neq ID_t$, $\mathcal{C}$ looks for the tuple $\langle ID_i, d_i, x_i, R_i, P_i, t_i\rangle$ in the list L_k. If such tuple exists in L_k, $\mathcal{C}$ returns x_i. Otherwise, $\mathcal{C}$ recalls Create(ID_i) query to obtain x_i and returns x_i as the answer.

Set-Public-Key queries: When $\mathcal{A}_{\mathcal{II}}$ asks Set-Public-Key query for ID_i, $\mathcal{C}$ searches the list L_k. If the public key for ID_i, (R_i, P_i, t_i) is found in L_k, $\mathcal{C}$ returns (R_i, P_i, t_i) as the answer. Otherwise, $\mathcal{C}$ executes a Create(ID_i) query to obtain (R_i, P_i, t_i) and then returns (R_i, P_i, t_i) as the answer.

Public-Key-Replacement queries: When $\mathcal{A}_{\mathcal{II}}$ asks Public-Key-Replacement query for ID_i, $\mathcal{C}$ checks whether $ID_i = ID_t$. If $ID_i = ID_t$, $\mathcal{C}$ aborts. Otherwise, $\mathcal{C}$ updates the corresponding tuple in the list L_k as $\langle ID_i, -, -, R'_i, P'_i, t'_i\rangle$, where (R'_i, P'_i, t'_i) is chosen by $\mathcal{A}_{\mathcal{II}}$. The current public key (i.e. replaced public key) is used by $\mathcal{C}$ for computations or responses to any queries made by $\mathcal{A}_{\mathcal{II}}$.

Symmetric Key Generation queries: $\mathcal{A}_{\mathcal{II}}$ produces a sender's identity ID_A, public key (R_A, P_A, t_A), the receiver's identity 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:

- If $ID_A \neq ID_t$, $\mathcal{C}$ computes the full private key sk_A corresponding to ID_A by executing the Extract-Partial-Private-Key query and Extract-Secret-Value query algorithm. Then, $\mathcal{C}$ gets the symmetric key K and an internal state information ω by running the actual SymmetricKeyGen algorithm. $\mathcal{C}$ stores ω and overwrite any previous value. $\mathcal{C}$ sends the symmetric key K to $\mathcal{A}_{\mathcal{II}}$.
- If $ID_A = ID_t$ (and hence $ID_B \neq ID_t$), $\mathcal{C}$ chooses $r_1, r_2, h_t, h'_t \in_R Z_q^*$ and computes $U = r_1P - {h_t}^{-1} \cdot h'_t \cdot P_A$, where P_A is obtained from the list L_k, $V = r_2P$, $T = r_2 \cdot Y$, $Y = R_B + H_0(ID_B, R_B, P_B, t_B)P_{pub} + P_B \bmod q$ and $K = H_1(Y, V, r_2 \cdot Y, ID_A, P_A, ID_B, P_B)$. Note that $\omega = (r_1, r_2, h_t, h'_t, U, V, T, ID_A, pk_A, ID_B, pk_B)$.
- $\mathcal{C}$ goes through the list L_1 looking for an entry $(Y, V, T, ID_A, P_A, ID_B, P_B, k)$ for some k such that ODDH(Y, V, T)=1. If such an entry exists, it computes $K \leftarrow l$. Otherwise it uses a random l and updates the list L_1 with $(Y, V, *, ID_A, P_A, ID_B, P_B, l)$. $\mathcal{C}$ stores ω and sends the symmetric key K to $\mathcal{A}_{\mathcal{II}}$.

Key Encapsulation queries: $\mathcal{A}_{\mathcal{II}}$ produces an arbitrary tag τ, the sender's identity ID_A, public key (R_A, P_A, t_A), the receiver's identity ID_B and public key (R_B, P_B, t_B) then sends to $\mathcal{C}$. $\mathcal{C}$ checks whether a corresponding ω value is stored previously.

- If ω does not exist, $\mathcal{C}$ returns invalid.
- If a corresponding ω exists and $ID_A \neq ID_t$, then $\mathcal{C}$ computes φ with ω and τ by using the actual Encapsulation algorithm, and deletes ω. Here, $\mathcal{C}$ gets the full private key of the sender $sk_A = (d_A, x_A)$ from the list L_k.
- If a corresponding ω exists and $ID_A = ID_t$, then $\mathcal{C}$ computes φ by performing the following steps. Note that ω is $(r_1, r_2, h_t, h'_t, U, V, T, ID_A, d_A, pk_A, ID_B, pk_B)$ and $\mathcal{C}$ does not know the secret value x_A corresponding to ID_A. So $\mathcal{C}$ should perform the encapsulation in a different way:
  1) Set $H = h_t$ and add the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_t\rangle$ to the list L_2.
  2) Set $H' = h'_t$ and add the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h'_t\rangle$ to the list L_3.
  3) Compute $W = r_1H$.
  4) Output $\varphi = (U, V, W)$ as the encapsulation.

  We show that $\mathcal{A}_{\mathcal{II}}$ 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:
  $$\begin{aligned} &R_A + H_0(ID_A, R_A, P_A, t_A) \cdot P_{pub} + H \cdot U + H' \cdot P_A \\ &= -l_t \cdot P_{pub} + l_t \cdot P_{pub} + h_t \cdot (r_1P - {h_t}^{-1} \cdot h'_t \cdot P_A) + h'_t \cdot P_A \\ &= h_t \cdot r_1P - h'_t \cdot P_A + h'_t \cdot P_A \\ &= h_t \cdot r_1P \\ &= W \cdot P \end{aligned}$$

Key Decapsulation queries: $\mathcal{A}_{\mathcal{II}}$ produces an encapsulation φ, a 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) to $\mathcal{C}$.

- If $ID_B \neq ID_t$, then $\mathcal{C}$ computes the decapsulation of φ by using the actual Decapsulation algorithm. Here, the full private key of the receiver (d_B, x_B) is obtained from the list L_k.
- If $ID_B = ID_t$, then $\mathcal{C}$ computes K from φ as follows:
  1) Searches lists L_2 and L_3 for entries of the type $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_t\rangle$ and $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h'_t\rangle$, respectively.
  2) If entries $H = h_t$ and $H' = h'_t$ 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.
  3) If the above equality holds, then retrieves the corresponding value of T from lists L_2 and L_3. Both the T values should be equal.
  4) $\mathcal{C}$ goes through L_1 and looks for a tuple $\langle Y, V, T, ID_A, P_A, ID_B, P_B, l\rangle$ such that ODDH $(Y, V, T) = 1$. If such an entry exists, then

$\mathcal{C}$ found the correct value of T and outputs $K \leftarrow l$.

5) If $\mathcal{C}$ reaches this point of execution, it places the entry $\langle Y, V, *, ID_A, P_A, ID_B, P_B, l\rangle$ for a random l on list L_1 and returns the corresponding $K \leftarrow l$ value as the decapsulation of φ. The symbol $*$ denotes an unknown value of Y.

Challenge: At the end of Phase I, $\mathcal{A}_{\mathcal{II}}$ sends a sender identity ID_{A*} and a receiver identity ID_{B*} on which $\mathcal{A}_{\mathcal{II}}$ wishes to be challenged to $\mathcal{C}$. Here, the secret value of the receiver ID_{B*} was not queried in Phase 1. $\mathcal{C}$ aborts the game if $ID_{B*} \neq ID_t$. Otherwise, $\mathcal{C}$ performs the following 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) Compute K_1 by executing Symmetric Key Generation queries.
5) Set $\omega^* = \langle -, Y^*, V^*, T^*, ID_{A*}, P_{A*}, R_{A*}, x_{A*}, d_{A*}, ID_{B*}, P_{B*}, R_{B*}\rangle$.
6) $\mathcal{C}$ chooses a bit $\delta \in_R \{0, 1\}$ and sends K_δ to $\mathcal{A}_{\mathcal{II}}$.
7) $\mathcal{A}_{\mathcal{II}}$ generates an arbitrary tag τ^* and sends it to $\mathcal{C}$.
8) Choose $h_i, h_i' \in_R \mathbb{Z}_q^*$, store the tuple $\langle U^*, \tau^*, T^*, ID_{A*}, P_{A*}, ID_{B*}, P_{B*}, h_i\rangle$ to the list L_2 and $\langle U, \tau^*, T, ID_{A*}, P_{A*}, ID_{B*}, P_{B*}, h_i'\rangle$ to the list L_3.
9) Since $\mathcal{C}$ knows the private key of the sender ID_{A*}, $\mathcal{C}$ computes $W = d_{A*} + r \cdot h_i + x_{A*} \cdot h_i'$.
10) $\mathcal{C}$ sends $\varphi^* = \langle U^*, V^*, W^*\rangle$ to $\mathcal{A}_{\mathcal{II}}$.

Phase II: $\mathcal{A}_{\mathcal{II}}$ adaptively queries the oracles as in Phase I, consistent with the constraints for a Type-II adversary. Other than this, it cannot query decapsulation on φ^*.

Guess: Since $\mathcal{A}_{\mathcal{II}}$ is able to break the IND-eCLSC-TKEM-CCA2-II security of eCLSC-TKEM (which is assumed at the beginning of the proof), $\mathcal{A}_{\mathcal{II}}$ should have asked a H_1 query with $(Y^*, V^*, T^*, ID_{A*}, P_{A*}, ID_{B*}, P_{B*})$ as inputs. Since ID_{B*} is a target identity ID_t, $P_{B*} = aP$. Here, aP was given as the instance of the OGDH problem and $\mathcal{C}$ does not know a. Thus, computing $T^* = b \cdot (-l_t P_{pub} + l_t P_{pub} + aP) = abP$ is to find abP when $\langle aP(= P_{B*}), bP(= V), T\rangle \in G_q$ are given. Therefore, if the list L_1 has q_{H_1} queries corresponding to the sender ID_{A*} and receiver ID_{B*}, one of the q_{H_1} values of T^* 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 list L_1 and outputs it as the solution for the OGDH instance.

Analysis: In order to assess the probability of success of the challenger $\mathcal{C}$, let E_1, E_2, and E_3 be the events in which $\mathcal{C}$ aborts the IND-eCLSC-TKEM-CCA2-II game.

- E_1 is an event in which $\mathcal{A}_{\mathcal{II}}$ queries the secret value of the target identity ID_t. The probability of E_1 is $Pr[E_1] = \frac{q_{sv}}{q_C \cdot q_{H_0}}$.
- E_2 is an event in which $\mathcal{A}_{\mathcal{II}}$ asks to replace the public key of the target identity ID_t. The probability of E_2 is $Pr[E_2] = \frac{q_{pkR}}{q_C \cdot q_{H_0}}$.
- E_3 is an event in which $\mathcal{A}_{\mathcal{II}}$ does not choose the target identity ID_t as the receiver during the challenge. The probability of E_3 is $Pr[E_3] = 1 - \frac{1}{q_C \cdot q_{H_0} - q_{sv} - q_{pkR}}$.

Thus, the probability that $\mathcal{C}$ does not abort the IND-eCLSC-TKEM-CCA2-II game is $Pr[\neg E_1 \wedge \neg E_2 \wedge \neg E_3] = (1 - \frac{q_{sv}}{q_C \cdot q_{H_0}}) \cdot (1 - \frac{q_{pkR}}{q_C \cdot q_{H_0}}) \cdot (\frac{1}{q_C \cdot q_{H_0} - q_{sv} - q_{pkR}})$

The probability that $\mathcal{C}$ randomly chooses the Y^* from L_1 and Y^* 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:

$$\begin{aligned} &Pr[\mathcal{C}(P, aP, bP) = abP] \\ &= \varepsilon \cdot (1 - \frac{q_{sv}}{q_C \cdot q_{H_0}}) \cdot (1 - \frac{q_{pkR}}{q_C \cdot q_{H_0}}) \\ &\cdot (\frac{1}{q_C \cdot q_{H_0} - q_{sv} - q_{pkR}}) \cdot (\frac{1}{q_{H_1}}) \end{aligned}$$

Therefore, the $Pr[\mathcal{C}(P, aP, bP) = abP]$ is non-negligible, because ε is non-negligible.

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.

Proof: 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 I: $\mathcal{A}_{\mathcal{III}}$ 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 $\mathcal{A}_{III}$ 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 $\mathcal{A}_{III}$ 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 $\mathcal{A}_{III}$ submits a H_1 query with $(Y_i, V_i, T_i, ID_j, P_j, ID_i, P_i)$, where $i \neq j$, $\mathcal{C}$ 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 $\langle Y_i, V_i, *, ID_j, P_j, ID_i, P_i, l_i \rangle$, 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 $\mathcal{A}_{III}$.

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 $\mathcal{A}_{III}$. 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 $\mathcal{A}_{III}$.

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 $\mathcal{A}_{III}$. 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 $\mathcal{A}_{III}$.

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 ($ID_i \neq ID_j$) 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 $d_i P = R_i + H_0(ID_i, R_i, P_i, t_i) P_{pub}$.

Extract-Secret-Value queries: $\mathcal{A}_{III}$ 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 $ID_i \neq ID_j$) 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 $\mathcal{A}_{III}$ has already replaced the public key of ID_i, $\mathcal{C}$ does not provide the corresponding secret value to $\mathcal{A}_{III}$. If $ID_i = ID_j$ and $t_j > t^\star$, $\mathcal{C}$ aborts.

Request-Public-Key queries: $\mathcal{A}_{III}$ 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: $\mathcal{A}_{III}$ 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 $\mathcal{A}_{III}$.

Symmetric Key Generation queries: $\mathcal{A}_{III}$ 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 ($ID_A \neq ID_j$) 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 $\mathcal{A}_{III}$. (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 $U = r_1 P - h_t{}^{-1} \cdot aP + h_t{}^{-1} \cdot P_t$, $V = r_2 P$, $Y = R_B + H_0(ID_B, R_B, P_B, t_B) \cdot P_{pub} + P_B$, $T = r_2 \cdot Y \bmod q = r_2 \cdot (H_0(ID_B, R_B, P_B, t_B) P_{pub} + R_B + P_B) \bmod q$ and $K = H_1(Y, V, T, ID_A, P_A, ID_B, P_B)$, 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 $(Y, V, T, ID_A, P_A, ID_B, P_B, k)$ 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 $(Y, V, *, ID_A, P_A, ID_B, P_B, l)$. $\mathcal{C}$ stores ω and sends K to $\mathcal{A}_{III}$.

Key Encapsulation queries: $\mathcal{A}_{III}$ 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 $ID_A \neq ID_j$) 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 $\mathcal{A}_{III}$ 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 ($ID_B \neq ID_j$) 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 $\langle Y, V, *, ID_A, P_A, ID_B, P_B, l\rangle$ 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 ($ID_{B*} \neq ID_j$) 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^*:

- Choose $r \in_R Z_q^*$ and compute $U^* = rP$.
- 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.
- Choose $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the eCLSC-TKEM.
- Choose a random hash value l^* and set $K_1 = l^*$.
- $\mathcal{C}$ chooses a bit $\delta \in_R \{0, 1\}$ and sends K_δ to $\mathcal{A}_{\mathcal{III}}$.
- $\mathcal{A}_{\mathcal{III}}$ generates τ^* and sends it to $\mathcal{C}$.
- 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.
- Since $\mathcal{C}$ knows the sender's private key, $\mathcal{C}$ computes $W^* = d_{A*} + r \cdot h_i + x_{A*} \cdot h'_i$.
- $\mathcal{C}$ returns $\varphi^* = \langle U^*, V^*, W^*\rangle$.

Phase II: $\mathcal{A}_{\mathcal{III}}$ adaptively queries the oracles as in Phase I. Besides it cannot query decapsulation on φ^*.

Guess: Since $\mathcal{A}_{\mathcal{III}}$ can break the IND-eCLSC-TKEM-CCA2-III security (which is assumed at the beginning of the proof), $\mathcal{A}_{\mathcal{III}}$ should have asked a H_1 query with $(Y^*, V^*, T^*, ID_{A*}, P_{A*}, ID_{B^*}, P_{B^*})$ 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 $\mathcal{A}_{\mathcal{III}}$ 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 $\mathcal{A}_{\mathcal{III}}$ was not detected. The probability is $Pr[E_2] = \frac{1}{q}$.
(3) E_3: $\mathcal{A}_{\mathcal{III}}$ 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

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

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:

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

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

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 [15].

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

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

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.

Proof: 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 $\mathcal{F_I}$ be a forger who is able to break the EUF-eCLSC-TKEM-CMA-I security of the eCLSC-TKEM. $\mathcal{C}$ can utilize $\mathcal{F_I}$ to compute the solution b of the ECDLP instance by playing the following interactive game with $\mathcal{F_I}$. 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 $H_i(0 \leq i \leq 3)$ 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 $\mathcal{F_I}$. In order to avoid the inconsistency between the responses to the hash queries, $\mathcal{C}$ maintains lists $L_i(0 \leq i \leq 3)$). 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: $\mathcal{F_I}$ may make a series of 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 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-eCLSC-TKEM-CCA2-I game.

Forgery: Eventually, $\mathcal{F_I}$ 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 $ID_A = ID_j$, $\mathcal{C}$ 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 $h_i^* \neq h_i$ on the same τ as done in forking lemma [39]. 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.

$$\begin{aligned} W^* \cdot P - W \cdot P &= h_i^* \cdot U - h_i \cdot U \\ \Rightarrow (W^* - W)P &= (h_i^* - h_i) \cdot U \\ \Rightarrow (W^* - W)P &= (h_i^* - h_i) \cdot bP \\ \Rightarrow (W^* - W) \cdot (h_i^* - h_i)^{-1} &= b \end{aligned}$$

Therefore, $\mathcal{F_I}$ 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 $\mathcal{F_I}$ can ask q_C create (ID_i) queries, q_E key-encapsulation queries and q_{H_i} random oracle queries to H_i $(0 \leq i \leq 3)$. We also assume that $\mathcal{F_I}$ never repeats H_i $(0 \leq i \leq 3)$ a query with the same input.

- The success probability of the Create(ID_i) query execution is $(1 - \frac{q_{H_0}}{q})^{q_C} \geq 1 - \frac{q_{H_0} \cdot q_C}{q}$.
- The success probability of the H_2 query execution is $(1 - \frac{q_{H_2}}{q})^{q_{H_2}} \geq 1 - \frac{q_{H_2}^2}{q}$.
- The success probability of the H_3 query execution/ is $(1 - \frac{q_{H_3}}{q})^{q_{H_3}} \geq 1 - \frac{q_{H_3}^2}{q}$.
- The success probability of the key encapsulation query execution is $\frac{q_E}{(1-\frac{1}{q})} \geq q_E \cdot (1 + \frac{1}{q})$.
- The probability of both $ID_i = ID_j$ is $\frac{1}{q_C}$
- The probability that $\mathcal{F_I}$ queries the partial private key of the target identity ID_t is $\frac{q_{ppri}}{q_{H_0}}$.
- The probability that $\mathcal{F_I}$ asks to query the set secret value of the ID_t is $\frac{q_{sv}}{q_{H_0}}$.

Thus, the success probability that $\mathcal{C}$ can win the EUF-eCLSC-TKEM-CMA-I game is

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

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

Lemma 5: In the random oracle model, if there exists a forger $\mathcal{F_{II}}$ 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

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

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.

Proof: A challenger $\mathcal{C}$ is challenged with an instance of the ECDLP. Given $\langle P, aP \rangle \in G_q$, $\mathcal{C}$ must find a. Let $\mathcal{F_{II}}$ be a forger who is able to break the EUF-eCLSC-TKEM-CMA-II security of the eCLSC-TKEM. $\mathcal{C}$ can utilize $\mathcal{F_{II}}$ to compute the solution a of the ECDLP instance by playing the following interactive game with $\mathcal{F_{II}}$. To solve the ECDLP, $\mathcal{C}$ chooses $s \in_R \mathbb{Z}_q^*$, sets the master public key $P_{pub} = sP$, 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. Then $\mathcal{C}$ sends the system parameter $\Omega = \{F_q, E/F_q, G_q, P, P_{pub} = sP, H_0, H_1, H_2, H_3\}$ and the master private key s to $\mathcal{F_{II}}$. In order to avoid the inconsistency between the responses to the hash queries, $\mathcal{C}$ maintains lists $L_i(0 \leq i \leq 3)$. It also maintains a list L_k of issued private keys and public keys. $\mathcal{C}$ can simulate the challenger's execution of each phase of the formal Game.

Training Phase: $\mathcal{F_{II}}$ may make a series of 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 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-eCLSC-TKEM-CCA2-II game.

Forgery: Eventually, $\mathcal{F}_{\mathcal{II}}$ returns a valid encapsulation $\langle \tau, \varphi = (U, V, W), ID_A, ID_B \rangle$ on an arbitrary tag τ to $\mathcal{C}$, where the target identity ID_t is the sender identity ID_A and ID_B is the receiver identity. The public key of the sender ID_t should not be replaced during the training phase. The secret value of the target identity ID_t should not be queried during the training phase. $\mathcal{C}$ searches the list L_3 and outputs another valid encapsulation $\langle \tau, \varphi^* = (U, V, W^*), ID_t, ID_B \rangle$ with different $h_i^{'*}$ such that $h_i^{'*} \neq h_i'$ on the same τ as done in the forking lemma [39]. Thus, we can get $W \cdot P = R_t - l_t \cdot P_{pub} + h_t \cdot U + h_t' \cdot P_t$ and $W^* \cdot P = R_t - l_t \cdot P_{pub} + h_t \cdot U + h_t^{'*} \cdot P_t$. Note that $P_t = aP$. Then if we subtract these two equations, we obtain following value.

$$
\begin{aligned}
W^* \cdot P - W \cdot P &= h_i^{'*} \cdot P_t - h_i' \cdot P_t \\
&\Rightarrow (W^* - W)P = (h_i^{'*} - h_i') \cdot aP \\
&\Rightarrow (W^* - W)P = (h_i^* - h_i')] \cdot aP \\
&\Rightarrow (W^* - W) = (h_i^* - h_i') \cdot a \\
&\Rightarrow (W^* - W) \cdot (h_i^* - h_i')^{-1} = a
\end{aligned}
$$

Therefore, $\mathcal{F}_{\mathcal{II}}$ solves the ECDLP as $a = \frac{W^* - W}{h_i^* - h_i'}$ using the algorithm $\mathcal{C}$ for a given random instance $\langle P, aP \rangle \in G_q$.

Analysis: In order to assess the probability of success of the challenger $\mathcal{C}$. We assume that $\mathcal{F}_{\mathcal{II}}$ can ask q_C, create (ID_i) queries, q_E key encapsulation queries, q_{H_i} random oracle queries to H_i $(0 \leq i \leq 3)$, q_{sv} set-secret-value queries, and q_{pkR} public key replacement queries. We also assume that $\mathcal{F}_{\mathcal{II}}$ never repeats H_i $(0 \leq i \leq 3)$ query with the same input.

- The success probability of the Create(ID_i) query execution is $(1 - \frac{q_{H_0}}{q})^{q_C} \geq 1 - \frac{q_{H_0} \cdot q_C}{q}$.
- The success probability of the H_2 query execution is $(1 - \frac{q_{H_2}}{q})^{q_{H_2}} \geq 1 - \frac{q_{H_2}^2}{q}$.
- The success probability of the H_3 query execution is $(1 - \frac{q_{H_3}}{q})^{q_{H_3}} \geq 1 - \frac{q_{H_3}^2}{q}$.
- The success probability of the key encapsulation query execution is $\frac{q_E}{(1 - \frac{1}{q})} \geq q_E \cdot (1 + \frac{1}{q})$.
- The probability that $ID_i = ID_t$ is $\frac{1}{q_C}$.
- The probability that $\mathcal{F}_{\mathcal{II}}$ queries the secret value of the target identity ID_t is $\frac{q_{sv}}{q_{H_0}}$.
- The probability that $\mathcal{F}_{\mathcal{II}}$ asks to replace the public key of the ID_t is $\frac{q_{pkR}}{q_{H_0}}$.

So, the success probability that $\mathcal{C}$ can win the EUF-eCLSC-TKEM-CMA-II game is

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

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

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

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

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

Proof: 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 $\mathcal{F}_{\mathcal{III}}$ be a forger who is able to break the EUF-eCLSC-TKEM-CMA-III security of the eCLSC-TKEM. $\mathcal{C}$ can utilize $\mathcal{F}_{\mathcal{III}}$ to compute the solution b of the ECDLP instance by playing the following interactive game with $\mathcal{F}_{\mathcal{III}}$. 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 $H_i (0 \leq i \leq 3)$ 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 $\mathcal{F}_{\mathcal{III}}$. In order to avoid the inconsistency between the responses to the hash queries, $\mathcal{C}$ maintains lists $L_i (0 \leq i \leq 3)$). 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: $\mathcal{F}_{\mathcal{III}}$ may make a series of 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 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-eCLSC-TKEM-CCA2-III game.

Forgery: Eventually, $\mathcal{F}_{\mathcal{III}}$ 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 $h_i^* \neq h_i$ on the same τ as done in forking lemma [39]. 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.

$$\begin{aligned} W^* \cdot P - W \cdot P &= h_i^* \cdot U - h_i \cdot U \\ &\Rightarrow (W^* - W)P = (h_i^* - h_i) \cdot U \\ &\Rightarrow (W^* - W)P = (h_i^* - h_i) \cdot bP \\ &\Rightarrow (W^* - W) \cdot (h_i^* - h_i)^{-1} = b \end{aligned}$$

Therefore, $\mathcal{F}_{\mathcal{III}}$ 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 $\mathcal{F}_{\mathcal{III}}$ can ask q_C create (ID_i) queries, q_E key-encapsulation queries and q_{H_i} random oracle queries to H_i $(0 \leq i \leq 3)$. We also assume that $\mathcal{F}_{\mathcal{III}}/$ never repeats H_i $(0 \leq i \leq 3)$ a query with the same input.

- The success probability of the Create(ID_i) query execution is $(1 - \frac{q_{H_0}}{q})^{q_C} \geq 1 - \frac{q_{H_0} \cdot q_C}{q}$.
- The success probability of the H_2 query execution is $(1 - \frac{q_{H_2}}{q})^{q_{H_2}} \geq 1 - \frac{q_{H_2}^2}{q}$.
- The success probability of the H_3 query execution is $(1 - \frac{q_{H_3}}{q})^{q_{H_3}} \geq 1 - \frac{q_{H_3}^2}{q}$.
- The success probability of the key encapsulation query execution is $\frac{q_E}{(1-\frac{1}{q})} \geq q_E \cdot (1 + \frac{1}{q})$.
- 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

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

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

APPENDIX B
SECURITY PROOF OF CL-MRES

If we remove the functionality of the digital signature from eCLSC-TKEM, the well-known hybrid encryption scheme with certificateless approach, called certificateless hybrid encryption scheme (CLHES) can be constructed. A certificateless hybrid encryption scheme for multi-receivers called CL-MRES is constructed by applying the random re-use technique to CLHES.

A DEFINITION OF REPRODUCIBLE CERTIFICATELESS HYBRID ENCRYPTION

Fix a certificateless hybrid encryption scheme CLHES=(SetUp, KeyGen, HybridEncryption, HybridDecryption). Let R be an algorithm that takes as input common public parameters, a full public key, and a ciphertext of a random message, another random message together with a full public-private key pair, and returns a ciphertext. Consider the following experiment.

```
**Experiment** $Exp_{CLHES,R}(k)$
$\Omega \leftarrow$ SetUp$(1^k)$; $(pk, sk) \leftarrow$ KeyGen$(\Omega)$
$M \leftarrow \{0, 1\}^*$; $s \leftarrow \mathbb{Z}_q^*$
$C \leftarrow$ HybridEncryption$(\Omega, s, M, pk)$
$(pk', sk') \leftarrow$ KeyGen$(\Omega)$; $M' \leftarrow \{0, 1\}^*$
If HybridEncryption$(\Omega, s, M', ID', pk') = R(\Omega, ID, pk, C, M', ID', pk', sk')$ then return 1.
Else return 0.
EndIf
```

We say that CLHES is reproducible if for any k there exists a random polynomial time algorithm R, called a reproduction algorithm, such that $Exp_{CLHES,R}(k)$ outputs 1 with probability equal to 1.

Lemma 7: Our certificateless hybrid encryption scheme CLHES=(SetUp, KeyGen, HybridEncryption, HybridDecryption) is reproducible.

Proof: On input $(\Omega, pk, (s \cdot P, C), M', ID', t', pk', sk')$, where $\Omega = (F_q, E/F_q, G_q, P, P_{pub}, H_0, H_1, H_2, H_3)$, $pk = (x \cdot P, r \cdot P)$, $pk' = (x' \cdot P, r' \cdot P)$, $sk' = (x', d')$, we can construct a reproduction algorithm R returns as follows:

```
  $\boldsymbol{R}(\Omega, pk, (s \cdot P, \tau), M', ID', t', pk', sk')$
Parse $\Omega$ as $(F_q, E/F_q, G_q, P, P_{pub}, H_0, H_1, H_2, H_3)$
$Y' \leftarrow R' + H_0(ID', R', P', t') \cdot P_{pub} + P'$, $T' \leftarrow (d' + x') \cdot (s \cdot P)$
$K' \leftarrow H_1(Y', s \cdot P, T', ID', P')$
$\tau' \leftarrow ENC_{K'}(M')$
Return $(s \cdot P, \tau')$
```

R first computes a secret key K' for a symmetric encryption algorithm ENC by using given $s \cdot P$.

We can know the output of HybridEncryption on input $(\Omega, s, M', ID', pk')$ is $(s \cdot P, \tau')$ as follows:

```
  **HybridEncryption**$(\Omega$**, s, M', ID', pk'**$)$
Parse $\Omega$ as $(F_q, E/F_q, G_q, P, P_{pub}, H_0, H_1, H_2, H_3)$
Given $s$, $V \leftarrow s \cdot P$
Parse $pk'$ as $(R', P', t')$
$Y' \leftarrow R' + H_0(ID', R', P', t') \cdot P_{pub} + P'$
$T' \leftarrow s \cdot Y'$; $K' = H_1(Y', V, T', ID', P')$
$\tau' \leftarrow ENC_{K'}(M')$
Return $(V(= s \cdot P), \tau')$
```

Therefore, it is easy to see that the experiment $Exp_{CLHES,R}(k)$ always outputs 1.

According to Bellare et al.'s reproducibility theorem [22], we have the following theorem.

Theorem 3: Fix a certificateless hybrid encryption scheme CLHES=(SetUp, KeyGen, HybridEncryption, HybridDecryption). Let CL-MRES be the associated certificateless multi-recipient encryption scheme obtained by applying the random re-use technique. If CLHES is reproducible and provides IND-CCA2 security against an adaptive chosen-ciphertext and identity attack, then the corresponding CL-MRES is IND-CCA2 secure under the same assumption that the one-sided gap Diffie-Hellman (OGDH) problem is intractable.

Proof: We know that CLHES is IND-CCA2 secure in a single-receiver setting due to the IND-CCA2 security of eCLSC-TKEM (see the Theorem 1 in Appendix A). Thus, the Theorem 3 is proved based on Theorem 1 and Lemma 7.

APPENDIX C SECURITY PROOF OF CLDA

Our CLDA is a variant signed ElGamal encryption [35] combining EC-ElGamal encryption with the signing function of our eCLSC-TKEM. The output message of the **Sensor data encryption** step of the CLDA scheme is an EC-ElGamal ciphertext together with the eCLSC-TKEM based signature of that ciphertext. So, the security of CLDA is based on the unforgeability of eCLSC-TKEM and the confidentiality of the EC-ElGamal encryption. Here, the confidentiality is defined as indistinguishability under chosen plaintext attacks (IND-CPA) while unforgeability is defined as existential unforgeability against adaptive chosen messages and identity attacks (EUF-CMA). eCLSC-TKEM is EUF-CMA secure under the assumption that the elliptic curve discrete logarithm problem (ECDLP) is intractable (See the Theorem 2 in Appendix A). Moreover, it is not possible to forge or expose the full private key of an entity based on the difficulty of ECDLP without the knowledge of both KGC's master private key and an entity's secret value. EC-ElGamal encryption is IND-CPA (indistinguishability under chosen plaintext attack) secure under the decisional ECDH (elliptic curve Diffie-Hellman) assumption: given a pair (xP, yP) of uniformly random and independent group elements it is hard to tell xyP from another independent, uniformly random group element zP [40].

REFERENCES

[1] J. Won, S.-H. Seo, and E. Bertino, "A secure communication protocol for drones and smart objects," in *Proc. ACM ASIACCS*, 2015, pp. 249–260.

[2] "Nokia solutions for public safety." (2016). [Online]. Available: https://networks.nokia.com/public-safety

[3] "PrecisionHawk, end-to-end aerial data platform." (2016). [Online]. Available: http://www.precisionhawk.com

[4] J. P. Lynch and K. J. Loh, "A summary review of wireless sensors and sensor networks for structural health monitoring," *Shock Vibrat. Dig.*, vol. 38, no. 2, pp. 91–130, 2006.

[5] D. Boneh and M. K. Franklin, "Identity-based encryption from the Weil pairing," in *Proc. CRYPTO*, 2001, pp. 213–229.

[6] L. Zhou *et al.*, "Supporting secure communication and data collection in mobile sensor networks," in *Proc. INFOCOM*, 2006, pp. 213–229.

[7] H. Song *et al.*, "Least privilege and privilege deprivation: Toward tolerating mobile sink compromises in wireless sensor networks," in *Proc. ACM TOSN*, 2008, p. 23.

[8] A. Rasheed and R. N. Mahapatra, "Secure data collection scheme in wireless sensor network with mobile sink," in *Proc. IEEE NCA*, 2008, pp. 332–340.

[9] A. Rasheed and R. N. Mahapatra, "The three-tier security scheme in wireless sensor networks with mobile sinks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 5, pp. 958–965, May 2012.

[10] C. Schurgers *et al.*, "Optimizing sensor networks in the energy-latency-density design space," *IEEE Trans. Mobile Comput.*, vol. 99, no. 1, pp. 70–80, Jan./Mar. 2002.

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

[12] G. Yang and C.-H. Tan, "Strongly secure certificateless key exchange without pairing," in *Proc. ACM ASIACCS*, 2011, pp. 71–79.

[13] H. Sun, Q. Wen, H. Zhang, and Z. Jin, "A novel pairing-free certificateless authenticated key agreement protocol with provable security," *Frontiers Comput. Sci.*, vol. 7, no. 4, pp. 544–557, 2013.

[14] S. S. D. Selvi, S. S. Vivek, and C. Rangan, "Certificateless KEM and hybrid signcryption schemes revisited," in *Proc. ISPEC*, 2010, pp. 294–307.

[15] S.-H. Seo, J. Won, and E. Bertino, "pCLSC-TKEM: A pairing-free certificateless signcryption-tag key encapsulation mechanism for a privacy-preserving IoT," *Trans. Data Privacy*, vol. 9, no. 2, Aug. 2016, pp. 101–130.

[16] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," in *Proc. ASIACRYPT*, 2003, pp. 452–473.

[17] K.-A. Shim, "A survey of public-key cryptographic primitives in wireless sensor networks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 577–601, 1st Quart., 2016.

[18] D. He, J. Chen, and J. Hu, "A pairing-free certificateless authenticated key agreement protocol," *Int. J. Commun. Syst.*, vol. 25, no. 2, pp. 221–230, 2012.

[19] M. Geng and F. Zhang, "Provably secure certificateless two-party authenticated key agreement protocol without pairing," in *Proc. CIS*, 2009, pp. 208–212.

[20] F. Li, M. Shirase, and T. Takagi, "Certificateless hybrid signcryption," in *Proc. ISPEC*, 2009, pp. 112–123.

[21] K. Kurosawa, "Multi-recipient public-key encryption with shortened ciphertext," in *Proc. PKC*, 2002, pp. 48–63.

[22] M. Bellare, A. Boldyreva, and J. Staddon, "Randomness re-use in multi-recipient encryption schemeas," in *Proc. PKC*, 2003, pp. 85–99.

[23] N. P. Smart, "Efficient key encapsulation to multiple parties," in *Proc. Secur. Commun. Netw. (SCN)*, 2004, pp. 208–219.

[24] M. Barbosa and P. Farshim, "Efficient identity-based key encapsulation to multiple parties," in *Proc. IMACC*, 2005, pp. 428–441.

[25] A. Pinto, B. Poettering, and J. C. Schuldt, "Multi-recipient encryption, revisited," in *Proc. ACM ASIACCS*, 2014, pp. 229–238.

[26] S. Ben Othman, A. Trad, H. Alzaid, and H. Youssef, "Performance evaluation of EC-ElGamal encryption algorithm for wireless sensor networks," in *Proc. MobiHealth*, 2013, pp. 271–285.

[27] O. Ugus, D. Westhoff, R. Laue, A. Shoufan, and S. A. Huss, "Optimized implementation of elliptic curve based additive homomorphic encryption for wireless sensor networks," in *Proc. WESS*, 2007, pp. 11–16.

[28] S. Antão, J.-C. Bajard, and L. Sousa, "RNS-based elliptic curve point multiplication for massive parallel architectures," *Comput. J.*, vol. 55, no. 5, pp. 629–647, 2011.

[29] J. W. Bos, "Low-latency elliptic curve scalar multiplication," *Int. J. Parallel Program.*, vol. 40, no. 5, pp. 532–550, 2012.

[30] S. Cui, J. Großschädl, Z. Liu, and Q. Xu, "High-speed elliptic curve cryptography on the NVIDIA GT200 graphics processing unit," in *Proc. ISPEC*, 2014, pp. 202–216.

[31] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.

[32] Certivox. (2014). *MIRACL Cryptographic SDK*. [Online]. Available: http://www.certivox.com/miracl/

[33] Nvidia. (2015). *Jetson TK1*. [Online]. Available: http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html

[34] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proc. ACM SenSys*, 2004, pp. 95–107.

[35] C. Schnorr and M. Jakobsson, "Security of signed ELGamal encryption," in *Proc. ASIACRYPT*, 2000, pp. 73–89.

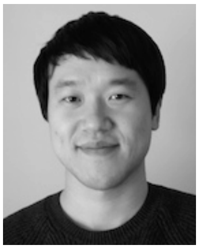
[36] AR.Drone2.0. (2013). [Online]. Available: http://ardrone2.parrot.com

[37] A. Liu and P. Ning, "TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks," in *Proc. IPSN*, 2008, pp. 245–256.

[38] N. Koblitz and A. Menezes, "Intractable problems in cryptography," in *Proc. 9th Int. Conf. Finite Fields Appl.*, 2010, pp. 279–300.

[39] D. Pointcheval and J. Stern, "Security arguments for digital signatures and blind signatures," *J. Cryptol.*, vol. 3, no. 3, pp. 361–396, 2000.

[40] Y. Tsiounis and M. Yung, "On the security of ELGamal based encryption," in *Public Key Cryptography* (Lecture Notes in Computer Science), vol. 1431. Berlin, Germany: Springer, 1998.



JONGHO WON received the B.S. and M.S. degrees from Seoul National University, South Korea, in 2006 and 2008, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science, Purdue University, Indiana, USA. He was a Research Engineer with the LG Electronics Advanced Research Institute for three years. Since 2011, he has been with the Department of Computer Science, Purdue University. His main research interests include secure communication protocols, security/privacy in wireless networks, and smart grids.

SEUNG-HYUN SEO received the B.S., M.S., and Ph.D. degrees from Ewha Womans University, South Korea, in 2000, 2002, and 2006, respectively. She was a Post-Doctoral Researcher of Computer Science with Purdue University, Indiana, USA, for two and half years, a Senior Researcher with Korea Internet and Security Agency for two years, and a Researcher with Financial Security Agency, South Korea, for three years. She was an Assistant Professor with Korea University Sejong campus for two years. She is currently an Associate Professor with Hanyang University, South Korea. Her main research interests include cryptography, Internet of Things security, mobile security, secure cloud computing, and malicious code analysis.

ELISA BERTINO (F'02) was a Faculty Member with the Department of Computer Science and Communication, University of Milan, Italy. She is currently a Professor of Computer Science with Purdue University, Indiana, USA, and also serves as the Director of the Purdue Cyber Center (Discovery Park) and as the Research Director of CERIAS. Her main research interests include security, privacy, digital identity management systems, database systems, distributed systems, and multimedia systems. She is a Fellow of the ACM. She received the 2002 IEEE Computer Society Technical Achievement Award for outstanding contributions to database systems, database security, and advanced data management systems and the 2005 IEEE Computer Society Tsutomu Kanai Award for pioneering and innovative research contributions to secure distributed systems.